**Incremental Temporal Frequent Pattern Mining Based on Spark Streaming**

<table>
<thead>
<tr>
<th><strong>Journal:</strong></th>
<th><em>Journal of Advanced Computational Intelligence and Intelligent Informatics</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manuscript ID:</strong></td>
<td>Jc-RP(R)-18-0318</td>
</tr>
<tr>
<td><strong>Manuscript Type:</strong></td>
<td>Regular Paper: Research Paper</td>
</tr>
<tr>
<td><strong>Date Submitted by the Author:</strong></td>
<td>23-Nov-2018</td>
</tr>
</tbody>
</table>
| **Complete List of Authors:** | YiJian, Zhao; Central South University, School of Information Science and Engineering  
Fang, Huang; Central South University, School of Information Science and Engineering  
ShaoYong, Wang; Central South University, School of Information Science and Engineering  
Chunxiu, Du; Central South University, School of Information Science and Engineering  
Zhang, Chengyuan; Central South University |
| **Keywords:** | parallel mining of streaming data, time accumulating frequency, temporal sub-frequent pattern, time frequent pattern tree, incremental mining |
| **Categories**<br><font color="blue">*Select at least 1 category from the list.</font>: | Distributed intelligent systems, Scientific applications, Intelligent systems |
As we are mining streaming data using frequent pattern, the accumulation of time is a huge factor in influencing the relationship between data items. In this case, how to keep track of the historical information of streaming data efficiently and design a temporal frequent measurement which considers time accumulation are two critical problems of frequent pattern for mining streaming data. As a result, we bring out a scheme of incremental temporal frequent pattern mining based on spark streaming framework. More specifically, as for the time property of streaming data, we design a calculating approach of temporal frequency which could decay and increase along with time. Simultaneously, as for the property of information accumulation, we put forward the ITFP (Incremental Temporal Frequent Pattern mining) algorithm. In order to minimize the spatial cost of recording historical information, this algorithm draws into TFP-tree (Time Frequent Pattern tree) to save historical frequent patterns, and reconstructs TFP-tree to achieve incremental mining. Additionally, a temporal sub-frequent pattern is proposed as backtracking window of historical information to reduce the deviation of frequent pattern. Finally, we test and verify above mentioned approaches through using public paper data set. And the experimental results show that the ITFP mining based on spark streaming has better performance of accuracy and validity, and even better extendibility under the distributed circumstance.

Keywords: parallel mining of streaming data, time accumulating frequency, temporal sub-frequent pattern, time frequent pattern tree, incremental mining

1. INTRODUCTION

As one of the important features of big data [1], streaming data, which is generated with the development of Internet and computer information technology, has properties of real-time and continuity. Since the generation of streaming data has the characteristic of infinite source along with time, traditional data mining methods could not be applied into the streaming data environment directly. Therefore, the mining of streaming data is an essential project in the field of big data knowledge discovery area.

In the domain of data mining, the researches and applications of temporal frequent pattern (TFP) mining have been processing for many years. It is to regard the time property of data as the factor which influences the relationship between data items, which is conducive to revealing the potential tendency of the development of things more effectively [2]. However, in the inside of streaming data, the data set accumulates over time, and the data mining system was unable to record the historical data completely. It affects the validity of the time-based frequent pattern calculation, making the TFP mining perform poorly. In addition, it is clear that the traditional ways of scanning data sets frequently could not meet the demands of streaming data mining. The classical FP-Growth algorithm does the frequent mining by establishing FP-tree when processing frequent pattern set mining. It does not require to scan the original data set frequently, and it is a common algorithm for parallel association mining. However, while mining streaming data, FP-tree cannot save complete historical records as subsequent data continues to generate and flow. This is the main obstacle to apply FP-Growth algorithm into streaming data mining. Accordingly, this paper proposes a FP-Growth based frequent pattern mining scheme of streaming data under Spark Streaming computing environment. The scheme first uses the FP-Growth algorithm to realize parallel counting of K-items and dynamically calculates the current TFP. Then, the ITFP algorithm with incremental updating and frequent pattern classification strategy is proposed in this paper, and the current TFP information is used as the time-sequence increment for cumulative updating, in order to achieve mining TFP of stream data.

In the parallelization framework, Apache Spark is a fast and universal computing engine designed for large-scale data processing, with good data parallelism and scalability [3]. Spark Streaming is the mainstream computing framework for streaming data processing [4]. We use Spark Streaming to realize parallel data mining. At the same time, designing an efficient FP-Growth parallelization scheme which is suitable for Spark Streaming with load balance will greatly enhance application value of FP-Growth. In addition, in processing streaming data, the accumulation and preservation of the intermediate results are also crucial to make the final result coherent and continuous. Therefore, a mining approach based on Spark streaming data and FP-Growth temporal pattern is proposed in this paper.
This approach combines the divide-and-conquer and balanced grouping strategy of FP-Growth to design a parallelization scheme which is suitable for Spark Streaming computing framework. The incremental temporal frequent pattern update algorithm (ITFP for short) is also deployed inside. Simultaneously, a temporal tree structure Time Frequent Pattern tree (TFP-tree for short) is used to dynamically record historical frequent pattern, and we use temporary sub-frequent pattern as the backtracking window to improve the effectiveness of temporal frequent pattern mining of stream data.

The paper is organized as follows. The second part is related work of research; the third part demonstrates the temporal frequent pattern mining based on Spark Streaming; the fourth part elaborates principles and parallelization strategy with algorithm designs of ITFP mining for streaming data. And the fifth part is the experimental comparisons and analysis; the sixth part summarizes this paper and future work.

2. RELATED WORK

Streaming data mining is a hot research topic in the field of big data analysis. The current algorithms of frequent-item set mining for streaming data is usually based on sampling, counting, and hashing techniques to achieve approximate mining of infinite streaming data within limited computing resource [5]. As for the sampling-based method, in 2002, C. Estan[6] with his team proposed the sample-and-hold algorithm, which sampled and counted the data depending on a certain rate, to realize the approximate mining of frequent-items sets. Subsequently, G.S. Manku et al. [7] proposed the Sticky Sample algorithm, which used dynamical sampling rate to sample and estimate the data in data stream, and stored the results in the global abstract data structure, to improve the accuracy of frequent itemset mining results. In the counting-based method, the Lossy Counting algorithm was proposed by L.Yang et al.[8] in 2004, which ensured the accuracy of mining frequent-items sets by maintaining the error threshold and counter of each item. And it indeed improved the mining accuracy of frequent-items sets greatly. In the following 2008, Tantono F.I et al. proposed the classical algorithm FP-Stream[9], which is based on the FP-Growth algorithm and combines the counting and global FP-Tree to save mining results of frequent-items sets. At the same time, the time window was also embedded into the FP-Tree structure, which better preserved the records of frequent-items sets in the data flow. And the accuracy of association rules mining about streaming data was also further improved. As for the hash-based method, Charikar et al[10]. designed a CountSketch data structure and mapped the data items to the counter through a hash function, so that the top k frequent items in the data flow can be quickly mined. This approach also improved the speed of mining. However, in the above three stream data association rules mining algorithms, the influence of data time attributes of data items on the frequent item sets is not considered. Because the data continuously generates and flows over time, the constant accumulation of the data must affect the frequency of the data items. This kind of accumulating attribute along with time is an inevitable factor in affecting the validity of the association rules mining results, and it also has important application value [11,12]. Therefore, temporal association rules mining based on streaming data is currently an open research issue.

Due to the huge amount of streaming data and the continuous increment over time, it is clear that traditional stand-alone processing modes are unable to meet people's demands. Thence, some scholars began to improve traditional mining algorithms based on association rules to meet the needs of distributed computing. In 2007, Wang Y[13] proposed PFP (Parallel Frequent Pattern Growth Algorithm) based on FP-growth. This algorithm used the distributed framework MapReduce to realize the parallel computation of FP-growth, which greatly improved mining speed of association rules. In 2011, Zhou L et al.[14] proposed the Balanced Parallel FP algorithm by improving PFP algorithm. The algorithm added load balance function on the basis of PFP, which further improved the running speed. With the continuous advent of newborn data in the flow, the existing association rules need to be stored and updated effectively. The literature [15] proposes a Fast Updated Algorithm FUP based on the Apriori and gave the baseline of classifying frequent items. Since this algorithm needs to scan the data set several times, its efficiency needs to be improved further. In 2007, the FUPF (Fast Updated Frequent Pattern) algorithm was proposed by Hong TP[16] and it was considered as one of the most effective algorithms for updating FP-tree. In this way, the efficiency of the operation has been greatly improved by updating FP-tree in FP-Growth to mine frequent-items set. As the current mainstream processing framework of streaming data, Spark Streaming architecture is regarded to be the basic environment of designing parallelizing FP-Growth algorithm. Simultaneously, combined with the rapid update of frequent-item sets, the efficiency of mining association rules will be greatly improved, which enhances the application value of the FP-Growth to a large extent.

3. TEMPORAL FREQUENT PATTERN MINING SCHEME BASED ON SPARK STREAMING

In the temporal frequent pattern mining system, which is based on the Spark Streaming architecture, the system receives the data flows arranged in time sequence according to real time and batches them. After that, it submits them to the Spark Streaming processing engine for distributed parallel mining, and then each computing node is aggregated to generate global mining results. As the data flows continue to arrive, the historical TFP is
also updated and stored at the same time. The scheme is shown in Fig. 1. Its main tasks consist of three parts:
1) distributed storage of time slice streaming data; 2) parallel mining of K-item sets count; 3) storage and update of temporal frequent patterns.

1) Distributed storage of time slice streaming data

When streaming data arrives continuously in units of time slices, the distributed file system (HDFS for short) is used to incrementally store all data on time slices in the file system through batches. The distributed file system is monitored by the file monitoring functions of Spark Streaming. When the data in the file system arrives, all of data is immediately taken out and processed by the parallel FP-Growth algorithm on the basis of Spark Streaming.

2) Parallel mining of K-item count

When mining the local K-item set parallel to obtain the K-item set count, first of all, each one-item set is counted parallel and then parallel counting results are merged. Then, the candidate frequent one-item sets are filtered out and sorted in descending order. According to the size of each candidate frequent one-item set and the number of parallel mining computing nodes, the candidate one-item sets are balanced grouping, and the transaction records corresponding to candidate one-items in the group are delivered to the computing nodes of the group. Later the local K-item sets are grouped and calculated through parallel FP-Growth algorithm. Finally, the local K-item set calculations are summarized.

3) Storage and update of temporal frequent pattern

For the formed global K-term sets, the dynamic calculating formula of frequent measurement based on time accumulation is used. As for the temporal frequent pattern and the temporal sub-frequent pattern of the current moment, both of them are mined in combination with the historical TFP-tree and historical temporal sub-frequent pattern. Then the temporal sub-frequent pattern is used to update its historical records. The TFP-tree is established according to the TFP set. And the temporal sub-frequent patterns are stored in the Radis, which are regarded as history records that may become a TFP set subsequently. When subsequent frequent pattern set continues to arrive, the ITFP combined with historical records of temporal sub-sequence pattern to incrementally update the current TFP. And above processes also store temporal information accumulated by the streaming data according to time sequence.

4. TEMPORAL FREQUENT PATTERN MINING OF STREAMING DATA

The TFP mining of streaming data based on Spark Streaming mainly includes two parts. The first part is to use the FP-Growth algorithm based on Spark Streaming to realize parallel counting of K-item sets, and then get the TFP sets and temporal sub-frequent pattern sets according to the frequency of dynamic time accumulation; the second part is to use the ITFP algorithm to quickly update the TFP with reconstructing TFP-tree. The purpose is to achieve dynamically maintaining the historical information.

4.1. Dynamic frequency calculation based on time accumulation

For streaming data, the number of occurrences of data items accumulates over time. At the same time, it is obvious that, because of passing time, the strength of the relationship between data items gradually decreases. So, the frequency calculation should consider not only the accumulation of data items over time, but also degree of their attenuation over time.

4.1.1. Time accumulation

If one data item appears frequently in the flow and its latest appearance is closer to current point of time, it means that its frequency at current time is higher, and the possibility of becoming a frequent pattern is greater, which naturally forms the basic idea of dynamic frequency calculation based on time accumulation. The detailed calculation processes are as follows.

Let \( T^i \) represent the number of counting times during the period divided by time granularity from the beginning to the current time \( t \). \( M_i = \{ m_i | i = 1, 2, \ldots \} \) \( m_i \) is the total number of times when data pattern \( i \) actually appears at each counting period from beginning to the time \( t \). For example, regarding the year as time granularity, the streaming data started in 2010, and current time is 2018, which means \( T^{2018} = 8 \). During this period, one data item appears in the 2011 and 2015 respectively. So, at the
current time, \( m_i^{t+2} = 2 \). The cumulative statistical formulas for \( T^t \) and \( m_i^t \) are shown respectively in equations (1) and (2):

\[
T^t = T^t + 1
\]

\[
m_i^t = \begin{cases} 
  m_i^{t-1} + 1 & \text{if } I_i \text{ did not appear at } t \\
  m_i^{t-1} & \text{if } I_i \text{ appear at } t
\end{cases}
\]

4.1.2. Time attenuation

Let \( L_t = \{I_i | i = 1, 2, \ldots \} \) represent the set of all the pattern sets at time \( t \). \( C_t = \{(I_i, w_i') | i = 1, 2, \ldots \} \) is the set of each pattern \( I_i \) in pattern set \( L_t \) corresponding to each weight set at time \( t \). With the inflow of data, the number of occurrences of the pattern \( I_i \) in current batch of data stream is combined with the time accumulation to update the weight \( w_i \) of the \( I_i \). If the \( I_i \) does not appear, its weight will decline. And the calculation formula (3) is as follows.

\[
\begin{align*}
    w_i^0 &= \text{count}(I_i) \\
    w_i^t &= w_i^{t-1} + \text{count}(I_i) \cdot (1 - \delta)^{T_e - m_i^t} \quad I_i \notin L_t \\
    w_i^t &= w_i^{t-1} \cdot (1 - \delta)^{t - \text{last time}} \quad I_i \in L_t
\end{align*}
\]

\( \text{count}(I_i) \) is the number of times when pattern \( I_i \) appears at time \( t \), \( \delta \) is the decay factor set by us, \( 0 < \delta < 1 \). And last_time is the time record when pattern \( I_i \) appeared last time. Formula (3) shows that if pattern \( I_i \) occurs frequently and the number of occurring times is copious, and the value of its temporal frequency will be higher; if pattern \( I_i \) is infrequent and the latest occurrence is far from current time, its temporal frequency value will decay over time. If pattern \( I_i \) does not appear for a long time and its value decays below a threshold, it will be removed from the frequent pattern set and becomes infrequent pattern.

4.2. Parallel mining of temporal frequent patterns based on Spark Streaming

The parallel mining process of TFP is divided into three parts: count and sort candidate one-item set, and load balancing grouping and temporal frequent pattern mining. In the whole process, in order to provide historical information for incremental mining of subsequent temporal frequent patterns, a special TFP-tree is used to record the accumulated information of historical frequent patterns. When acquiring the current TFP, temporal sub-frequent patterns are saved simultaneously, which is regarded as a historical infrequent item to provide a retroactive window for the next frequent pattern mining. As for the parallel mining scheme based on Spark Streaming, firstly, a Map-Reduce parallel processing is used to obtain the count of one-item sets, and candidate frequent one-item sets are obtained from it. Then the candidate one-item sets are merged and sorted, and load balance grouped. Each group is assigned with a parallel computing node by using Map-Reduce, and to carry out the mining of local K-item set. Finally, the local K-item sets are aggregated into a global K-item set, and then the temporal cumulative dynamic frequency of the current frequent items are calculated on the basis of the historical temporal frequent pattern and its sub-frequent pattern. In this way, the current temporal frequent pattern and its sub-frequent pattern are obtained.

4.2.1. Candidate frequent pattern and sub-frequent pattern

In the streaming data, since the historical data set is not stored, when the frequent pattern mining is processed, the previous history records cannot be traced back, which reduces the accuracy of frequent pattern mining. According to the classification idea of item sets [15], the sub-frequent pattern is employed to retain the historical information of valid infrequent pattern, which is a great possibility that they will becomes frequent patterns with the accumulation of subsequent data. So, we save them as historical sub-frequent patterns regarded as an effective part of subsequent calculations to further facilitate backtracking, thereby improving the accuracy of frequent pattern mining.

Definition (1): The item \( I \) is regarded as candidate frequent pattern, only when \( I \cdot \text{count} \geq \sigma |DB| \) and \( 0 < \sigma < \delta \). \( DB = \{DB_1, DB_2, \ldots, DB_i, \ldots \} \) represents data flow. The minimum support is \( \delta \). \( |DB| \) represents the total number of events in \( DB_i \), and the candidate minimum support is \( \sigma \). \( I \cdot \text{count} \) is the count of \( I \) in the data \( DB_i \).

Definition (2): The item \( I \) is regarded as sub-frequent pattern, only when \( \sigma \cdot |DB| < I \cdot \text{count} < \delta \cdot |DB| \) and \( 0 < \sigma < \delta \).

In the process of parallel frequent pattern set mining, the sub-frequent pattern set is stored in Radis as the retroactive window for subsequent updates. At this point, all data items in the event record can be classified into a frequent pattern set, a sub-frequent pattern set, and an infrequent pattern set. And the frequent pattern set and the sub-frequent pattern set constitute the candidate frequent pattern set.

4.2.2. Load balance grouping based on frequent pattern tree

When there are multiple computing nodes, if there is a large difference in computation cost between the computing nodes, and the task distribution is uneven, it will result in a higher cost of the overall computing time of the system. Therefore, before parallel FP-Growth mining, it is necessary to realize load balance grouping for data by integrating computing node resources and the amount of data calculation.

(1) Parallel computing of candidate one-item set

When transaction records continue to arrive, their item counts are first counted parallel through Map-Reduce program to obtain candidate one-item sets. The main steps are as follows:

1st step: Get the data item from transaction records.
The row number of transaction record is used as the key of the dictionary(key-value pairs). The transaction record of the row is used as the value of the dictionary, which is also the input data of Mapper. The output is a key-value pair with the transaction record as the key and 1 as the value.

2nd step: Get the count of each one-item. The MapReduce program together the key-value pairs of same key, and form the dictionary of key-value pairs which is the output whose key is the item and value is the counts.

3rd step: Merge and sort. The items that are less than the candidate minimum support \( \sigma \) are removed to obtain a candidate one-item set. And then the one-item set is sorted in descending order. The sorted candidate frequent one-item sets are recorded as \( F\text{-list} \) and then grouped.

(2) Load balance grouping

The traditional approach of load balance is to group directly, in other words, dividing the \( F\text{-list} \) into \( n \) groups according to the number of computing nodes \( n \). However, in the frequent pattern tree, the nodes in the candidate one-item sets are used as nodes in the tree. The nodes passing through a path from the root to the leaf nodes are candidates, and the weight of the nodes on the path is the number of occurrences of nodes in one-item set, which is gradually decreasing from the top to the bottom. The closer to the bottom, the smaller the number of occurrences. The higher the corresponding pattern tree, the more iterations happen when mining frequent items and the computing load is also bigger. Therefore, the leaf nodes and the non-leaf nodes of the frequent pattern tree are reasonably matched and grouped, so that the calculation amount of each computing node corresponding to each group is almost same. Consequently, the load balance of system calculation is achieved.

An example of the process of grouping original transaction records is presented in Fig. 2. Assuming the minimum support 0.4. There are two compute nodes in the system. First, the candidate one-item set \( F\text{-list} \) is obtained by removing item \( m \) smaller than the minimum support by parallel counting. Then the items in each transaction record are sorted according to the \( F\text{-list} \) order, which are inserted into the frequent pattern tree. After all the transaction records are completely processed, balance collocation strategy is adapted to divided the data items into two groups two groups \( G\text{-list} \) and \( G\text{-list} \) by the level of data items in the frequent tree and the computing resources of the system. The candidate one-item set can guarantee the balance calculation to a certain extent.

4.2.3. The parallel mining of temporal frequent pattern

According to the candidate one-item set after grouping, the corresponding original transaction records are grouped. This process of the temporal frequent pattern is mined is mainly consisted of three steps: 1) transaction records are grouped; 2) local K-item sets are parallel counted in each computing node; 3) calculate temporal frequent pattern. Local parallel computing using MapReduce program is exerted in the first and second steps, and then the results are merged into a global K-item set count. The specific mining processes are as follows:

1st step: According to the groups of the candidate one-item sets, the data items in the event record are sorted in descending order by \( F\text{-list} \). The candidate item set \( (G\text{-list})_{\text{gid}} \) and the event record set \( T\text{-list} \) are read by the Mapper, and according to the item \( a_i \) in the \( (G\text{-list})_{\text{gid}} \), the corresponding group number \( \text{gid} \) is assigned to each record \( T_i \) in the \( T\text{-list} \), denoted as \(<\text{key}=\text{gid},\text{value}=a_0,a_1,...a_q>\). Then we combine all the transaction records having same group number to obtain the event record packets \( (T\text{-list})_{\text{gid}} \).

2nd step: The Map-Reduce program uses the grouped transaction records \( (T\text{-list})_{\text{gid}} \) and the corresponding \( (G\text{-list})_{\text{gid}} \) as the input data of each compute node of Reduce, and then calls the FP-Growth algorithm in the Reduce program to do K-item set mining in order to get the counts of the local K-item sets.

3rd step: All the local k-item set counts are merged into a global K-item set count, and accumulate according to the historical information of TFP-tree records and the temporal sub-frequent pattern records. Calculate the dynamic time accumulating frequency of top K-item set, which leads to get the TFP pattern of current moment. Then update the historical temporal sub-frequent pattern stored in the Redis and update TFP-tree according to the new TFP sets in order to store historical TFP information.

![Fig. 2. The illustration of counts and load balance](image-url)
4.3. Incremental temporal frequent pattern (ITFP) mining algorithm

In the traditional FUFP updating algorithm, the historical information can be maintained by the FP-tree. When new data arrives, the tree updates the node information is updated or adds a new node without changing the original tree structure. Its requirements are that the following incremental data of event data is small, and FP-tree can store all candidate item sets. Due to the large amount of subsequent inflow of the streaming data, it cannot store the complete history records. Therefore, in the ITFP algorithm, a special TFP-tree is used to record the accumulation of the temporal frequency over time. But if the amount of follower data is huge, the order of the items in the historical TFP will change, and it cannot be updated directly on the original TFP-tree. Thus, this paper proposes to a reconstruction update method for TFP-tree, which can achieve by using minimum space cost to store the historical information of TFP. At the same time, the sub-TFP (Temporal sub-Frequent Pattern) is used as an important historical retroactive window, which is likely to become the frequent pattern in the future. The K-item sets of the current time are counted and it is accumulated with above two parts of historical information to achieve incremental TFP mining, which could reduce the error of TFP. This algorithm is mainly composed of two parts: 1) the incremental mining of the TFP and the sub-TFP; 2) the reconstruction of the TFP-tree.

4.3.1. Algorithm principle

When the data flow at time t arrives, the K-item sets of the current data are counted. And the dynamic time accumulating frequency is calculated together with the TFP at time t-1 and the historical sub-TFP. This process is to mine out the TFP and the sub-TFP of the current moment. The TFP-tree is reconstructed and updated using TFP of current moment. The specific process is illustrated in Fig. 3.

The main content includes incremental mining of TFP and sub-TFP. And the reconstruction update of TFP-tree. The specific description is as follows:

1) Incremental mining of TFP and sub-TFP

In the Fig. 3, for the current count of K-item set (K-list), if the item appears in the historical TFP (FP-list)_t, at time t-1, it will be added to the current TFP (FP-list)_t after being calculated its temporal frequency. For example, the frequency of AB appears twice in (K-list). According to the second formula in Equation (3), the temporal frequency accumulation of current AB can be obtained as 5. And then it is added to (FP-list). However, at time t, since ADE does not appear in (K-list), according to the third formula in Equation (3), we attenuate its frequency value to 4 and add it to (FP-list).

For the newly born in (K-list), such as {BF:1, BE:5, AE:1, AC:1}, whose BF appears once in the historical sub-TFP sets (SFP-list)_t. The frequency is greater than the threshold 2 by accumulating their temporal frequency, it is removed from (SFP-list)_t and added to (FP-list).

If AE does not appear in historical sub-TFP set (SFP-list)_t, its temporal frequency is less than the frequent threshold and greater than the sub-frequent threshold, it is added to the current sub-TFP set (SFP-list). In this way, the update of the current TFP (FP-list)_t and sub-TFP (SFP-list) is finished.

2) Reconstruction of the TFP-tree

In the Fig. 3, after obtaining the updated TFP set (FP-list)_t, firstly we count one-item sets in the (FP-list)_t, and the F-list is obtained by sorting the counts.

Then the frequent patterns in (FP-list)_t are sorted in the order of the items in the F-list, and the sorted TFP (FP-list)_t is obtained. At this point, it can be found that the order of some frequent items has changed. For example, the order of the items in ADE has been changed compared with AED at time t-1. Therefore, the frequent pattern tree at time t-1 cannot be directly modified. And in order to effectively take advantage of characteristics from prefix tree to reduce the size of the TFP-tree, we must reconstruct TFP-tree as follows. First initialize an empty (TFP-tree), then traverse all the TFP in (FP-list)_t, and insert each TFP into the new (TFP-tree), path in order. The weight of the node represents the temporal frequency of the path from the root node to the current node. The reconstruction of new (TFP-tree), will not complete until all frequent patterns in the (FP-list)_t are traversed.

4.3.2. Algorithm design

1) Incremental mining

Due to the fast read and write performance of Redis, we use Redis storage tech to stores the temporal frequent item sets for historical data flow, which provides historical information retrospect for the next batch of data. After counting the K-item set, the algorithm obtains the temporal frequency.
Algorithm : Updating TFP and historical sub-TFP

Input:
- \( DB_t \) : The data set at time \( t \)
- \( (SFP\text{-}list)_t \)
- \( (TFP\text{-}tree)_t \)

Output:
- \( \theta_1 \) : The threshold of TFP
- \( \theta_2 \) : The threshold of sub-TFP
- \( (K\text{-}list)_t = get\text{Item}\text{Account}(DB_t) \)

For each item \( m \) in \( (K\text{-}list)_t \) // Traverse each item in \( (K\text{-}list)_t \)

If \( (ימיContainsItem(TFP\text{-}tree)_t,m) \) :
- Get historical temporal frequency of \( m \) from TFP-tree
  \( \text{oldValue} = get\text{ValueFromTree}(TFP\text{-}tree)_t, m) \)
Else:
- Get historical temporal frequency
  \( \text{oldValue} = get\text{ValueFromList}(SFP\text{-}list)_t, m) \)

End if

End if

End for

Return \( (SFP\text{-}list)_t, (FP\text{-}list)_t \).

(2) The reconstruction of TFP-tree

When the data \( DB_t \) at time \( t-1 \) is loaded into the system, a TFP-tree \( (TFP\text{-}tree)_t \) will be established after the parallel TFP mining. When the subsequent data arrives, the incremental mining of current TFP is finished. Then the historical \( (TFP\text{-}tree)_t \) is reconstructed to store the current TFP information in the memory. The specific algorithm is described as follows:

Algorithm : TFP-tree reconstruction

Input:
- \( (FP\text{-}list)_t \)

Output:
- \( (TFP\text{-}tree)_t \)

// Initialize an empty TFP-tree with a root node
\( (TFP\text{-}tree)_t\text{.init}() \)

\( (F\text{-}list)_t = getFlist(FP\text{-}list)_t \) // Get the count order of one-item set

// Get ordered temporal frequent patterns

5. EXPERIMENTAL ANALYSIS

In this paper, the Aminer-Paper public data set[15] is used and its website is (www.aminer.cn/billboard/aminernetwork). Each data record from it contains the author, paper title, publishing time, the name of the magazine and etc. We extracted about 1.2 million data records containing authors’ information and publish time from 2005 to 2014. These records are divided by year. The annual data amount is shown in the Table 1. This data set is used to test the effectiveness of Spark Streaming-based ITFP mining from three aspects: 1) use the classical FP-Growth algorithm as a basic standard to verify the accuracy of ITFP algorithm; 2) do the mining of ITFP for temporal frequent or not, and separately to verify the validity of the TFP; 3) verify the extendibility of the ITFP based on Spark Streaming under a distributed environment.

<table>
<thead>
<tr>
<th>Year</th>
<th>Data amount (record)</th>
<th>Year</th>
<th>Data amount (record)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>98396</td>
<td>2010</td>
<td>139235</td>
</tr>
<tr>
<td>2006</td>
<td>121018</td>
<td>2011</td>
<td>131463</td>
</tr>
<tr>
<td>2007</td>
<td>116104</td>
<td>2012</td>
<td>123842</td>
</tr>
<tr>
<td>2008</td>
<td>128413</td>
<td>2013</td>
<td>92431</td>
</tr>
<tr>
<td>2009</td>
<td>169283</td>
<td>2014</td>
<td>21479</td>
</tr>
</tbody>
</table>

5.1. The accuracy and performance of ITFP

In the experiment, the result of the non-incremental mining algorithm FP-Growth is used as basic standard to compare with the ITFP algorithm’s result. We select the 2010 data from the 2010-year data set as the initial data, and then select 10,000 data, 30,000 data, 60,000 data, and 100,000 data in the 2011 data as incremental data. The initial data and one of the incremental data comprise 4 data sets respectively. The experimental environment is CPU i7-6700, 4G RAM, Ubuntu 14.04, Scala 2.11 and Spark 2.2. The frequency threshold of ITFP \( \theta_1 \) and the minimum support \( \delta^\# \) of FP-Growth are both set as 4. We use these two algorithms to do the mining of four different data sets respectively, and the results are shown in Table 2.
It can be seen from the four data sets in Table 2 that the overlapping results of frequent pattern between ITFP and FP-Growth is up to 94.0% on average. It can be concluded that the incremental mining of streaming data through ITFP can basically maintain the mining quality. This is because the ITFP algorithm adopts the sub-frequent pattern set to maintain important historical information in order to ensure the accuracy. By properly setting the threshold \( \theta_2 \) of sub-frequent pattern in ITFP (set 2 in the experiment), the ITFP can guarantee the quality of the results for the frequent pattern mining of streaming data.

### 5.2. Effectiveness of ITFP algorithm with temporal frequency

In order to verify the effectiveness of the ITFP, we compare the results between ITFP with temporal frequency and the other ITFP which is not with temporal frequency. The experiment environment is configured as follows: a host with CPU i7-6700 and RAM 16GB. The Spark cluster is built on it, including one master management node and two compute nodes Slave. The master node is configured with CPU i7-6700, 4G RAM, Ubuntu14.04 Scala2.11 and Spark2.2. The memory for Slave node is 2G RAM, and its other configurations are the same as the Master node. Both of these two approaches are set the same threshold \( \theta_1 \) and threshold \( \theta_2 \) ( \( \theta_1=3 \), \( \theta_2=2 \)). The data from 2005 to 2013 is extracted and used for frequent pattern mining through. The prediction sets of two frequent patterns after 9 years (2013) are obtained. The temporal patterns whose confidence interval are top-10, top-30, top-50, top-80, and top-100 are respectively composed of five sets of frequent patterns. And then we observe that whether the frequent pattern items of each group in the two prediction sets appear in the actual data set of 2014, so as to compare the accuracy of the prediction results of the two algorithms. The results are shown in Fig. 4.

<table>
<thead>
<tr>
<th>Data set</th>
<th>FP-Growth</th>
<th>ITFP</th>
<th>Same frequent pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO.1</td>
<td>2049</td>
<td>1899</td>
<td>1887</td>
</tr>
<tr>
<td>NO.2</td>
<td>2227</td>
<td>2117</td>
<td>2102</td>
</tr>
<tr>
<td>NO.3</td>
<td>2510</td>
<td>2411</td>
<td>2377</td>
</tr>
<tr>
<td>NO.4</td>
<td>4299</td>
<td>4106</td>
<td>4093</td>
</tr>
</tbody>
</table>

### 5.3. The extendibility of ITFP based on Spark Streaming

The extendibility experiment of Spark Streaming based ITFP algorithm is exerted on Spark cluster built with Ubuntu16.04, CPU i7-6800k (6-core 12-thread), RAM64GB host, and a total of 11 virtual hosts. One of them is Master management node, the environment is 4G RAM, Ubuntu16.04, Scala2.12, Spark2.3. And the remaining 10 nodes are Slaver, and its memory is 2G RAM. Its other configurations are the same as the Master’s. The number of compute nodes is gradually expanded from 2 to 10. The ITFP algorithm's threshold \( \theta_2 \) is set to 2, and the threshold \( \theta_1 \) is set to 3, 4, and 5 respectively. As for different thresholds and different number of compute nodes, the running time to verify the extendibility of system. We extract data from 2005 to 2014 (10 years), and the experiment results are shown in the Fig. 5.

As you can see in Fig. 5, in the case where the temporal frequency thresholds are same, the running time is
getting shorter and shorter along with increasing number of computing nodes, and as the node number increases, the slope of curve starts to decrease in the end. This is because the increasing exchange information among computing nodes causes a higher cost of time. At the same time, when the node number is same but the temporal frequency thresholds are different, the running time decreases along with the increment of thresholds. This is because the larger we set thresholds, the fewer frequent items we get. In parallel computing, the amount of distributed data is decreasing, resulting in a reduction of running time.

6. CONCLUSION

In this paper, we propose the incremental temporal frequent pattern mining algorithm (ITFP). At the same time, we propose a computing model of mining temporal frequency for streaming data based on Spark Streaming. This approach takes time sequence data flow as input and makes full use of the time sequence characteristics to measure the degree of close relationship between different objects in the data set over time. So, the frequent pattern mining is effective. Simultaneously, the TFP-tree is used in the ITFP algorithm to solve the problem of accumulation and update for historical information based on Spark Streaming. In this paper, the accuracy of the ITFP is verified on the standard basis of classical FP-Growth algorithm. And also, the validity of considering temporal frequency is verified through the prediction experiment. In addition, under the Spark Streaming environment, the two parallel computing nodes are gradually expanded to 10 nodes, which verify the extendibility of Spark Streaming-based ITFP mining in distributed environment. With sharp increase of data volume, the computing nodes must also increase correspondingly. The communication cost among computing nodes will affect the computing speed.

How to design a more reasonable approach through load balance data distribution to reduce the communication cost among nodes is our next step.

Acknowledgements

This work was supported by Project 2016JC2011, 2018J13691 of Science and Technology Plan of Hunan Province and Project 61073105, 61702560 of National Natural Science Foundation of China.

REFERENCES: