A Mining Frequent Itemsets Algorithm in Stream Data Based on Sliding Time Decay Window

Xin Lu[1], Shaonan Jin[1]*, Xun Wang[2], Jiao Yuan[2], Kun Fu[2], Ke Yang[2]
1. School of software, University of Electronic Science and Technology of China
2. China Railway Eryuan Engineering Group Co. Ltd
Chengdu, China
Email: luxinmail@uestc.edu.cn; 591450436@qq.com; 49481181@qq.com; 280771670@qq.com; 364202818@qq.com; jakke.yang@foxmail.com

ABSTRACT
In order to reduce the time and memory consumption of frequent itemsets mining in stream data, and weaken the impact of historical transactions on data patterns, this paper proposes a frequent itemsets mining algorithm SWFIUT-stream based on sliding decay time window. In this algorithm, the time attenuation factor is introduced to assign different weights to each window unit to weaken their influence on data mode. In order to realize the fast stream data mining processing, when mining the frequent itemsets, the two-dimensional table is used to scan and decompose the itemsets synchronously to mine all the frequent itemsets in the window, and the distributed parallel computing processing is carried out based on storm framework. Experimental data show that the algorithm consumes less time and consumes less memory space than conventional algorithms when mining frequent itemsets in stream data.

CCS Concepts
• Information systems→Information systems applications→ Data mining →Data stream mining • Computing methodologies→Distributed computing methodologies→ Distributed algorithms.

Keywords
Sliding window; stream data; frequent itemset; storm framework

1. INTRODUCTION
Stream data is a group of data sequences that arrive in time sequence. It has the characteristics of fast, continuous and infinite. It is often used in sensor network data acquisition, communication data processing, stock trading data analysis, e-commerce sales analysis and other fields [1]. Mining frequent itemsets in stream data is one of the basic problems in stream data mining. Different from the frequent patterns of traditional data sets, the hidden knowledge in stream data will change with time, that is, concept drift [2]. In the analysis of stream data mining, historical transactions will affect the results of data mining and lead to the deviation of mining results. Therefore, it is necessary to eliminate the obsolete data when mining and analyzing frequent itemsets of stream data. Generally, the window model is used to process the data. Only frequent itemsets are mined for the stream data in the window. In practical application, according to the different period range of stream data, the processing model of stream data window can be divided into the following three categories: the processing model based on landmark window, the processing model based on time decay window, and the processing model based on sliding window.

In the landmark window model [3], we focus on the data set from a start time point t to the current time point T. With the continuous arrival of data stream, the landmark model window needs to occupy a lot of memory space. Therefore, in order to adapt to stream data mining, this pattern must design an appropriate period of data elimination method. In addition, because the start time of the landmark window model is fixed, the information contained in the stream data will drift with time. Therefore, the accuracy of data pattern mining based on landmark window model algorithm is not high.

In the time attenuation window model [4], the attenuation weight method is used to reduce the impact on the data mode of the early data flowing into the window. Similarly, this mode also needs to determine the appropriate window period. On the one hand, it is necessary to store enough stream data for mining and analysis; on the other hand, it is necessary to use an efficient data structure to reduce the occupation of memory space.

In the sliding window model [5], when new data flows in, the early data can be discarded, so that the window data can satisfy the concept drift processing. However, if the period of sliding window is too large, the concept drift of stream data in the window may have occurred, so the accurate data pattern mining can not be carried out in this case. On the other hand, if the window period is too small, the data patterns in the stream data can not be mined.

In view of the limitations of the sliding time window model, this paper proposes a novel algorithm SWFIUT-stream for mining frequent itemsets in stream data. The algorithm sets the size of sliding window according to the period of application data, and introduces the time attenuation factor to set different weights for window units, and uses the improved FIUT algorithm [6] to mine and analyze the frequent itemsets of stream data in the window. In addition, the algorithm will be implemented on storm platform, and the performance of the algorithm will be improved through distributed parallel computing [7].

AIPR 2020, June 26–28, 2020, Xiamen, China

ACM ISBN 978-1-4503-7551-1/20/06…$15.00
DOI: https://doi.org/10.1145/3430199.3430226

2. RELATED RESEARCH

2.1 Frequent Itemset Mining for Stream Data

Mining frequent itemsets of massive stream data is a hot topic in the field of big data mining. Although there are many researches and applications on frequent itemsets mining, there are still many challenges to be solved. Stream data has the characteristics of continuity, real-time and unbounded. In practical application, the knowledge or pattern implied in stream data will change with time. Therefore, the impact of reducing historical data should be considered in data mining [8]. When mining frequent itemsets of stream data, infrequent items are likely to become frequent items in the near future, and some frequent items may become infrequent items over time [9]. Therefore, the frequent itemsets of stream data need to be updated periodically.

In recent years, many algorithms for mining frequent items in stream data have been proposed. For example, Wang Jinwei et al. [10] proposed CFMoment algorithm, which maintains frequent closed itemsets in the data stream through sliding windows, and uses bit sequences to reduce memory space occupation. However, the algorithm can't mine the frequent itemsets accurately from the stream data with concept drift. Sun Dujing et al. [11] proposed DPFP-stream algorithm. The algorithm is improved on the basis of FP-stream algorithm, maintaining the global frequent pattern tree of data stream in memory, and embedding inclined time window in the pattern tree. However, when mining frequent itemsets, the algorithm needs to traverse the pattern tree, which costs a lot of time and space. Yin Shaohong et al. [12] proposed a SWM-MFI algorithm. The algorithm uses the transaction matrix to store the flow data in the sliding window, and establishes the binomial set matrix through the transaction matrix, so as to mine all the frequent itemsets iteratively. Although the algorithm optimizes the data structure of sliding window and saves storage space, it needs to scan the binomial matrix repeatedly when mining frequent itemsets, which increases the time cost. In addition, the algorithm does not consider the weight attenuation of the early stream data in the window. When the stream data in the window has been concept drifted, it is difficult to accurately mine frequent itemsets.

In conclusion, in order to improve the algorithm of mining frequent itemsets, it is necessary to optimize the storage structure of streaming data in sliding window to reduce the memory cost of algorithm processing. Then it is necessary to optimize the frequent itemsets mining algorithm to avoid some complex redundant calculations and improve the processing efficiency of frequent itemsets. Finally, we need to weaken the impact of historical transactions on data mining patterns, so that the algorithm can obtain accurate frequent itemset result data.

2.2 Related Definitions

In order to better understand the algorithm proposed in this paper, some basic terms are defined here.

Definition 1: A sequence is an ordered list of transactions, which can be counted as S = {t_1, t_2, t_3, …t_i, … }. Each t_i is a set family of one or more items.

Definition2: The stream data sequence DS is a transaction sequence that continuously arrives according to time. It is expressed as DS = {t_1, t_2, t_3, … , t_i, … }. Where t_i represents i-th arriving transaction.

Definition 3: Let DS be a stream data sequence and itemset X be a subset of its transaction t_i. The number of transactions that contain itemset X is called the support number of X, denoted P(X). The ratio of P(X) to the total number of DS transactions is called the support degree of X, which is denoted as σ(X). The calculation method is shown in Formula 1.

$$\sigma(X) = \frac{|\{t_i | X \subseteq t_i, t_i \in DS\}|}{|DS|} \quad (1)$$

Definition 4: Given the minimum support ε (0 < ε < 1). If σ(X) > ε, then X is called a frequent itemset in DS. Otherwise, X is called the infrequent itemset of DS. The frequent itemsets with length k are called k-frequent itemsets.

Definition 5: The sliding time window SW consists of multiple window units, denoted as BW_i, i = 1.. n. The size of the window unit is the number of transactions that each BW_i can hold, which is counted as $|BW_i|$. The transactions in DS flow into the window in chronological order. When a new transaction enters the window, the earlier transaction will be squeezed out of the window. Here, a window with two units is taken as an example to illustrate the principle of sliding window, as shown in Figure 1.

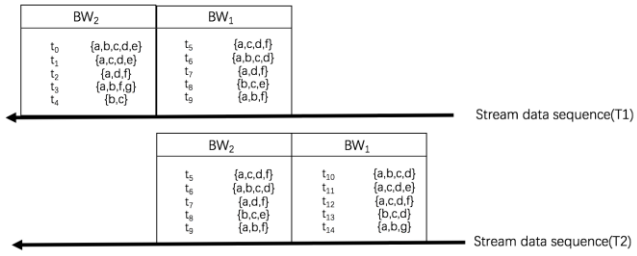

Figure 1. The sliding time window.

At time T1, the data stream {t_0, t_1, t_2,…, t_9 }. It flows into the sliding window in time series. Where t_9 is the most recent transaction into the window, t_0 is the first transaction to flow into the window. As time goes on, at T2, when a new transaction t_{15} flows into the window. The earliest transaction data in the sliding window, t_5 will crowd out the window.

Definition 6: In order to weaken the influence of the early data of sliding window on the mining accuracy of frequent itemsets, the time attenuation factor η (0 < η < 1) is introduced. Different window units are given different weights by time attenuation factor. Suppose there are n window units in SW, and each window unit is numbered BW_1, BW_2, … , BW_n. Where BW_1 represents the window unit nearest to the current time. BW_n is the window unit farthest from the current time. The calculation method of transaction data weight in BW_i window unit is shown in formula 2.

$$W(BW_i) = \eta^{i-1} \quad (2)$$

After introducing the attenuation factor, the calculation of support number $P_{SW}(X)$ and support $\sigma_{SW}(X)$ of itemset X in sliding time window are shown in Formula 3 and formula 4 respectively.

$$P_{SW}(X) = \sum_{i=1}^{N} \sigma_{BW_i}(X) \times |BW_i| \times \eta^{i-1} \quad (3)$$

$$\sigma_{SW}(X) = \frac{P(X)}{\sum_{i=1}^{N} \eta^{i-1} \times |BW_i|} \quad (4)$$

3. SWFIUT-STREAM ALGORITHM

In this paper, the SWFIUT-stream algorithm scheme is proposed for the data processing model of sliding time window, data updating and weight processing, frequent itemset mining method, and parallel computing implementation based on strom framework.

3.1 SWFIUT-Stream Data Processing Model

The data processing model of SWFIUT-stream algorithm proposed in this paper consists of two parts: 1-itemset count table and window unit transaction two-dimensional table.

The 1-itemset counter table is used to record the items in the sliding time decay window and the number of times they appear in the window. An example is shown in Table 1 below.

Table 1. 1-Itemset counter table data

Item	Counter
a	8
b	6
c	6
d	6
e	3
f	5
g	1

In the 1-itemset counter table, the item column lists all items that appear in the window. The counter column gives the number of times each item appears in the window. When the window slides, the item and counter columns are updated accordingly.

Sliding window SW stores data by transaction two-dimensional tables of multiple window units. In the window unit transaction two-dimensional table, each column holds the item value of DS transaction of stream data sequence flowing into window unit. Each row stores a transaction of the stream data sequence DS flowing into the window unit. Tid is the timestamp of the transaction. In the window unit transaction two-dimensional table, if a transaction contains an item, the corresponding item column cell of the transaction row is set to 1, otherwise, the item cell is set to 0. For example, in window unit BW_1, a transaction with Tid 9 contains a, b, f items. When the transaction flows into the window unit, the column cells of items a, b, and f in the row data need to be set to 1, and the column cells of other items a, e and g should be set to 0, as shown in Table 2.

Table 2. Data of BW_1 table

Tid	a	b	c	d	e	f	g
5	1	0	1	1	0	1	0
6	1	1	1	1	0	0	0
7	1	0	0	1	0	1	0
8	0	1	1	0	1	0	0
9	1	1	0	0	0	1	0

3.2 Data Updating and Weight Processing of Sliding Time Window

Due to the fixed size of the sliding window. When there is data flowing into the window, there must be data flowing out of the window. Therefore, the data stored in the BW_i table is updated synchronously with the flow of stream data. For example, the updated data of BW_1 table and BW_2 at T2 time are shown in Table 3 and Table 4.

Table 3. Data of BW_1 table at T2 time

Tid	a	b	c	d	e	f	g
10	1	1	1	1	0	0	0
11	1	0	1	1	1	0	0
12	1	0	1	1	0	1	0
13	0	1	1	1	0	0	0
14	1	1	0	0	0	0	1

Table 4. Data of BW_2 table at T2 time

Tid	a	b	c	d	e	f	g
5	1	0	1	1	0	1	0
6	1	1	1	1	0	0	0
7	1	0	0	1	0	1	0
8	0	1	1	0	1	0	0
9	1	1	0	0	0	1	0

When the new data flows into the window, the counting table of 1-itemset needs to be updated. The 1-itemset counting table can change synchronously with the sliding window. The update method is to subtract 1 from the counter column value of the 1-itemset count table for the items appearing in the transaction in the outflow window at this time. Add 1 to the counter column value of the 1-itemset count table corresponding to the new transaction inclusion item flowing into the window. Table 5 shows the updated 1-itemset count table data at T2 time.

Table 5. 1-Itemset counter table data at T2 time

Item	Counter
a	8
b	6
c	7
d	6
e	2
f	4
g	1

In addition, in order to reduce the impact of early transactions on the accuracy of data mining, it is necessary to reduce the weight attenuation of data in the unit table of earlier windows. According to definition 6, this algorithm designs different weights for different BWi tables. For example, if the transaction ti is in the window unit BWi, the support number of this piece of data needs to be calculated according to the BWi window unit's weight. As the window slides, the data will flow into the following window unit, and its support number will be further reduced until it is squeezed out of the sliding window.

3.3 Mining Frequent Itemsets Based on Sliding Decay Time Window

Traditional FIUT algorithm [13] needs to scan window data twice. In the first scan, all transaction data in the window are read one by

one. Count the number of elements in each transaction, and judge whether the count value of each item reaches the set support number. If reached, the item is marked as a 1-frequent itemset. All 1-frequent itemsets can be obtained by judging the count value of each item. In the second scan, the itemsets collection is initialized, which is used to store the intermediate results of frequent itemsets mining. When each data is processed one by one, the items that are not in the 1-frequent itemset are first deleted. An itemset consisting of only frequent items is obtained. If the length of the itemset is k, it is called k-itemset. Put the k-itemset of this transaction data into itemsets collection. After each transaction is processed in this way, the itemsets collection will store the k-itemsets corresponding to all transaction data in the window. After that, we need to process the intermediate result data stored in the itemsets collection to mine all frequent itemsets.

For the traditional FIUT algorithm, it starts from the k-itemset in the itemsets collection. Use them to build the k-FIU-tree. Determine whether the support number of leaf nodes of the tree is greater than the minimum support number. If it is greater than, it means that the itemset composed by the path is a frequent itemset. Then, all k-itemset of length k in itemsets collection are decomposed into several (k-1)-itemset by enumeration. The (k-1)-FIU tree is constructed by combining the decomposed (k-1)-itemset with the original (k-1)-itemset in itemsets collection. Then all (k-1)-frequent itemsets are mined. And so on until k = 2.

It can be seen that the traditional FIUT algorithm needs to operate the itemsets repeatedly and iteratively to get the frequent itemsets of all lengths in the descending order of k value. Every i-frequent itemset mining depends on the mining results of the previous (i+1)-frequent itemset, which is not conducive to parallel mining, and the operation is cumbersome.

In order to improve the execution efficiency of frequent itemsets mining, the FIUT algorithm is optimized as follows. First of all, the algorithm can directly determine whether the counter value of each item item is greater than the set support number from the 1-itemset counter table, so as to determine the 1-frequent itemsets in the sliding window. It is no longer necessary to scan all transaction data in the window again, which can reduce the computational cost. Secondly, different from the traditional FIUT algorithm, this algorithm decomposes the k-itemsets into (k-1)-itemsets, (k-2)-itemsets,...,2-itemsets. And insert them into (k-1)-FIU-tree, (k-2)-FIU-tree,...,2-FIU-tree.To avoid the problem that traditional FIUT algorithm needs repeated operations on different length itemsets to construct k-FIU-tree. Each transaction data in the window is processed in this way. When all transaction data in the window are processed, k-FIU-trees with different k values can be obtained. Finally, after getting all the k-FIU-trees, the algorithm can process their leaf node records. If the support number of the leaf node record is greater than the set minimum support number, the itemset composed of the path to which the leaf node belongs is a frequent itemset. In this way, frequent itemsets of all lengths can be mined out.

The improved frequent itemsets mining algorithm no longer depends on repeated iteration itemset decomposition. Instead, k-FIU-trees of various lengths are directly constructed in the process of scanning data. This can solve the problem that the traditional algorithm decomposes the frequent itemsets repeatedly. At the same time, the mining of frequent itemsets with different lengths can be carried out independently, and the parallel mining processing can be realized, which can improve the efficiency of the algorithm.

For example, suppose the minimum support threshold of frequent itemsets is 0.6, the total number of window units is 2, each window can hold 5 pieces of data, and the time attenuation factor $\eta = 0.8$. In the sliding window, there are two window unit tables BW1 and BW2, which store 10 transaction data. The weight of BW1 is 1 and that of BW2 is 0.8. According to formula 3, the minimum support number of the window is 5.4, that is, the itemset with more than 5.4 support number is frequent itemset.

At T2 time, frequent itemsets mining is started. Firstly, the row whose count value is less than the minimum support number in the 1-itemset counter table is deleted, and the remaining four rows a, b, c and d are 1-frequent itemsets. The first transaction data in the window is processed, and a 3-itemset (a, c, d) is obtained. According to the weight of its unit window, the support number of this transaction data after attenuation is calculated to be 0.8. Put it in the 3-FIU-tree. If the path a, c, d already exists in 3-FIU-tree, add 0.8 to the support number of leaf node d of this path. If this path does not exist on the 3-FIU-tree, the path is first created in the 3-FIU-tree, and the leaf node count value is given 0.8. Then it is decomposed into three 2-itemsets (a, c), (a, d) and (c, d). The support number of the three 2-itemsets is 0.8. Similarly, insert these 2-itemsets into 2-FIU-tree. When the first transaction data ends processing, the next transaction data will be processed. When all transaction data in the window are processed in this way, 2-FIU-tree, 3-FIU-tree and 4-FIU-tree are finally obtained as shown in figures 2, 3 and 4. The leaf node records in these k-FIU-trees record the support number of itemsets in the path.

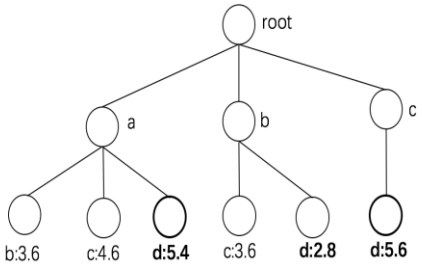

Figure 2. 2-FIU-tree.

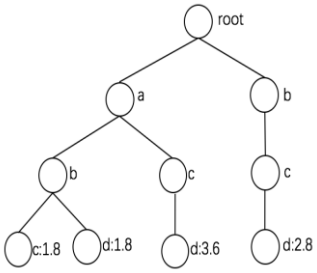

Figure 3. 3-FIU-tree.

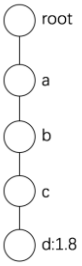

Figure 4. 4-FIU-tree.

After establishing k-FIU-trees with different K values, the support number of all leaf nodes were judged. The path of leaf node whose support number is more than 5.4 can be used as the frequent itemset and put into the k-frequent itemset table. Finally, different k-frequent itemsets are obtained, as shown in Table 6.

Table 6. K-frequent itemsets table

K-frequent item set	Items	Support Number
2-frequent item set	(a,d)	5.4
	(c,d)	5.6
1- frequent item set	a	7.2
	b	5.4
	c	6.2
	d	7.2

The pseudo code of SWFIUT-stream algorithm is as follows:

SWFIUT-stream algorithm: Mining frequent itemsets in stream data

Input：Stream data, minimum confidence ε
Output：K-frequent itemsets table
1: Initialize k-FIU-tree
2: Initialize k-frequent itemset table k_frequent_itemset
3: for i=1 … all BW_i do:
4: for each item in BW_i { Read every transaction data in BW_i }
5: k_item=item.delete() { The k-itemset is obtained by deleting all the infrequent items in the transaction data }
6: k=k_item.length() { Get the length value k of k-itemset }
7: k_item.support=supportNumber(i) { According to the window unit weight of the transaction, the support number of the transaction decay is given }
8: for j=k … 3,2 do:
9: j-items[]=k_item.factoring(j) { k-item is decomposed into several itemsets of length J and put into j-items [] set }
10: k-FIU-tree.build(j-items[]) { Insert the decomposed j-itemset into the corresponding j-FIU-tree }
11: end for
12: end for
13: end for
14: for each leaves in k_FIU_tree { Traversing all leaf nodes on k-FIU-tree }
15: support = leaves.getSupport() { Get the support number of leaf nodes }
16: if support >= ε then
17: k_frequent_itemset.add(leaves.path()) { The frequent itemsets in the path of the leaf node are inserted into the k-frequent itemsets table }
18: end for
19: return k_frequent_itemset{ Output k-frequent itemsets table }

3.4 Parallel Computing Implementation of SWFIUT-stream Algorithm

In order to mine frequent itemset quickly, this paper uses distributed parallel computing method to implement parallel processing of SWFIUT-stream algorithm in strom framework [14-15]. The running topology of SWFIUT-stream algorithm on strom framework is shown in Figure 5.

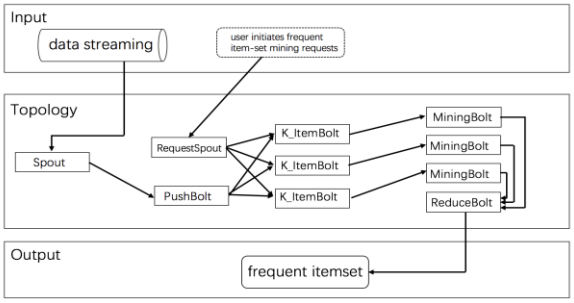

Figure 5. Running topology of SWFIUT-stream algorithm on strom framework.

In the implementation scheme of this algorithm, the stream data is received and sent to pushbolt. PushBolt is responsible for putting the transaction data into the BW_i table after processing and updating the items in the 1-itemset counter table.

When a user initiates a frequent itemset mining request, RequestSpout is responsible for receiving the user's request and executing K_ItemBolt handler. There will be multiple K_ItemBolt handler execute in parallel in the cluster. According to the minimum support degree threshold ε, they scan the 1-itemset counter table to get 1-frequent itemsets, process each transaction data and construct k-FIU-tree. The obtained k-FIU-tree is sent to MiningBolt for mining frequent itemsets. Finally, the frequent itemsets are integrated by ReduceBolt to get the final mining results.

4. EXPERIMENTAL ANALYSIS

4.1 Experimental Data and Operating Environment

This experiment runs on the server hardware environment of 2.8 GHz Intel Core i7 16GB memory. The SWFIUT-stream algorithm is written in Java language and compared with SWM-MFI algorithm and DStree algorithm. The simulation data set T10I4D100K generated by IBM data generator is used in the experiment. In the experiment, the number of window units is set to 10, and each unit window capacity is 10000 transactions. In order to obtain stable experimental data, the average value of 10 experimental results was used. This algorithm is compared with other two algorithms in running time and memory cost.

In order to verify the parallel performance of the algorithm, the speedup ratio of different cluster sizes is used to measure. Speedup ratio = execution time of algorithm program in single machine mode / execution time in cluster mode.

4.2 Comparison of Algorithm Running Time

In this experiment, three algorithms are operated in different support degree data sets, and the time cost of each algorithm in mining frequent itemsets under the same data set is compared. The result data is shown in Figure 6.

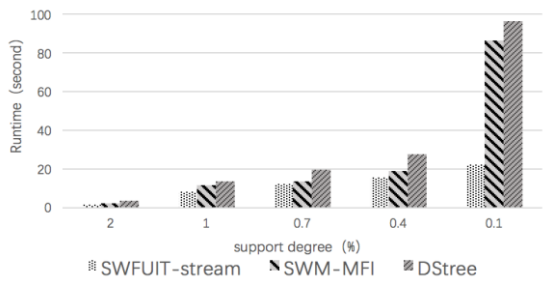

Figure 6. Time cost of each algorithm under different support degree levels.

From the running result data, it can be seen that the time cost of SWFIUT-stream algorithm is less than the other two algorithms in both high and low support degree. This is because SWM-MFI algorithm needs to construct binomial set matrix when mining frequent itemsets, and gradually expand the transaction matrix and binomial matrix to obtain the complex itemsets with higher dimension. The algorithm needs to scan the transaction matrix repeatedly for frequent itemsets mining. When mining frequent itemsets, DSTree algorithm needs to construct a tree structure similar to FP-Tree, which increases the time cost. The SWFIUT-stream algorithm only needs to scan the data in the sliding time attenuation window once and decompose the data to avoid redundant calculation and save time.

4.3 Memory Cost Comparison of Algorithm Running

This experiment also compares the memory space overhead of the three algorithms in the running process. The amount of memory space required for the three algorithms to run under different number of window units is shown in Figure 7.

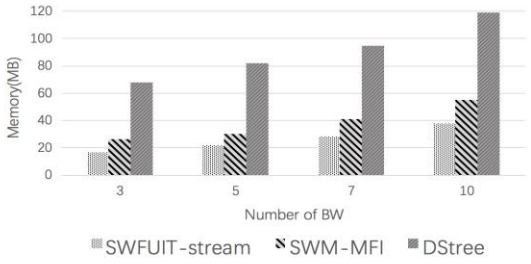

Figure 7. Memory cost of each algorithm under different number of window units.

According to the running result data, the memory consumption of SWFIUT-stream algorithm is less than that of the other two algorithms. This is because SWFIUT-stream uses a two-dimensional table structure to store data in the window. The SWM-MFI algorithm uses transaction matrix to store the data. Although it reduces the memory cost to a certain extent, it also constructs a binomial matrix to mine frequent itemsets, which increases the memory cost. The DSTree algorithm uses a tree data structure similar to FP-tree for data storage and recursion to mine and traverse FP-tree, which has a large space cost.

4.4 Parallel Execution Ability of the Algorithm

In order to improve the performance of the algorithm, it is implemented in the Strom framework. Compared with the speedup index of this algorithm running in different cluster node scale of multiple data sets processing. The results of the algorithm are shown in Figure 8.

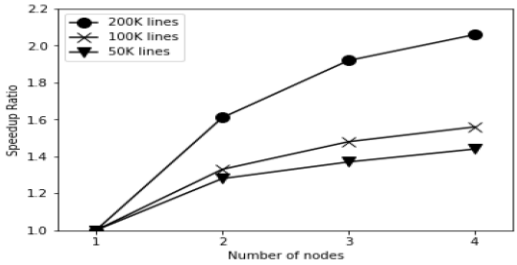

Figure 8. The speedup ratio of the algorithm under different number of cluster nodes.

The running results show that SWFIUT-stream algorithm has good parallel execution ability. With the increase of the number of nodes, the efficiency of the algorithm is higher. The larger the amount of data set, the more obvious the improvement of parallel execution efficiency. Therefore, this algorithm has good scalability.

5. CONCLUSION

In this paper, a novel algorithm SWFIUT-stream is proposed for mining frequent itemsets of stream data based on sliding time decay window. It uses window unit transaction two-dimensional table to store the stream data. The time attenuation factor is introduced to assign different weights to different window units. It makes it possible to mine frequent itemsets accurately. When mining frequent itemsets, k-itemsets of different lengths are decomposed directly and k-FIU-tree is constructed. Frequent itemsets are obtained by judging the support number of leaf nodes. Combined with Storm framework, the SWFIUT-stream algorithm is implemented in parallel, which can be applied to big data scenarios. The experimental results show that the proposed algorithm has the advantages of less memory cost, higher execution efficiency and good scalability. It can effectively support stream data mining frequent itemsets.

6. REFERENCES

[1] WANG Hongmei, LI Fentian, WANG Zeru. 2017. A summary of frequent item-sets mining model of data flow based on sliding window. *J. Journal af Changchun University of T echnology* 38, 5(Oct. 2017), 484-490. DOI: 10.15923/j. cnki. cn22- 1382/t.2017.5.14

[2] HAN Meng, DING Jian. 2019. Survey of frequent pattern mining over data streams. *J. Journal of Computer Applications* 39, 3(Mar. 2019), 719-727. DOI: 10.11772 /j. issn. 1001-9081. 2018081712

[3] WEN Yingyou, WANG Shaopeng, ZHAO Hongjie. 2017. The Maximal Regular Patterns Mining Algorithm Based on Landmark Windowover Data Stream. *J. Journal of Computer Research and Development* 54, 1(Jan. 2017), 94-110. DOI:10. 7544/ issn1000-1239. 2017. 20150804

[4] Ahsan Shah, Zahid Halim. 2019. On Efficient Mining of Frequent Itemsets from Big Uncertain Databases. *J. Journal of Grid Computing: From Grids to Cloud Federations* 17, 1(Jan. 2019), 831-850. DOI: 10.1007/s10723-018-9456-0

[5] RU Bei, HE Xinzheng. 2017. Efficient complete frequent itemsets mining algorithm of data stream. *J. Computer Engineering And Design* 38, 10(Oct. 2017) 2759-2766. DOI: 10.16208/j. issn1000-7024. 2017.10.030

[6] ZHANG Hang, ZHANG Xin, ZHANG Pingkang, LI Qi. 2019. Incremental updating algorithm of parallel frequent itemsets based on FIUT. *J. Application Research of Computers* 36, 7(Jul. 2019), 1991-1993. DOI: 10. 19734 /j. issn. 1001-3695. 2017. 12. 0854

[7] ZHANG Zhou, HUANG Guorui, JIN Peiquan. 2019. Task Scheduling on Storm: Current Situations and Research Prospects. *J. Computer Science* 46, 9(Sep. 2019), 28-35. DOI: 10. 11896/j. issn. 1002-137X. 2019. 09. 004

[8] ZHAO Xuejian, XIONG xiaoxiao, ZHANG Xinhui, SUN Zhixin. 2019. A Frequent Itemset Mining Algorithm for Uncertain Data Based on Top -K Queries. *J. Computer Technology And Development* 29, 07(Jul. 2019), 49-54. DOI: 10. 3969/j. issn. 1673－629X. 2019. 07. 010

[9] LIANG Wenjuan, CHEN Hong, ZHANG Jing, ZHAO Dan, LI Cuiping. 2020. An effective scheme for top- k frequent itemset mining under differential privacy conditions. *J. Science China Information Sciences* 63, 5(May. 2020), 25-36. DOI: https://doi.org/10.1007/s11432-018-9849-y

[10] WANG Jinwei, WU Shaohua, QU Zhiguo. 2019. CFMoment: Closed Frequent Itemsets Mining Based on Data Stream. *J. JOURNAL OF APPLIED SCIENCES—Electronics and Information Engineering* 37, 3 (May. 2019), 389-397. DOI: 10.3969/j.issn.0255-8297.2019.03.009

[11] SUN Dujing, LI Lingjuan, MA Ke. 2017. Realization and Implementation of Distributed Parallel Mining of Frequent Patterns for Data Streams. *J. Computer Technology And Development* 27, 7(Jul. 2017), 29-33. DOI: 10. 3969 /j. issn. 1673-629X. 2017. 07. 007

[12] YIN Shaohong, SHAN Kunyu, FAN Guidan. 2015. Mining algorithm research of data stream maximum frequent itemsets in sliding window. *J. Computer Engineering and Applications* 51, 22(Nov. 2015), 145-149. DOI: 10.3778/j.issn. 1002-8331.1312-0070

[13] YAN Yi, XU Su. 2019. Mining research on FIUT algorithm based on parallel entropy in Hadoop environment. *J. Computer Engineering And Design* 40, 3(Mar. 2019), 685-690. DOI: 10. 16208/j. issn1000-7024 2019. 03. 016

[14] YANG Zonglin, Li Tianrui, LIU Shengjiu, YIN Chengfeng, JIA Zhen and ZHU jie. 2020. Streaming Parallel Text Proofreading Based on Spark Streaming. *J. Computer Science* 47, 04(Apr. 2020), 685-690. DOI: 10.11896/jsjkx. 19030007

[15] LIU Su, YU jiong, LU liang, LI Ziyang. 2018. Task scheduling strategy based on topology structure in Storm. *J. Journal of Computer Applications* 38, 12(Dec. 2018) 3481-3489. DOI: 10. 11772 /j. issn. 1001-9081. 2018040741