A Study on Distributed Frequent Co-occurrence Patterns Algorithms across Multiple Data Streams

Jing Guo*
School of Computer science and Information Engineering, Chongqing Technology and Business University, Chongqing, China.

* Corresponding author. Tel.: +86-13638309422; email: dorothy-guo@163.com
Manuscript submitted July 10, 2016; accepted August 23, 2016.
doi: 10.17706/jsw.11.12.1191-1198

Abstract: With the era of big data coming, the data streams are fast, continuous, and unbounded. The real-time requirements of the data streams processing results are very high. A large number of researches have been on Frequent Co-occurrence Patterns across multiple data streams. But those algorithms are centralized, which is worked on a single compute node. The memory of a single compute node and CPU calculation can be limited, which is difficult to deal with the increasing data streams. Using the distributed server cluster is an effective way. However, the centralized algorithm cannot be directly deployed to distributed server cluster. This paper designs a Distributed Frequent Co-occurrence Pattern across multiple data streams to solve these problems. Through a lot of experiments to evaluate it, the algorithm can detect all the objects that meet the conditions in real time, and have good scalability. In order to save memory, this paper also improves the algorithm, and proposes Modified Distributed Frequent Co-occurrence Pattern based on P-condition deletion strategy. The improved algorithm can delete element combinations which can not constitute Frequent Co-occurrence Patterns in the initial stage, so as to effectively save memory.

Key words: Distributed, frequent Co-occurrence pattern, multiple data streams, segment, MDFCP.

1. Introduction

Frequent Co-occurrence pattern discovery problem is a classical problem in the field of data mining. The frequent Co-occurrence Pattern refers to a set of objects occurs in one data stream within a short time span and this set of objects appears in multiple data streams in the same fashion within another user-specified time span. Frequent Co-occurrence Pattern can usually represent the intrinsic relationship between the elements [1]. The real time discovery of Frequent Co-occurrence Pattern has been applied in many areas such as the prevention of crime, the discovery of new hot spots, location information services and e-commerce. Frequent pattern mining on multiple data streams is also very important for many applications. A lot of research and high performance algorithms about it have emerged. In the context of the current big data, the size of the flow data is also growing rapidly, and the flow data is very fast, and the real-time requirements of the processing results are very high. The existing Frequent Co-occurrence Pattern algorithms have achieved very good results [2] in the memory costs, Index Maintenance costs, and frequent patterns discovery efficiency. But the existing Frequent Co-occurrence Pattern is centralized, which is worked only on a single compute node [3], [4]. The memory of a single compute node and CPU calculation can be limited, which is difficult to deal with the increasing data streams. In order to solve this problem, it is an effective way to use the distributed server cluster. However, the current centralized algorithms can't be

deployed directly to the distributed server cluster. Therefore, it is very important to design a frequent pattern discovery algorithm based on distributed server cluster. This paper designs Distributed Frequent Co-occurrence Pattern (DFCP) algorithm and Modified Distributed Frequent Co-occurrence Pattern (MDFCP) algorithm. The two algorithm can be easily deployed on the distributed server cluster, and realize Frequent Co-occurrence Pattern across data streams.

2. DFCP Algorithm

2.1. Relevant Definition

Definition 1. (Data Stream) *A data stream is a continuous ordered (by timestamps) sequence of objects.* Data streams are unbounded, and thus it is infeasible to store the data stream locally in its entirety. For an object o_i in the data stream, its timestamp are t_i respectively[5].

Definition 2. (Co-occurrence Pattern, CP) *For a set of objects* $O = \{o_1, o_2, \dots o_k\}$ *that appear in a data stream si, we say that O is a co-occurrence pattern* CP_k *(where k is the number of objects) if* $t_{max}^{O^{s_i}} - t_{min}^{O^{s_i}} \leq \xi$, *where* $t_{max}^{O^{s_i}} = \max\{t_1, \dots, t_k\}$, $t_{min}^{O^{s_i}} = min\{t_1, \dots, t_k\}$, *and* ξ *is a user-specified threshold.*

Definition 3. (Frequent co-occurrence pattern, or FCP) *Given a set of unbounded streams of objects* s_i *(i=1, 2, ..., n), a set of objects* $O = \{o_1, o_2, \dots o_k\}$ *is called a frequent co-occurrence pattern(FCP) if (1) they appear in at least* θ *streams within a period of time no longer than* τ, *and (2) their appearance within each of those streams happens within a time windows of size no greater than* ξ, *where* τ, ξ, *and* θ *are user-specified thresholds* [2], [6].

Definition 4. (Segment) For a given data stream s, a segment $G(o_1o_2...o_m)$ is a subsequence of s that satisfies both of the following conditions: (1) $\left|\boldsymbol{t_i} - \boldsymbol{t_j}\right| \leq \xi$, o_i, o_j is any two elements in G; (2) The time span of G must be maximal with respect to $\boldsymbol{\xi}$. That is, that does not exist a subsequence of s, G', such that G' is a segment and G is a strict subsequence of G' [7].

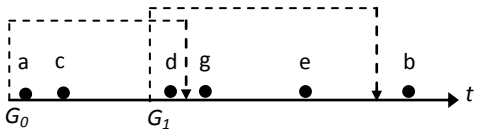

Fig. 1. A segment.

The temporal relationship between the objects in figure 1 is as follows: $t_d - t_a < \xi$, $t_g - t_a > \xi$, $t_g - t_d < \xi$, $t_g - t_c > \xi$, $t_e - t_d < \xi$, $t_b - t_d > \xi$. Figure 1 shows a stream of objects, where *a* is the first object of the segment G_0. Since $t_d - t_a < \xi$ and $t_g - t_a > \xi$, *d* is the last object of the segment G_0. Note that the objects *c*, *d*, and *g* do not constitute a segment because $t_g - t_c > \xi$.

DEFINITION 5. (Prefix of the segment) *For a given segment Gi(o1o2...om), its prefix is a subsequence of G, that is G'(o1o2...oj), where* $1 \leq j \leq m$.

2.2. DFCP Algorithm Description

To facilitate the search for FCPs, we divide each stream into overlapping segments. Each segment is a sequence of objects ordered by their timestamps, and the time span of a segment is the time interval between its first and last objects. Any data stream can be uniquely partitioned. In addition, any co-occurrence pattern must be covered by some segment(s). Therefore, the problem of mining FCPs across multiple streams can be converted to finding the FCPs contained in the segments of the data streams.

Each newly generated segment may form a new FCP. Therefore, each object in the segment also may form a new FCP. In order to detect all possible FCP, DFCP generates all object combinations which may be a FCP

on a segment as a unit. Assuming that the number of objects in a FCP is k, then each segment will generate all combination using k objects which is called a candidate FCP (C-FCP). For example, a segment is $\{o_1, o_2, o_3\}$, and k=2, C-FCPs is $\{(o_1, o_2), (o_1, o_3), (o_2, o_3)\}$.

In DFCP, the FCP-INDEX is designed. The index's idea is to first take the C-FCP as the index unit (IU), and then make the same C-FCP from the different server computing nodes to arrive at the same IU. In this case, an IU of the FCP-INDEX records the same C-FCPs, which may come from different segment fragments of different data streams. DFCP will judge whether the C-FCPs found out with FCP-INDEX is FCPs, when any arbitrary index unit receives a C-FCP. If the difference between the latest reach time about a C-FCP in an IU (IU_i) and the current time is greater than θ, and moreover, it is not co-occurrence object, the index unit will be deleted. In this situation, all C-FCPs of the IU_i is not likely to be co-occurrence. Due to each other IU of FCP-INDEX is mutual independent, FCP-INDEX is a distributed index structure. It can be deployed on any number of computing nodes. Each computing node does maintenance any number of FCP-INDEX's IUs according to itself workload capacity. Therefore the index structure has a good scalability.

The non-code description of the DFCP algorithm is shown below.

```
Algorithm: DFCP
Require: Sliding window data, θ, τ, k
Ensure: co-occurrence data combination
1: Sliding window data of each data entrance is received from the DTS-Window
algorithm;
2: According to the object of the window, the C-FCPs of number k are generated;
3: While any new C-FCP then
4:    If existing IU can index the C-FCP then
5:          Add the C-FCP to IU;
6:          If the number of C-FCPs in IU ≥ θ and
                the time interval between the first C-FCP and the last C-FCP ≤ τ then
7:          The C-FCPs is a set of co-occurrence objects;
8:          End If
9:     If the current time - the time for the latest C-FCP added to IU  >τ then
10:              Delete IU;
11:         End If
12:    Else
13:       Create a new index unit for C-FCP;
14:    End If
15: End While
```

With the arriving of continued data, the DFCP algorithm maintains each IU in real time, and determines whether each C-FCP in IU is a FCP. Since FCP-INDEX can be deployed to any number of computing nodes, the DFCP algorithm is a distributed algorithm that can run on multiple physical machines. When data streams increases, we can increase computing nodes to run DFCP algorithm, which can improve the algorithm's throughput capacity to achieve real-time detection of FCP across the data streams.

Fig. 2 is an example of the DFCP algorithm. It is assumed that we want to find out 2 FCP. In this graph, each data entrance corresponds to a data stream. The time of a segment is ξ. During ξ, there are three objects (A, B, C) through data entrance P1, then P1 produces three C-FCPs (AB, AC and BC). Similarly, P2 and P3 also produce three respective C-FCPs. Each IU receives the same C-FCP. As a result, the same C-FCP from different data entrance (i.e., different data streams) will be handled by the same IU. According to the definition of FCP, if IUi can handle C-FCPs from more than or equal to θ data entrances, the C-FCPs is a set of FCP. If θ equal 2 and the time of IU1 handled three C-FCPs (AB) is less than τ, then the object (AB) is a FCP.

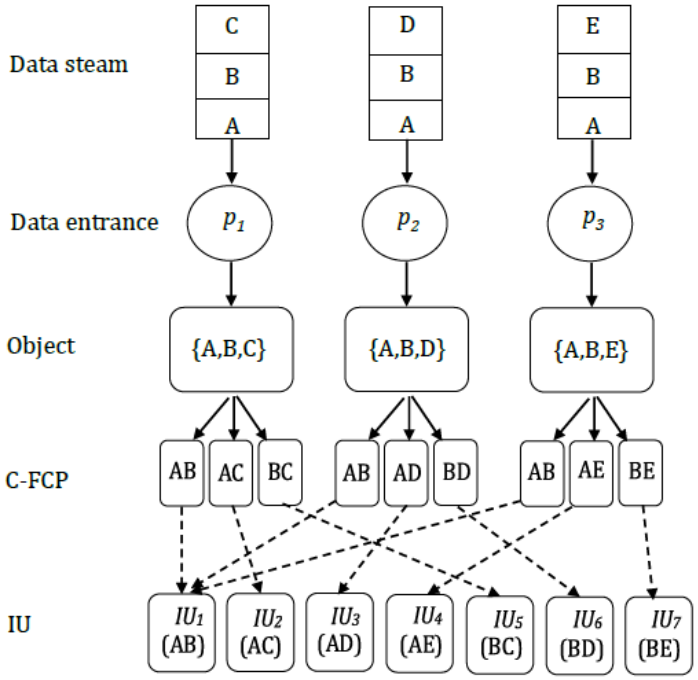

Fig. 2. FCP-DISCOVER algorithm architecture.

2.3. DFCP Algorithm Analysis

Spatial analysis: In order to ensure DFCP is real-time, every IU of FCP-INDEX must be stored in memory. So the use of memory is one of the important problems in this paper. Calculate the amount of memory used for a period of time is to calculate the number of C-FCPs. Due to a C-FCP was deleted after at least time of τ, the calculation of memory usage during time τ can reflect the memory usage when running DFCP algorithm. Assume that each segment has m objects ($m<n$), then the initial sliding window can generate C_m^k C-FCPs[2]. Assume that segments are capable to displace, and a new object will be increased after a displacement. When a segment is displaced, there is at least one object (the first object) to be timed out. New C_{m-1}^{k-1} C-FCPs will be generated after window displacement. The window will be displaced n-m times during time of β. So the total number of C-FCPs during time of β is $p * [C_m^k + (n-m) * C_{m-1}^{k-1}]$.

Time cost of window displacement: Assuming that segment S contains m objects, when S is displaced, time of t_a will be added as a new object's time. When window S deletes outdated data, it needs to track m objects at most, which is needs times of $m*t_d$. So a displacement time of window is t_a+m*t_d [8].

Time cost of window S generating C-FCPs: Assuming C-FCPs have k objects, and the time for generating a C-FCP is t_p, then the total time of window S generating all C-FCPs is $C_m^k * t_p$.

Time cost of IU handling C-FCPs: When IU received a C-FCP, determining whether the C-FCP is a FCP or not needs only two times of compare, which's consumption time is t_c. The time of deleting outdated C-FCP is t_e.

Therefore, the total time of DFCP algorithm to deal with an object is $t_a + m * t_d + C_m^k * (t_p + t_c + t_e)$ [9] at most. As t_a, t_d, t_p, t_c, t_e are constants, the total time of DFCP is proportional to m (i.e., the length of segment) in the case of a certain k.

3. MDFCP Algorithm

DFCP algorithm needs to store a large amount of C-FCPs, which leads to a lot of memory overhead, even though the algorithm can detect all FCPs across multiple data streams in real time. In order to reduce the number of C-FCPs, we optimize DFCP algorithm, which is denoted as MDFCP algorithm.

In MDFCP algorithm, the objects of any C-FCPs are sorted by the sequence of the dictionary, such as C-FCP (B) should be recorded as AB. In order to reduce the number of C-FCPs to be searched, the C-FCP deletion condition (P-Condition) is introduced. For any C-FCP of data entrance p_i, o_i is the first object in it. If the data entrances which o_i appeared is $\Gamma=\{p_j\}(1\leq j\leq m$,and $j\neq I$), and a C-FCP is found at n numbers of data entrances in Γ, then the C-FCP can be deleted.

P-Condition mainly aimed at C-FCPs that has appeared but the probability of appearing again is very small. Deleting these C-FCPs in time can reduce the memory cost to a great extent. Although P-Condition will cause a handful of co-occurrence objects is accidentally deleted, but many applications allow the minimum error probability, such as electronic business sites permit a minimum of error in exchange for higher query efficiency in the detection of goods. Through the adjustment of ε and λ, we can avoid mistakes. For example, when $m=|P|$, P-Condition will not cause the deletion by mistake, where $|P|$ represents the number of all data entrances.

Since P-Condition is difficult to be applied to the FCP-INDEX structure, MDFCP is no longer used a separate CAD as IU. MDFCP builds a two-level inverted index (TI-INDEX) based on memory for all C-FCPs. The first level index of TI-INDEX is indexing all C-FCPs using IU which the first letter is C-FCP. The second level index of TI-INDEX is recording the identification of the data stream for each C-FCP. For example, if the given C-FCPs are {AB(1), AC(1), AB(2), BC(2), BD(3), BD(4), AB(4), BD(5)}, the TI-INDEX is shown in Fig. 3.

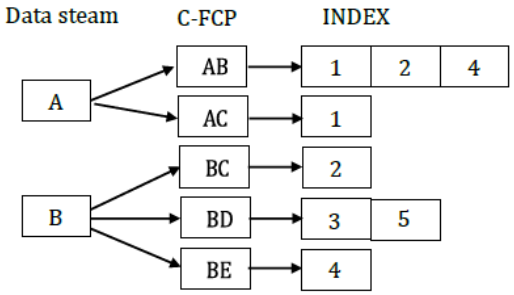

Fig. 3. An Example of TI-INDEX.

The idea of MDFCP is when the new C-FCP arrives at an IU a_i found by the first level index of TI-INDEX. MDFCP tracks all C-FCP in a_i. If a C-FCP satisfies the definition of FCP, the C-FCP is a co-occurrence object. If a C-FCP satisfies P-Condition, then the CAD will be deleted. In the example in Fig. 1, it is assumed that θ=3, and ε=3, λ=1/3(in P-Condition). When AB(4) arrives, AB satisfies the definition of FCP, so AB is a FCP. When BD(5) arrives, BC(2) satisfies the definition of P-Condition, so BC(2) will be deleted.

Compared with the DFCP algorithm, the MDFCP algorithm has the following advantages:

(1) MDFCP introduces the P-Condition deletion condition, which can delete some unsuitable C-FCPs and save memory space greatly. By adjusting the parameter ε and λ, we can balance algorithm efficiency and recall radio goodly.

(2) MDFCP builds TI-INDEX based on memory, which's IUs are independent of each other. It is easy to deploy to multiple servers on a distributed platform, and has good expansibility. Compared with the common inverted index, TI-INDEX has no overlapping, which saves the memory cost. TI-INDEX supports P-Condition delete conditions goodly, and improves the efficiency of the MDFCP algorithm.

4. Algorithm Implementation Based on S4 Platform

S4 is an open source distributed data streams processing platform released by Yahoo in 2010. S4 uses the role model, in which the data between the systems flow in the way of *event*[10]. Each *event* can be expressed as *<event-type, key-value pair, message>*. The most basic operation of S4 is Processing Element (PE). A PE is capable of selectively handling events generated by other PE, and sends out events that can contain the new message, which is able to be processed by other PE. In the S4 platform, PE is designed to be lightweight. So a server (a computing node) can support millions of PE operations.

DFCP and MDFCP are deployed to the S4 platform to achieve real-time distributed frequent co-occurrence objects detection platform. The deployment process of the two algorithms is similar. Each data entrance is responsible for a PE of S4. According to the segment fragment partitioning strategy, data entrance PE divides data streams and generates the corresponding C-FCP. Each IU of FCP-INDEX is also managed by a PE in S4, which is called index PE. Index PE determines whether the corresponding FCP is C-FCP by the definition of FCP. Data entrance PE and index PE are automatically generated by the S4.

5. Experiments

This paper uses co-occurrence vehicles in the traffic field to test the performance of DFCP algorithm and MDFCP algorithm.

5.1. Experimental Setup

8 DELL R210 servers are used in experiments. Each server is configured with a dominant frequency of Intel 2.4GHz processor and 8G memory. These servers are connected through an Ethernet with 1G bandwidth. The distributed platform uses S4-0.6.0 version. In experiments, traffic records from 1093 traffic crossings one day is used as the test data streams.

5.2. Experimental Content

Firstly, we tested the performance of the two algorithms. 1093 data entrances were simulated on 8 nodes. 3 million 200 thousand passing records during morning rush hour 7:00-9:00 were sent to the processing platform to test the system's resistance at 10,000/s, 30,000/s, 50,000/s, 70,000/s, 90,000/s rate respectively. In the following experiments, the relevant parameters of DFCP was set as ξ=60s, τ=6h, θ=4, k=3, and the relevant parameters of MDFCP parameter was set as ε=3, λ=1/3.

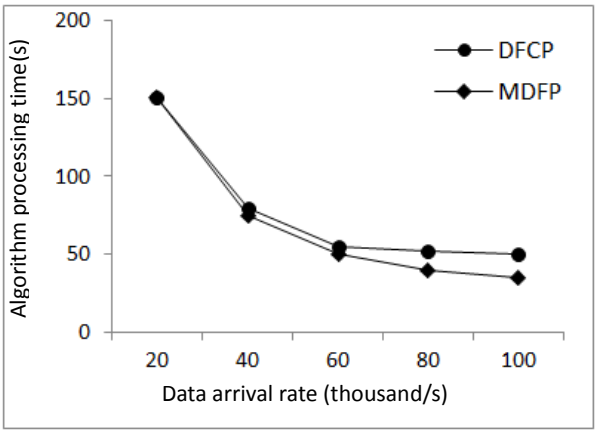

Fig. 4. Comparison of two algorithms processing time.

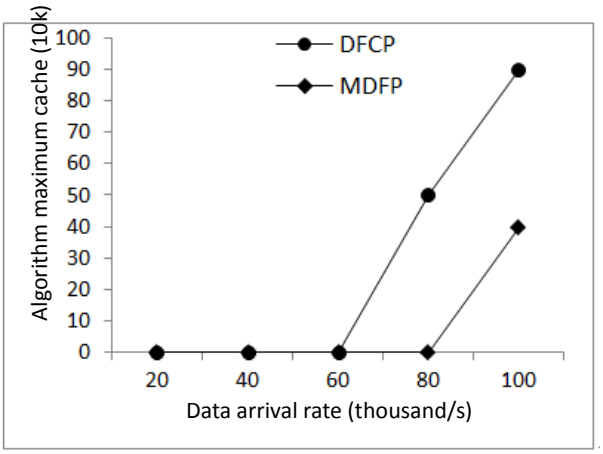

Fig. 5. Comparison of two algorithms maximum cache.

In Fig. 4 and Fig. 5, two algorithms are given respectively to deal with the changes of the time and the maximum cache of all traffic records in the case of different data arrival rates. In Fig. 4, when data arrival rate is less than 50,000/s, processing times of two algorithms decrease with improvement of the arrival rate.

When the data arrival rate is greater than 50,000/s, MDFCP processing time is still down, but DFCP processing time is almost no longer drop. So the performance of MDFCP is better than DFCP. In Fig. 5, when data arrival rate is less than 50,000/s, because the data were processed by a rate greater than the data arrival rate, the number of traffic records in system cache tends to 0. When the rate is greater than 50,000/s, the maximum cache of DFCP starts to increase obviously. This shows that the maximum processing power of DFCP is 50,000/s. It can be seen also in Fig. 5 that the maximum processing power of MDFCP is 70,000/s.

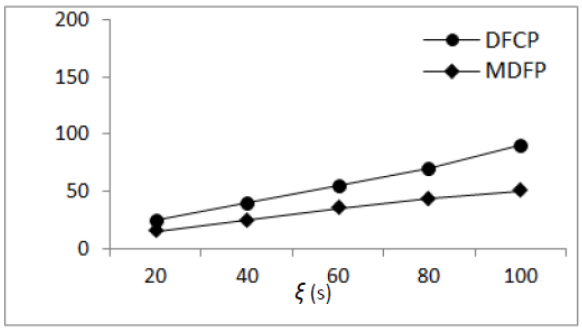

Fig. 6. Processing time w.r.t. ξ.

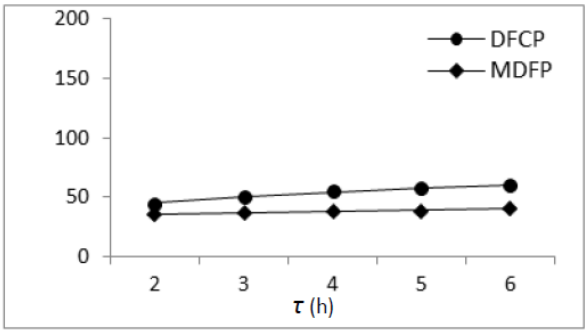

Fig. 7. Processing time w.r.t. τ.

Fig. 6 and Fig. 7 represent the effects of parameter ξ, τ on the processing time of two algorithms. The number (m) of objects in the segment is decided by ξ. m is greater if ξ is greater. In Fig. 6, the two algorithms' processing time increases with the increase of ξ. We can conclude algorithm processing time is proportional to m. The experiment results prove the conclusion in the foregoing algorithm time analysis. MDFCP can delete some C-FCPs according to P-condition, so ξ 's impact on MDFCP is less than that of DFCP. In Fig. 7, the DFCP algorithm processing time increases with increasing of the parameters τ. Although τ *makes* the number of C-FCPs in memory to increase, but its impact on the processing time is smaller than ξ 's impact (in Fig. 6). Since MDFCP can delete redundant C-FCPs in time, it is almost not affected by the influence of τ.

Secondly, we tested how the related parameters impact on the query results of the co-occurrence vehicles. In the experiments, we assumed that the co-occurrence vehicles passed by crossings of different roads. If the specified crossings is located on the same road, then the algorithm will return too much of the co-occurrence vehicles, which is lack of practical significance. In addition, the processing time in the experiment is the time to finish all the passing records, but the discovery of FCPs is in real time.

6. Conclusions

This paper proposes a distributed data mining algorithm --DFCP algorithm. This algorithm can be easily deployed to the distributed server clusters, which will discover frequent co-occurrence patterns across on multiple data streams. We improve the DFCP algorithm, and propose MDFCP algorithm based on the P-condition deletion strategy, which can save memory effectively. DFCP algorithm and MDFCP algorithm are deployed on the S4 platform. Through a large number of experiments to evaluate their performance, we think two algorithms both have a very good effect. For future work, we would like to study how to use an iterative calculation model instead of a hierarchical model. At that time, we do not need to appoint the maximum length of FCPs. The maximum FCPs can be calculated by a step-by-step iteration method.

References

[1] Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applivations.

[2] Chi, Y., Wang, H., Yu, P. S., & Muntz, R. (2004). Moment: Maintaining closes frequent itemsets over a

stream sliding window.
[3] Guha, S., Meyerson, A., Mishra, N., Motwani, R., & O'Callaghan, L. (2003). Clustering data streams: Theory and practice. *In Tkde Special Issue on Clustering*.
[4] Wang, H., Fan, W., Yu, P., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers.
[5] Mozafari, B., Thakkar, H., & Zaniolo, C. (2008). Verifying and mining frequent patterns from large windows over datastreams.
[6] Guo, J., Zhang, P., & Tan, J. (2011). Mining frequent patterns across multiple data streams.
[7] Yu, Z. Q., Xiao, H. Y., Yang, L., Li, W. Z., & Jian, P. (2015). Mining frequent co-occurrence patterns across multiple data streams. *Proceedings of the 18th Internation Conference on Extending database Technologe* (pp. 23-27)
[8] Kohlborn, T., Korthaus, A., & Rosemann, M. (2009). Business and software service lifecycle management. *Proceedings of the IEEE International Enterprise Distributed Object Computing Conference* (pp. 87-96).
[9] Ficek, M., & Kencl, L. (2010). Spatial extension of the reality mining dataset. *Proceedings of the International Conference on Mobile Adhoc and Sensor Systems* (pp. 666-673)
[10] Cao, H., Mamoulis, N., & Cheung, D. W. (2005). Mining frequent Spatio-temporal sequential patterns.

Jing Guo received the B.S. and M.S. degrees from Southwest Petroleum University, China in 2001 and 2004 respectively. Now she is a Ph.D. candidate in Chongqing University, and a lecturer in School of Computer science and Information Engineering of Chongqing Technology and Business University. Her research interests include computer architecture, information system, data mining and system security.