MovStream: An Efficient Algorithm for Monitoring Clusters Evolving in Data Streams*

Liang TANG[1], Chang-jie TANG[1], Lei DUAN[1], Chuan LI[1],
Ye-xi JIANG[1], Chun-qiu ZENG[1] and Jun ZHU[2]

[1]*School of Computer Science, Sichuan University*
Chengdu, P.R.China, 610065
{tangliang, tangchangjie}@cs.scu.edu.cn

[2]*Birth Defects Supervising Centre of Western China Medical School, Sichuan University,*
Chengdu, P.R.China, 610065

Abstract

Monitoring cluster evolution in data streams is a major research topic in data streams mining. Previous clustering methods for evolving data streams focus on global clustering result. It may lose critical information about individual cluster. This paper introduces some basic movements of evolution of an individual cluster. Based on the measurement of the movements, a novel algorithm called MovStream is proposed to monitor clusters' evolving in data streams. The experimental results on real datasets show that our MovStream algorithm surpasses the well-known CluStream algorithm by 25-50% in accuracy and one order of magnitude in efficiency.

1. Introduction

The evolution of data clusters in data streams can be used to analyze the trends of the continuous data streams. Recently, several data clustering algorithms for data streams have been introduced [2, 3, 6-12]. Most of them focus on finding the centers of clusters and building clusters. However, in many applications, the information about fashions of movements and changes on clusters is also very useful. It provides a better understanding on how and why data streams change. *CluStream* is a well-known clustering algorithm that can handle this information in evolving data streams [4]. It proposes a new concept of *pyramidal time frames* and stores the properties of micro-clusters at particular time as snapshots. According to subtractive properties of the micro-cluster, it is convenient to get the clustering result in a past time-horizon. In the applications of detecting and monitoring data streams, the information about appearance and disappearance of clusters may not be enough. Discovering how a new cluster comes from and how an old cluster changes are interesting.

This paper proposes a new method named *MovStream* to monitor the data clusters in evolving data streams over the sliding windows. Previous clustering methods often use the global clustering strategy to monitor the synopsis of evolving data stream. However, each cluster has its own evolving behavior, and the behavior contains practical meanings in certain situation. The global uniform method may lose some critical information. Our method provides evolving information of each individual cluster in data streams. The evolving behavior of each cluster is treated as a movement of the cluster. Further, we introduce several measurements for basic movements. These measurements defined based on the properties of data cluster were introduced in previous literatures [4, 5]. We refer significant movements to *movement events*. There are corresponding operations to refine the clustering result when a *movement event* is detected. These operations include *move*, *split, merge, stretch* and *delete*. As the *split* and *merge* operations in B^+ Tree keep the balance of the indexing tree, these operations keep the accuracy of clustering result in evolving data streams.

In Fig. 1, the position of the cluster center changes. This kind of movement is defined as *drift*. Suppose the data points of data streams follow a *Gaussian* distribution, when the *mean* parameter of the *Gaussian* model is changed, there should be a *drift* of movement to the cluster of these data points in data streams. In

* This work is supported by the National Natural Science Foundation of China under grant No. 60773169, the 11th Five Years Key Programs for Sci. &Tech. Development of China under grant No. 2006BAI05A01 and the Youth Foundation of Sichuan University No. 06036.

Fig. 2, two clusters are the same cluster with two time stamps. They have the same *centroid*, but their *radiuses* are different. The *radius* is defined as the average distance from data points to the *centroid* that will be introduced later. The movement comes from different *radius* of the cluster will be defined as *expand* or *shrink*. For example, if the cluster follows a *Gaussian* distribution, *expand* or *shrink* can be seen as an alteration of the *variance* parameter. There is also a straightforward way to increase or decrease the *radius* of the cluster.

Motivated by these observations, we propose a heuristic strategy to detect and monitor the actions of the clusters in evolving data streams including the movements introduced in Fig. 1 and Fig. 2.

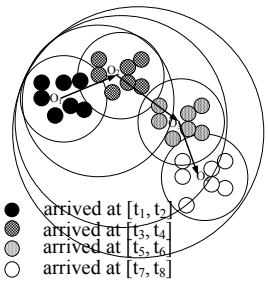

Fig.1 The *drift* to a cluster

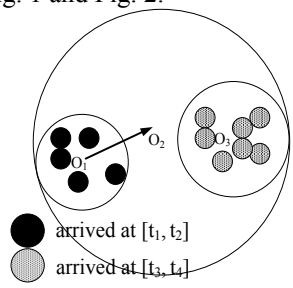

Fig.2 The *expand* to a cluster

The main contributions of this paper are summarized as follows. First, we propose the *movement event* with the measurement to the cluster. It provides valuable information about evolving behavior of each cluster in evolving data streams. Second, we design a novel data stream clustering method *MovStream.* It consists of the *movement event* detecting algorithm and the clustering refinement operations. The time and space complexities of *movement event detecting* algorithm are constant. Since the clustering refinement operations are invoked only when a *movement event* is triggered, the cost of these operations will be small in the situation of stable data streams.

2. Related Work

Data streams clustering methods require the scalability and efficiency in the data processing. Most single pass clustering approaches are under the assumption that the data points arrived in several chunks [2, 3, 7, 8]. Guha et al. proposed a *k*-means based algorithm for clustering data streams [3, 7]. The algorithm only makes a single pass over the data stream and uses small space. It requires $O(kN)$ time and $O(N^{\varepsilon})$ space, where k is the number of centers, N is the length of data stream, and $\varepsilon < 1$. O'Chalaghan et al proposed the STREAM algorithm to cluster data streams [2]. The algorithm is similar to Guha's. Nasraoui et al. proposed an artificial immune system (AIS) based clustering approach TECNO-STREAMS [9].

Aggarwal et al. proposed *CluStream* algorithm [4]. It adopts micro-clusters introduced in BIRCH algorithm and uses micro-clusters to absorb arrived data points in online step. The offline step is to use *k*-means [1] to cluster the micro-clusters into macro-clusters. The properties of micro-clusters are subtractive, so that according to two snapshots of micro-clusters, people can get clustering result on every past time-horizon. However, when the purpose of the algorithm is monitoring clusters in data streams, previous work has two weaknesses. First, the evolving of micro-clusters cannot represent the evolving of individual natural clusters. Second, the natural clusters cannot be clustered from micro-clusters in a straightforward way. For monitoring clusters, the macro-clustering is not executed in offline step. It should be executed to monitor natural clusters every period time. So it is not efficient in this situation.

3. Preliminaries

Previous literatures gave some preliminary concepts and terminologies of the data cluster [4, 5]. Suppose $|X_i - X_j|$ stands for the distance between two given data points X_i and X_j. Given a cluster C of data points $\{X_i\}$, where $i = 1, 2\ldots, N$, the *centroid* X_0, the *radius* R and the *diameter* D of the cluster are defined as follows.

$$X_0 = (\sum_{i=1}^{N} X_i) / N \quad (1)$$

$$R = (\frac{1}{N}\sum_{i=1}^{N} | X_i - X0 |^2)^{\frac{1}{2}} \quad (2)$$

$$D = (\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1}^{N} | X_i - X_j |^2)^{\frac{1}{2}} . \quad (3)$$

where, R is the average distance from data points to the centroid. D is the average pairwise distance within a cluster. They are two alternative measures of the tightness of the cluster around the *centroid*. In this paper, we use these properties to monitor the changes in data streams.

Given two clusters C_i and C_j, let $X0_i$ and $X0_j$ be the *centroids* of C_i and C_j. Then the *mean distance* of C_i and C_j is defined as:

$$d_{mean}(C_i, C_j) = | X0_i - X0_j |. \quad (4)$$

In our model, each cluster has a sliding window to maintain recent data points. The sliding window is implemented by a dynamic-length queue. The element of the queue contains the data point and it's time stamp.

The sum of the lengths of queues of all clusters is fixed by a constant parameter *horizon time* given by the user. It was introduced in previous literature [4]. When a new arrived data point is inserted to a cluster, the oldest data point in the *horizon time* in one of the clusters should be popped out.

Each data point has a time stamp. There is a constraint for the sliding window. Suppose X_n is the nth element in the sliding window. Two data points X_i and X_j with time stamp T_i and T_j are in one sliding window, if $i < j$, there must be $T_i < T_j$.

Notation. Given a sliding window *SW* of a cluster *C* and a time stamp *t*, $S(SW)_t$ denotes the set of data points $X_1 \ldots X_n$ in *SW* at time stamp *t*, $C(SW)_t$ denotes the cluster of $S(SW)_t$. $|SW|$ denotes the number of data points in *SW* and the length of *SW*. *LS(SW)* denotes the linear sum of data points in *SW*, and *SS(SW)* denotes the square sum of data points in *SW*. i.e., $LS(SW) = \sum_{i=1}^{n} X_i$, $SS(SW) = \sum_{i=1}^{n} X_i^2$.

Lemma 1. Given a sliding window *SW* of a cluster *C* with time stamp *t*, let $X_1, X_2 \ldots, X_N$ be data points of *SW* where $N = |SW| > 1$. The *centroid*, *radius* of cluster *C* can be calculated as follow:

$$X0 = LS(SW) / N \quad (5)$$

$$R = \sqrt{\| SS(SW)/N - (LS(SW)/N)^2 \|_1} \quad (6)$$

Due to space limitation, we omit proof here. The detail can be found in [13].

Lemma 1 gives the method to calculate the *centroid* and *radius* according to *LS* and *SS*. When a new data point arrives at a cluster, *LS(SW)* and *SS(SW)* of the sliding window can be updated in a straightforward way, whose time complexity is $O(d)$. So the cost of calculating the new *centroid* and *radius* of the cluster are also $O(d)$, where *d* is the number of dimensions.

4. Monitoring Clusters Evolving

As new data comes continuously, the data points in the sliding window and the cluster are always altered. Based on the properties of *cluster* introduced before, we define measurements for movements as follows:

Definition 1 (Cluster Movement) Given a sliding window *SW* of a cluster *C* and two time stamp *t* and *t'* ,$t < t'$, if $S(SW)_t \neq \varnothing, S(SW)_{t'} \neq \varnothing$, let R_t and $R_{t'}$ be the *radiuses* of $C(SW)_t$ and $C(SW)_{t'}$ ($R_{t'} > R_t$). *drift*, *expand, shrink* from *t* to *t'* are defined as follows:

(1) $drift_C(t, t') = d_{\text{mean}}(C(SW)_t, C(SW)_{t'})$
(2) $expand_C(t, t') = R_{t'} - R_t$
(3) $shrink_C(t, t') = R_t - R_{t'} = -expand_C(t, t')$
(4) $decline_C(t, t') = 1 - |S(SW)_{t'}| / |S(SW)_t|$

Based on observations, only significant movements are meaningful and worth to detect in practical applications. We introduce some significant movements and refer it to *movement events*. To control the sensitivity of the *movement event* detecting, we introduce a predefined parameter set $\{\delta_i\}$ ($\delta_i > 0$) where $i = 1 \ldots 5$.

Definition 2 (Movement Event) Given a sliding window *SW* of a cluster *C* and two stamp *t* and *t'* ($t < t'$), let the user-defined *sensitivity parameter set* be $\{\delta_i\}$ ($\delta_i > 0$) where $i = 1 \ldots 5$, and R_t be the *radius* of cluster *C* at time stamp *t*. The conditions for *movement event* at time stamp *t'* are defined as follows:

- *Drift event:* $S(SW)_t \neq \varnothing, S(SW)_{t'} \neq \varnothing$, $drift_C(t, t') > R_t \cdot \delta_1$.
- *Expand event*: $S(SW)_t \neq \varnothing$, $S(SW)_{t'} \neq \varnothing$, $expand_C(t, t') > R_t \cdot \delta_2$.
- *Shrink event*: $S(SW)_t \neq \varnothing, S(SW)_{t'} \neq \varnothing$, $shrink_C(t, t') > R_t \cdot \delta_2$.
- *Die-out event:* $S(SW)_t \neq \varnothing$, $decline_C(t, t') > \delta_3$.$(0 < \delta_3 < 1)$

Note that, Parameters δ_4 and δ_5 are not used in the definition of *movement event*. We will discuss them in sub-section 5.2. In data streams, there may be more than one *movement event* to trigger. The same type of *movement event* except *die-out event* may be generated several times to accomplish a large movement. The type of *movement event* may be not exclusive either. So that the *drift event* and *expand event* will be generated alternately during the change.

Definition 3 (ClusterState) Given a sliding window *SW* of a cluster *C*, the data points in *SW* are $X_1, X_2 \ldots X_N$ with time stamps $T_1, T_2 \ldots, T_N$, the *cluster state* of the cluster is a 6-tuple *ClusterState* = (*SW, LS, SS, t,* OX_0, *OR*) defined by:

- *SW* is the sliding window of the cluster.
- *LS* is *LS(SW)* of the sliding window *SW*.
- *SS* is *SS(SW)* of the sliding window *SW*.
- *t* is the time stamp when the last cluster refinement operation be executed.
- OX_0 is the original *centroid* of the cluster, which is the *centroid* at time stamp *t*.
- *OR* is the original *radius* of the cluster, which is the *radius* at time stamp *t*.

ClusterTable *ClusterTable* is a dynamic list data structure that stores the *ClusterState* tuple of each cluster in data streams. In our work, *ClusterTable*[*i*] presents the *ClusterState* tuple of cluster *i*.

Algorithm 1 describes the first phase of our method. It is enlighten by the idea of *k*-means clustering to initialize clusters [1].

Algorithm 1 InitClusterTable (*k, InitNumber*)
1 Store first *InitNumber* data points.
2 cluster the *InitNumber* data points into *k* clusters.
3 **for each** initial cluster *i* **do**
4 Create the *ClusterTable*[*i*] from the cluster *i*.
5 Fill *ClusterTable*[*i*].*SW*.
6 Calculate *LS*, *SS* of *ClusterTable*[*i*].
7 Calculate *centroid*, *radius* of *ClusterTable*[*i*]
8 *ClusterTable*[*i*]. *t = 0*

Algorithm 2 describes how to process the new arrived data, where X_{new} is from data stream and X_{last} is the oldest data point in all clusters. The time complexity of line 1 and 4 is less than O(*MaxNumCluster*), where *MaxNumCluster* is a user-defined factor that is maximal the number of clusters in *ClusterTable*. Algorithm 2 is a constant time complexity cost.

Algorithm 2 ProcessNewData (X_{new})
1 Find the nearest cluster *i* in *ClusterTable* to X_{new}.
2 Add X_{new} and $X_{new} \cdot X_{new}$ to *LS* and *SS* of *ClusterTable*[*i*]
3 Push X_{new} in *ClusterTable[i].SW*
4 Find the cluster *j* contains X_{last} in *ClusterTable.*
5 Pop out X_{last} from *ClusterTable*[*j*].*SW*.
6 Subtract X_{last} and $X_{last} \cdot X_{last}$ from *LS* and *SS* of *ClusterTable*[*j*]

Based on *ClusterTable* data structure and *sensitivity parameter set*, we propose an algorithm for detecting and refinement operations to of clusters in data streams. The refinement operations guarantee the accuracy of the clustering result.

Move The *move* operation is executed when a *drift event* occurs. For example, in Fig. 1, it revises the original *centroid* of the cluster to current *centroid* according to *LS*.

Split This operation is to split the cluster into two smaller clusters if the *radius* is larger than maximum of *radius*. It is executed when *expand event* is triggered. For example, from Fig. 3 to Fig. 4, the cluster O_1 is split into cluster O_3 and O_4. We use δ_4 that is one of the given *sensitivity parameter set* to represent the limit of maximal *radius* in data streams. Given two *ClusterState* tuples $CS_i = (SW_i, LS_i, SS_i, t_i, OX_{0i}, OR_i)$ of cluster *i*, if $OR_i > \delta_4$ then the *split* operation will be executed to the cluster *i*. The splitting method consists of two $O(|SW|)$ steps. The first is to find out the farthest data point from the center in *SW*. The second step is to make the farthest data point be a seed to absorbing data points in *SW*. Some data points which are nearer to the seed than to the center will be assigned to the seed. Finally, the seed with its data points becomes a new cluster. New *ClusterState* tuple of the new cluster will be created and inserted to the *ClusterTable*.

Merge The *merge* operation is to merge two near clusters. It is executed when *drift event* or *expand event* is triggered. For example, from Fig. 3 to Fig. 5, the cluster O_1 and cluster O_2 are merged into cluster O_1. Given two *ClusterState* tuples $CS_i = (SW_i, LS_i, SS_i, t_i, OX_{0i}, OR_i)$ of cluster *i* and $CS_j = (SW_j, LS_j, SS_j, t_{0j}, OX_{0j}, OR_j)$ of cluster *j*, if $(OR_i + OR_j) \cdot \delta_5 > |OX_{0i} - OX_{0j}|$ then the *merge* operation will be executed to merge cluster *i* and *j*. The sliding windows of the two clusters will be also merged by a merge sort. The time complexity of *merge* is $O(|SW_i| + |SW_j|)$.

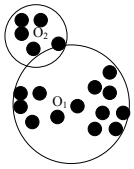

Fig.3 Two clusters

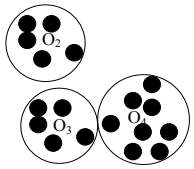

Fig.4 After split

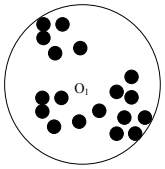

Fig.5 After merge

Stretch The *stretch* operation is executed when *shrink event* or *expand event* occurs. This operation will update the original *radius* by current *radius* of the cluster. Current *radius* can be calculated by Lemma 1.

Delete The *delete* operation is executed when *die-out event* is triggered. This operation firstly distributes all the data points with their time stamps of the cluster to other cluster. Every data point will be assigned to the nearest one of other clusters. Then it deletes the *ClusterState* of the cluster in the *ClusterTable*.

Algorithm 3 describes maintaining the clusters and *ClusterTable*. It is an event-driven process that checks each condition of *movement event* and trigger corresponding operations introduced in this subsection. There are two integer parameters *MaxNumCluster* and *MinNumCluster* that limit the range of the number of clusters in data streams processing.

Algorithm 3 Maintain($\{\delta_i\}$, *MaxNumCluster, MinNumCluster*)
1 **for each** cluster *i* **do**
2 **while** exist any *movement event* of cluster *i* is triggered **do**
3 *event* = the *movement event* triggered
4 Invoke corresponding operations to *event*
5 **if** the number of clusters < *MinNumCluster*

6 Invoke *split* to the maximal *radius* cluster.
7 **if** the number of clusters > *MaxNumCluster*
8 Invoke *merge* to the two closest clusters.

The time complexities of all refinement operations are less than O(*h*), where *h* is the constant parameter *horizon time*. Furthermore, the process of Algorithm 3 doesn't need to be executed when every new data points arrive in. The user can define a parameter *checkinterval* as the number of data points in the interval of maintaining process execution.

5. Experimental Study

All experiments were conducted on a PC with Intel Pentium IV 3.20GHz processor and 2 GB memory, which runs Windows XP. The well-known *CluStream* [4] was used as the benchmark algorithm, and all algorithms were implemented in standard C++.

(1) Real datasets. In order to demonstrate the effectiveness of *MovStream*, the KDD-CUP'99 Network Intrusion Detection data set was used. Aggarwal et al [4] used the data set to evaluate the accuracy of *CluStream* with respect to STREAM [2]. As Aggarwal and Callaghan et al. did in [4] and [2], the 34 continuous attributes of total 42 attributes were selected for the experiments.

(2) Synthetic datasets. To demonstrate the scalability of *MovStream*, we generated several synthetic dataset which follows a series of Gaussian distributions. The size of dataset varies from 100K to 400K, the number of clusters varies from 5 to 20, and the dimensionality of data points varies from 4 to 50. The mean and variance of the Gaussian distributions are changed every 10K data points. The quality of clustering on data sets is measured by the sum of square distance (SSQ), which is used in many previous literatures [2, 3, 4, 6 and 7]. Both algorithms were executed five times and their average SSQs were calculated.

We determined 5 clusters for each algorithm in real dataset. For *MovStream*, the *MaxNumCluster* and *MinNumCluster* were both set to 5 so that it can output the same number of clusters as *CluStream*. For *CluStream* algorithm, the parameters were set at *micro-ratio* = 10, Max. Boundary Factor *t* = 2. Aggarwal et al. did extensive experiments and discussed that the clustering can achieve high-quality and stable in that situation [4]. *InitNumber* for both algorithms was set to 2000. For *MovStream* algorithm, the *ClusterTable* maintaining process was invoked every 100 data points, *checkinterval*=100. The *sensitivity parameter set* was set at {1.5, 1.5, 0.95, 45500, 0.1}. Fig. 6 gives the results when *horizon time* is set to 1000. At time stamp 30000, the average SSQ of *MovStream* is about equal to that of *CluStream*. Both algorithms output the average SSQs every 1000 data points. We have found that the average SSQs of two algorithms are almost in a same magnitude, but most of time *MovStream* is a little more accurate than *CluStream*. Fig. 7 shows the results when *horizon* is set to 5000 as Fig. 6. We compare the total average SSQs for all SSQs for each *horizon time* in data streams as showed in Fig. 8. *MovStream* is more accurate than *CluStream* about 25% to 50%.

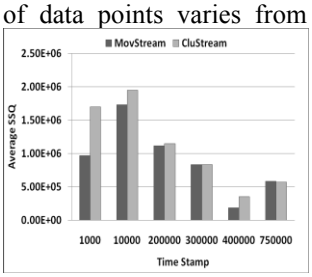

Fig.6 Network Intrusion dataset (*horizon time*=1000)

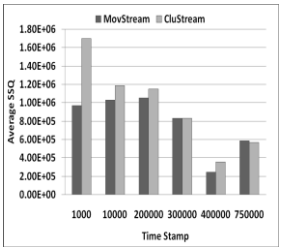

Fig.7 Network Intrusion dataset (*horizon time*=5000)

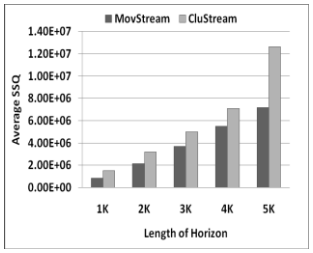

Fig.8 Network Intrusion dataset total Average SSQs

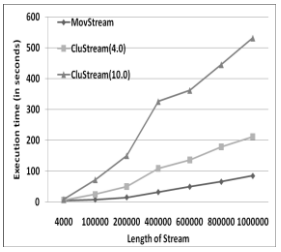

Fig.9 Network Intrusion dataset (*horizon time*=2000, *monitor-period*=2000)

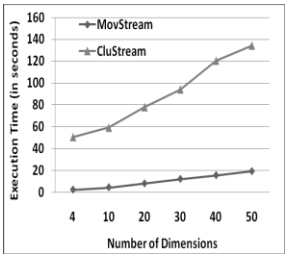

Fig.10 (B100C5 *horizon time*=2000, *monitor-period*=2000)

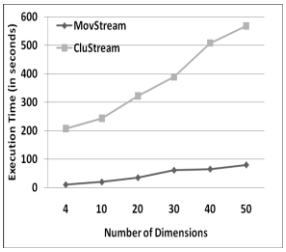

Fig.11 (B400C20 *horizon time*=5000, *monitor-period*=5000)

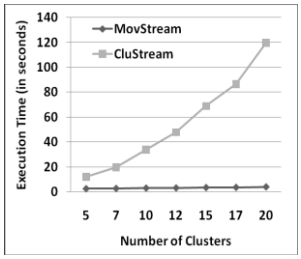

Fig.12 (B100D4 horizon time=2000, monitor-period=2000)

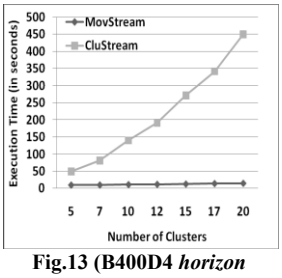

Fig.13 (B400D4 *horizon time*=2000, *monitor-period*=2000)

We also compare the efficiencies of both algorithms. The micro-clusters in *CluStream* don't represent the natural clusters in data streams, so when the purpose is monitoring natural clusters, *CluStream* needs to do macro-clustering for each period time. The number of data points streamed in the period time can be defined as a parameter *monitor-period*. Another important parameter for *CluStream* is *micro-ratio*, which is the number of micro-clusters divided by the number of natural clusters [4]. As Aggarwal et al. discussed in [4], large amount of micro-clusters leads to high-quality clustering result but low-performance. We compared the efficiency of *CluStream* with different *micro-ratio* as showed in Fig. 9. The size of load buffer of all algorithms is set to 100000 (points), and the execution time contains the time of data reading from hard disk. The parameters are set as *horizon time*=2000, *monitor-period*=2000. Other parameters are same as before. For example, *CluStream*(4.0) indicates the *micro-ratio* is 4.0 that the number of micro-clusters is equal to the number of natural clusters. From Fig. 9, we can see large amount of micro-clusters leads the performance to fall down rapidly. Considering the same clustering quality, *CluStream*(10.0) is slower about 10 times than *MovStream*. From the reports from Code Performance Profiler, we found that there are two main reasons for *MovStream* algorithm is s faster than *CluStream*. First, the number of clusters maintained in *CluStream* is larger than that in *MovStream*. *CluStream* need a large number of micro-clusters to achieve high-quality clustering, the cost of find and merge the suitable micro-clusters is high. Second, *CluStream* need *k*-means algorithm [1] to cluster the micro-clusters into macro-clusters in period time. The *k*-means algorithm is not efficient as many single-pass data clustering algorithms [2, 3 and 7]. Furthermore, it has to be executed in period time.

Fig. 10 and Fig. 11 show the scalability on the number dimensions test on synthetic datasets. Fig.12 and Fig.13 show the scalability on the number of clusters to determine on synthetic datasets. The *micro-cluster* is set as 4.0. In these synthetic dataset tests, we can see *MovStream* has a better linear scalability than *CluStream*, and *MovStream* is also faster than *CluStream*. Because *MovStream* fits the datasets perfectly, there is no *split movement* or *merge movement* could be triggered in whole processes. Thus the efficiency of *MovStream* is always very high.

6. Conclusions and Future Work

In this paper, we have studied: (a) introducing basic movements of the cluster with the measurement for them, (b) proposing an effective and efficient cluster monitoring method named *MovStream* based on the measurements, and (c) giving extensive experiments to show that *MovStream* can achieve more accurate by 25-50% than that of highest clustering quality of *CluStream*..*MovStream* includes basic movements of cluster and operations like balance keeping operations in B^+ Tree. More valuable types of movement of the cluster and more precise and efficient refinement operations will be proposed in the future.

7. References

1. R. Dua and P. Hart: Pattern Classification and Scene Analysis. J. wiley and Sons. Pages 10-45, 1973.
2. L. O'Callaghan, N. Mishra A. Meyerson and S. Guha: Streaming-Data Algorithms for High-Quality Clustering. In Proc. of ICDE, 2002
3. S. Guha, A. Meyerson, N. Mishra and R. Motwani.: Clustering Data Streams: Theory and Practice. IEEE Trans. on KDE, 15(3), pages 515-528, (2003)
4. C. C. Aggarwal, J. Han, J. Wang and P. S. Yu.: A Framework for Clustering Evolving Data Streams. In Proc. of the 29th VLDB, 2003
5. T. Zhang, R. Ramakrishnan, and M. Livny.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. ACM SIGMOD Conference, 1996
6. C. Gupta and R. Grossman.: GenIc: A Single Pass Generalized Incremental Algorithm for Clustering. In SIAM Conference on Data Mining 2004
7. S. Guha, N. Mishra, Motwani R and Challaghan LO': Clustering data stream. In Proc. of the 41st annual symposium on FOCS 2000, Redondo Beach, CA, Pages 359-366
8. M. M. Gaber, A. Zaslavsky and S. Krishnaswamy.: Mining Data Streams: A Review. SIGMOD Record, 32(2), pages 18-26, 2005
9. O. Nasraoui, C. Cardona, C. Rojas and F. Gonzalez.: TECNO-STREAMS: Tracking Evolving Clusters in Noisy Data Streams with a Scalable Immune System Learning Model. ICDM 2003, pages 235–242
10. O. Nasraoui and C. Rojas.: Robust Clustering for Tracking Noisy Evolving Data Streams. SDM, 2006.
11. C. Ordonez: Clustering binary data streams with k-means. In Proceedings of the 8th ACM SIGMOD workshop on research issues in DMKD 2003, pages 12–19.
12. F. Cao, M. Ester, W. Qian and A. Zhou: Density-Based Clustering over an Evolving Data Stream with Noise, SDM, 2006.
13. http://cs.scu.edu.cn/~tangliang/movstreampf.htm