A Multi - flow Streaming Data Frequent Pattern Mining Adaptive Algorithm

Fan Feng, Husheng Liao and Xueyun Jin

Faculty of Information Technology, Beijing University of Technology, Beijing, China
Email:littlefun_ff@163.com, liaohs@bjut.edu.cn, jinxueyun@bjut.edu.cn

Keywords: Frequent sequential pattern mining, Stream data, Parallel processing, Self-adaption.

Abstract. Frequent sequential pattern mining is an important field in data mining. Compared with the static data, the stream data is a single scan data obtained in a continuous and real-time way. The frequent pattern mining algorithm of traditional static sequence database has been difficult to meet the frequent pattern mining requirements for streaming data. The traditional serial processing method is time-consuming and cannot meet the requirements of high performance processing. Based on the existing Pisa algorithm, this paper presents a parallel algorithm named Parallel-Pisa, it can adjust the parallel strategy according to the different velocity of the stream data to improve the efficiency of the algorithm so that it can be better applied to frequent sequence pattern mining of stream data.

1. Introduction

With the development and application of Internet and communication technology, economic, natural sciences, engineering and other fields have accumulated more and more data. Which has a class of important data, can reflect the data in the context of the relationship, known as time series data. Frequent pattern mining for time series data can identify the periodic and frequently occurring patterns of data in the order of time, helping decision makers to make more informed decisions.

In order to mine frequent patterns in time series data, researchers have made continuous improvement to the frequent pattern mining of static database. The papers [1, 2, 3] use the incremental mining concept so that the algorithm can excavate the newly added data set on the basis of the original static data, which known as incremental mining. At the same time, due to limited storage space or obsolete data is no longer valuable and other reasons, sometimes need to delete the excavation of the object. In order to solve this problem, the papers [4,5,6] use the progressive mining concept to make the algorithm support the increase and deletion of the data set in the mining process. This mining method is called progressive mining. Parallel-Pisa will use the concept of progressive mining to mining time series data to find out the cyclical changes and frequent sequence patterns.

In real life a large amount of time series data are constantly real-time output and difficult to preserve, such as the real-time data of the stock exchange, interactive data in a network and so on. This kind of time series is called stream data. The frequent pattern mining of static data can‘t meet the demand of real-time and timeliness of mining frequent data in the performance. The papers [8,9,10] improves the progressive frequent pattern mining for stream data, but there is still a lot of room for improvement in terms of time consumption and space consumption. The current frequent pattern mining algorithm for streaming data is mainly based on serial processing, which is difficult to adapt to the fast and large flow data, which can't meet the requirement of real-time mining of streaming data, and there is no higher load handling capability to face the explosive changes in convective data.

In view of the above problems, this paper studies an adaptive parallel algorithm for frequent data mining of multi-data flow, and the main contributions are as follows:

1) Implements a frequent pattern mining algorithm for parallel processing, which improves the mining efficiency of the algorithm so that it can carry out frequent pattern mining on multi-data stream data model.
2) Proposes a strategy to adaptive adjust the parallel algorithm so that the algorithm can adjust the explosive growth or sudden decrease of the number of flow data to make it more reasonable to use the system resources

The structure of this paper is as follows: In the second section, we introduce some preliminary knowledge, and the third part introduces the parallel design idea and algorithm principle of the frequent sequence pattern mining. The fourth section introduces the adaptive parallel strategy, and the fifth Section gives the relevant experiment and test results of the algorithm.

2. Preliminaries

2.1. Problem Description

The characteristics of the stream data are: fast, massive, and so on. The aim of our algorithm is to find out the sequence patterns that frequently occur among multiple data streams.

Definition 1(Frequent pattern mining on multiple data streams). Let $I = \{x_1,x_2,....,x_n\}$ be a set of data items. An element set e □ I denoted by $(x_ix_j...)$ is a subset of items which appear at the same time. Sequence $s = <e_1,e_2,...,e_m>$, consists of a series of ordered data items. A sequence $\alpha = <a_1,a_2,...,a_n>$ is a subsequence of another sequence $\beta = <b_1,b_2,...,b_n>$, means α□β, if there exists a set of integers, $1 \le i_1 < i_2 < ... < i_n \le m$. Data flow F_i contains a set of sequences. Multi-flow Frequent Sequence Pattern Mining can be defined as "Given a number of data flow F_1 to F_n and a user-defined minimum support *min_sup*, find the complete set of subsequences whose occurrence times ≥ *min_sup*."

Parallel-Pisa is based on the data structure PS-tree (section 2.2) in [4], and then the frequent pattern mining algorithm of multi-data flow is described in detail in Figure 1(a) as the example in [4].

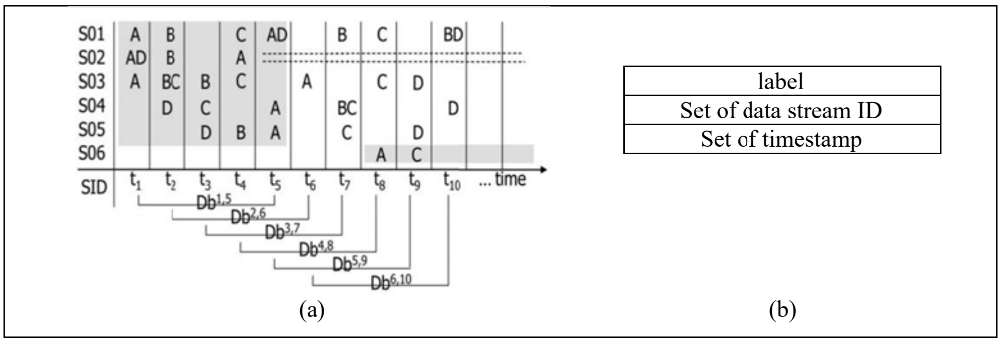

Figure 1 (a) Multi-flow data example. (b) Ordinary node structure.

In Figure 1(a), S01, S02,..., Sn represent different data flows, A, B, C, and D represent different data items, t_1、t_2、...、t_k represent timestamps, the data contained in the timestamp t_i represents a different set of data items *e*. As time goes on, each stream continues to deliver real-time streaming data. $Db^{p,q}$ represents a collection of data items in each data stream between timestamp p and timestamp q.

2.2. PS-tree

This section introduces the PS-tree data structure (PS) which is borrowed from [4]. There are two kinds of nodes in the PS. The root node stores only a series of ordinary nodes as his child nodes. The data structure of the ordinary nodes is given in Figure 1(b). The nodes store the label of the nodes, the data stream of the labels and the timestamp corresponding to the data flow, the data stream ID and timestamp are stored in a set and one-to-one correspondence.

In the process of constructing PS, when the number of data stream stored in a node is not less than the minimum support, it is shown that the sequential pattern represented by the root node to the current node is frequent, and it is regarded as a frequent sequential pattern.

2.3. Algorithm Pisa

Parallel-Pisa is extended on the basis of [4].The main idea of Pisa is to store candidate elements in PS and find out the frequent sequence patterns. Figure 2 shows the processing flow of Pisa.

```
Procedure of Pisa：
Each timestamp $t_i$ :
For each nodes of the PS in post order：
    1)  Check the timestamp information in the PS node and delete the expired node that is not within the time window.
    2)  If the data stream is passed to the element of a node label information，update the data flow within the node and
        the corresponding timestamp.
    3)  If the label of an existing node does not contain a new incoming data item, a new node is added to the PS.
    4)  Output the frequent sequence pattern according to the threshold.
END
```

Figure 2 Procedure of Pisa.

If the Pisa is applied to the flow data model in Figure 1(a), with the increase of the number of nodes in the PS, the time consumed by the serial processing PS is too much. Therefore, due to the performance factor Pisa can't produce good performance in frequent pattern mining of stream data.

This paper proposes a kind of the optimization of PS-tree traversal methods, which uses parallel processing of the primary subtrees in the PS for improving its efficiency.

3. Algorithm Parallel-Pisa

3.1. Detailed Explanation of Algorithm Parallel-Pisa

Figure 3 shows the details of the Parallel-Pisa. The parameters *support* and *LW* are the minimum support and the sliding time window length set by the user respectively. On lines 4-10, if new data arrives, the PS is processed in parallel, and then the timestamp is updated to the latest moment until no new data arrives again. In line 6, the function updates the node information in the primary subtree of the PS-tree in parallel. After the root node has finished processing, the PS has updated the PS for the time t_{i+1} and contains the latest sequence information in the time window. In Figure 4, the **handleRoot** and **handleSubtree** operation are presented.

```
Algorithm ParallelPisa(support,LW)
1. var root                                  //PS-tree PS
2. var currentTime                           // timestamp now
3. var pool                                  // threadPool
4. while (there is still new data)
5.     for (each sub in root)                // sub is the subtree of PS-tree
6.         pool.submit(handleSubtree(sub, data, currentTime, LW, support))
7.         Wait for all subtree tasks to finish
8.     handleRoot(root, data, currentTime, LW, support)
9.     currentTime++
10. pool.shutdown()
END
```

Figure 3 Algorithm Parallel-Pisa.

In Figure 4(a), the every data item set *e* is from data streams at line 1, and then the set of data items in the set *e* are used to update the input of the PS tree at line 2.

```
Algorithm handleRoot(root, data, curtime)
1.  for (data item set e of each flow in data)
2.      for (candidate elements ele in e)
3.          if (ele == lable of on of root)
4.              if (flow is in root.child.flowList)
5.                  update timestamp of flow to curtme
6.              else
7.                  create a new flow with curtime
8.          else
9.                  create a new child with ele, flow and
    curtime
10. return root.
11. END

(a)
```

```
Algorithm handleSubtree(node, data, curtime)
1.  for (every flow in node.flowList)
2.      if (flow.timestamp <= curtime - LW)
3.          delete flow in flowList and continue to next flow
4.      if (there if new items set e of flow in data)
5.          for (candidate elements ele in e)
6.              if (ele is not on the path from root)
7.                  if (ele == label of one of node.child)
8.                      if (flow is in node.child.flowList)
9.                          child.flowList.flow.timestamp = flow.timestamp
10.                     else
11.                         create a new flow with flow.timestamp
12.                 else
13.                     create a child with ele,flow and flow.timestamp
14. if (node.flowList.size == 0)
15.     delete this node and all of its children
16. else if (seq_list.size >= support)
17.     output the path from root to node as frequent pattern
END

(b)
```

Figure 4 (a) Root node processing algorithm. (b) Subtree processing algorithm.

Figure 4(b) shows the specific process of processing the ordinary node in the subtree. Lines 2-3 check the timestamp of the data stream. If the data stream is not within the current time window, the data stream within the node is deleted and continues to traverse to the next data stream. Lines 4-13 deal with new arrivals, the lines 8-11 update the timestamps of the data streams and data streams in the nodes. If the corresponding data flows already exist within the nodes, the corresponding timestamps are updated to ensure that the recurring sequence patterns are not deleted. If the number of data streams in the node is greater than the minimum support in advance, it means that the sequence pattern represented by the root node to the current node is frequent.

4. Adaptive Strategy for Parallel Processing

Due to the uncertainty of the stream data, the stream data may be accompanied by explosive growth or decrease over a certain period of time, which can affect the efficiency and resource use of the algorithm. Adding the adaptive strategy to the algorithm can reduce the influence of the flow data change, and is more suitable for the actual use of the flow data in the scene.

This paper summarizes the three factors of adaptive adjustment of the algorithm, namely PS width, stream data flow rate, thread pool task queue length.

Definition 2(PS width). PS width is equal to the number of child nodes under the PS root node, indicated by the letter B.

Definition 3(Flow rate). Let the number of data streams be N(S_t) and the number of data items of all incoming data is N(I_t), The stream data flow rate at time t is expressed as V_t= N(I_t)/ N(S_t)

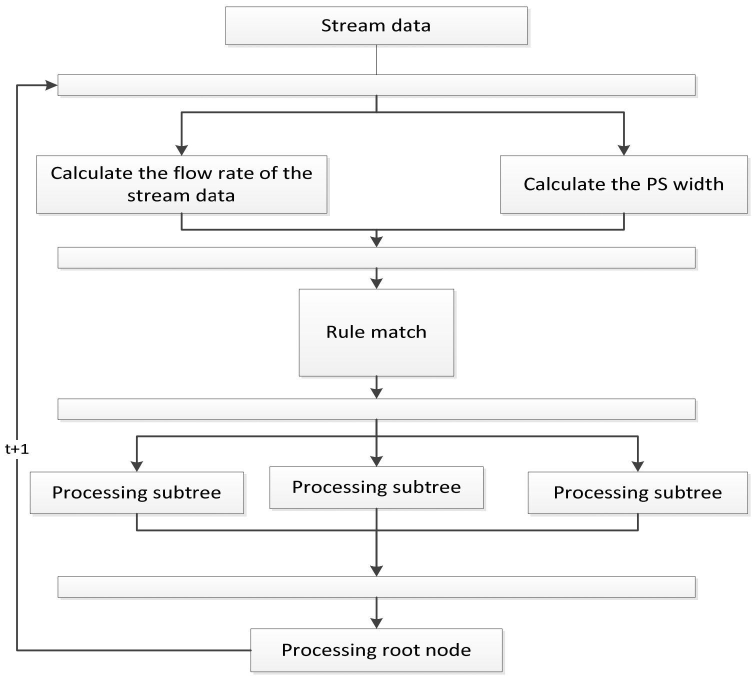

Figure 5 Adaptive strategy.

As shown in Figure 5, the adaptive rule in the strategy is based on the current time data flow rate V_t, PS width B_t and the task queue length *LQ* passed at the last time, and then adjusts the size of the thread pool. Then Parallel-Pisa started to deal with PS. The adaptation rules are given in Figure 6.

Adaptation rules:

1) If $V_t > V_m$: Increase the number of threads.
2) If $(V_t\text{-}V_{t-1})/(V_m\text{-}V_{t-1}) >= T_v$: Increase the number of threads.
3) If $LQ/B_t >= T_b$ && $((B_t >= B_{t-1})$ or $(V_t >= V_{t-1}))$: Increase the number of threads.
4) If $LQ/B_t < T_b$ &&$((V_t < V_{t-1})$ or $(B_t < B_{t-1}))$: Decrease the number of threads.
5) If $(V_{t-1}\text{-}V_t)/(V_m\text{-}V_t) >= T_v$ && $LQ/B_t < T_b$: Decrease the number of threads.

Figure 6 Adaptive adjustment rules.

V_m in the rule represents the maximum flow rate in history and *LQ* represents the length of the task queue in the thread pool. T_v and T_b are the two thresholds given by the user. T_v represents the magnitude of the flow rate, in the range of 0 to 1. If the change in flow rate exceeds the threshold value T_v, the flow data represents a significant increase or decrease in the current time. T_b represents the degree of blocking of threads in the thread pool, in the range of 0 to 1. If the value of (*LQ*/*B*) exceeds the threshold value T_b , the current task queue is regarded as a high blocking state and vice versa as a blocking state of the task queue normal.

It should be noted that the frequent thread switching will lead to the performance degradation of the program, so the adaptive adjustment strategy is not an unlimited increase in the thread, because the algorithm in the convective data processing is a compute-intensive operation, so the adaptive strategy the maximum number of threads will be set to (CPU * 2).

5. Experiment

This section examines Parallel-Pisa in terms of data throughput, efficiency, accuracy, and so on. The synthetic data sets are generated in a way similar to the IBM data generator in [7] designed for testing sequential pattern mining algorithms. As Pisa [4], Fast-Pisa [4] and similar to Parallel-Pisa are using sliding window model for frequent data mining model, and the sliding time window model is inherently capable of processing the stream data. So later use the above two algorithms and Parallel-Pisa for comparison. The experiments are executed on a computer with Windows 7, Intel (R) Core (TM) i7-4790 processor, RAM 8G. All the algorithms are coded in java.

5.1. Data Throughput

This section examines the throughput of the algorithm, comparing Pisa, Fast-Pisa, and Parallel-Pisa to handle the total amount of data for candidate elements per second. Table 1 gives the throughput of Pisa, Fast-Pisa and Parallel-Pisa per second under the same window length and minimum support. In terms of throughput Parallel-Pisa has obvious advantages over the other two algorithms.

Table 1 Throughput test.

Algorithm	Parallel-Pisa	Fast-Pisa	Pisa
candidate elements(million/sec)	85.1	23.7	7.6

5.2. Efficiency and Result Integrity

Figure 7(a) shows the execution time for three algorithms at different flow rates. In terms of efficiency, Fast-Pisa and Parallel-Pisa are far higher than Pisa, but Fast-Pisa is an improved algorithm of Pisa to improve the efficiency of the algorithm at the expense of the accuracy of mining results. We will experiment with the accuracy of the mining results in next.

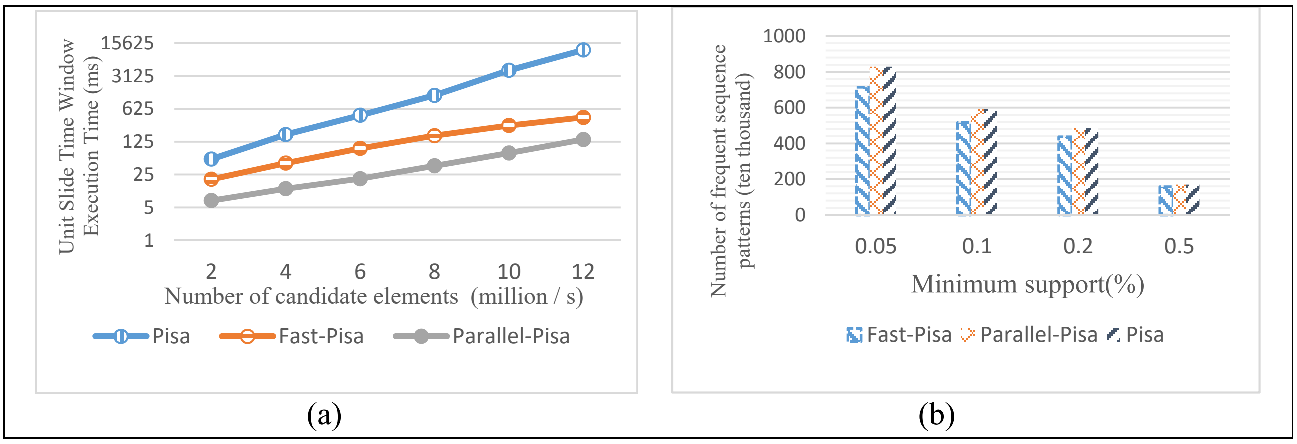

Figure 7 (a) Performance Testing. (b) Result Integrity.

Figure 7(b) shows the number of results obtained by three algorithms for the same data at different minimum support levels. With the decrease of the minimum support, the loss rate of Fast-Pisa mining results is gradually increasing, and the advantages of Parallel-Pisa and Pisa are becoming more and more obvious. Parallel-Pisa has better results integrity than Fast-Pisa, which can better assist the user in making decisions.

5.3. Adaptive Strategy Effect

Figure 8 shows the Parallel-Pisa adaptive strategy effect by receiving a different number of candidate elements per second.

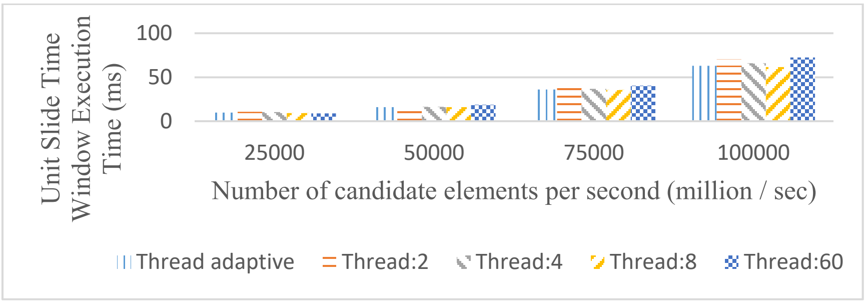

Figure 8 Adaptive testing.

It can be observed that Parallel-Pisa with adaptive strategy has always maintained a high level of efficiency, and its flexibility can be found when compared to an experimental group with a fixed number of threads of 60. Due to the relationship between the CPU and the inter-thread resource competition, the increase in the number of threads cannot solve the problem of the performance of the algorithm. Adaptive strategy allows Parallel-Pisa to deal with the change of stream data more stably. So that the system can use the resources more reasonably and avoid the unreasonable number of threads.

6. Conclusion

In this paper, a frequent pattern mining algorithm named Parallel-Pisa that can be applied to stream data is studied based on the large, infinite and unpredictable characteristics of streaming data. The existing algorithms are mainly based on single thread, so it is difficult to meet the fast arrival of streaming data. Parallel-Pisa uses the parallel processing technology to deal with the PS-tree, and at the same time through the adaptive strategy to increase the unpredictability of convective data control, so that it can be better applied to frequent data mining in the stream.

Stream data for the pursuit of algorithm efficiency is endless, the next step we will continue to optimize the algorithm to assess its application in the real data effect.

References

[1] S.Y. Yang, C.M. Chao, P.Z. Chen and C.H. Sun. (2011) Incremental mining of closed sequential patterns in multiple data streams, Journal of Networks 6(5), 728–735.

[2] C.W. Lin, T.P. Hong and W.H. Lu. (2009) An efficient FUSP-tree update algorithm for deleted data in customer sequences, in: Fourth International Conference on Innovative Computing, Information and Control, 1491–1449.

[3] T.P. Hong, H.Y. Chen, C.W. Lin and S.T. Li. (July 2008) Incrementally fast updated sequential patterns trees, in: Proceedings of the Seventh International Conference on Machine Learning and Cybernetics Kunming, 3991–3996.

[4] J.W. Huang, C.Y. Tseng, J.C. Ou and M.S. Chen. (2008) A general model for sequential pattern mining with a progressive database, IEEE Transactions on Knowledge and Data Engineering 20(9), 1153–1167.

[5] C.W. Lin, T.P. Hong and W.H. Lu. (2009) An efficient FUSP-tree update algorithm for deleted data in customer sequences, in: Fourth International Conference on Innovative Computing, Information and Control, 1491–1449.

[6] A. Mhatre, M. Verma and D. Toshniwal. (2009) Extracting sequential patterns from progressive databases: A weighted approach, in: IEEE International Conference on Signal Processing Systems, 788–792.

[7] R. Agrawal and R. Srikant. (1995) “Mining Sequential Patterns,” Proc. 11th Int’l Conf. Data Eng. (ICDE ’95), pp. 3-14, Feb.

[8] Bhawna Mallick, Deepak Garg. (2013) Incremental mining of sequential patterns: Progress and challenges, in :

Intelligent Data Analysis 17 , 507-530.
[9] Hui Chen,LihChyun Shu,Jiali Xia,Qingshan Deng. (2012) Mining frequent patterns in a varying-size sliding window of online transactional data streams[J]. Information Sciences .
[10] Mahmood Deypir,Mohammad Hadi Sadreddini. (2011) A dynamic layout of sliding window for frequent itemset mining over data streams[J]. The Journal of Systems & Software . (3)