ISBN: 978-981-18-7950-0
2023 the 13th International Workshop on Computer Science and Engineering (WCSE 2023)
doi: 10.18178/wcse.2023.06.007

A Partitioning and Distributed Caching Approach Based on Adaptive Spectral Clustering for Big Data Streams

Shun Wang [1, 2] and Guo-sun Zeng [1, 2 +]

[1] Department of Computer Science and Technology, Tongji University

[2] Tongji Branch, National Engineering & Technology Center of High Performance Computer, Shanghai, 201804, China

Abstract. Big data streams with diversity are generally processed by parallel computing environments with multiple computational nodes. Before processing, the big data streams need to be partitioned into sub-streams and cached on each computational node for subsequent processing. Existing partitioning methods are difficult to process streams with diversity and high-dimensional characteristics. Partitioning with low quality leads to unreasonable cache placing, which in turn leads to more data migration, lower computational efficiency, and smaller velocity of big data streams on a processing system. Inspired by the advantages of spectral clustering in identifying arbitrary manifolds, an approach of partitioning for big data streams based on spectral clustering is proposed, which transforms the partitioning of streams received during each micro window into the clustering of similarity graphs. We formulate an optimization problem for data items received during each micro window. Then, we present an algorithm to optimize the similarity graphs. With the characteristics of data streams changing gradually in adjacent windows, a distributed caching algorithm based on stream partitioning is presented for continuous windows. Experimental analysis shows that the proposed method can significantly improve the velocity and efficiency of the system for stream processing.

Keywords: big data stream, adaptively spectral clustering, distributed caching, similarity matrix

1. Introduction

In recent years, big data streams with heterogeneous data items have been emerging in many applications, such as social networks, e-commerce, and the Internet of Things. Usually, the instantaneous volumes of the streams are very big and require to be processed in real-time or near real-time. However, the contradiction between the reality for big volumes and the requirement of fast processing is very prominent. For example, social networks such as Twitter can generate thousands of messages every second. In public opinion analysis with Twitter, it is necessary to process the streams of social messages fast and respond as quickly as possible to master hot events or emergencies. The parallel computing environment is commonly adapted to speed up the processing of big data streams. Because of data diversity, big data streams should be partitioned and cached in distributed computing nodes in a parallel computing environment before processing. In this way, the parallelism of tasks in the computing environment can be fully developed, thus reducing the processing time.

Data diversity is a common feature of the data items in big data streams. Data items are usually generated in different scenarios from various data sources distributed in many places of the world. Besides, they are collected and transmitted to a data center by heterogeneous sensors or terminals. Then, diversified application scenarios and data sources are bound to generate big data streams with the feature of diversity. Data items with different features or classifications should be processed with different procedures or tasks. For example, messages generated by a social network include many topics. In the public opinion analysis application, messages of different topics are processed with different procedure tasks. For some Internet services, the requirements of different customers vary greatly; thus, their service levels can also be distinct. That is, the requests of a stream are processed with diverse service tasks. we should partition big data streams into sub-streams by classifying or clustering before they are cached on computational nodes.

[+] Corresponding author. Tel.: +8613681767358.
E-mail address: gszeng@tongji.edu.cn.

Inaccurate stream partitioning may bring many problems in processing big data streams on a parallel system. First, it may cause a waste of memory storage in the system. The data items of streams need to be cached in the memory of each computational node before processing. There have been many studies on caching streaming data on computational nodes before processing. For example, Blair [1] studied a data compression method using wavelet transform to reduce bandwidth requirements and improve end-to-end efficiency and latency for streaming data. An efficient partitioning can decrease the memory overhead of the distributed computational nodes. On the contrary, inefficient partitioning would increase the cache memory of each computational node and thus slow down the processing of big data streams. Secondly, inaccurate partitioning may lead to more communication overhead. We know that data items with different characteristics are often processed by specific tasks. Different tasks have different processing processes. If a certain type of data is too scattered, it may lead to more task interactions on different computing nodes and significantly increase the communication overhead. For example, Chen et al. [2] studied the processing of stateful heterogeneous tasks in parallel systems. The amounts of intermediate results produced during the execution of different types of tasks are different. Ideally, the same type of tasks should be executed on one computing node as much as possible. Unreasonable partitioning often leads to increased communication overhead in the processing process, which will delay the completion time. In addition, different types of tasks process different types of data with different computational resource costs, and low-quality partitioning may cause uneven load distribution among different computing nodes, which will lead to additional computational resource costs and reduce the processing efficiency of data streams. Therefore, it is necessary to partition the data stream with high quality to improve the efficiency of subsequent processing.

Existing clustering methods such as K-means, density-based methods, and their variants like StreamKM++ [3] and DenStream [4] are widely used for stream partitioning of high-dimensional. However, the accuracy of these methods is low, especially for streams with non-convex manifolds. To overcome these disadvantages, we study an adaptive stream partitioning and distributed caching approach based on spectral clustering for big data streams.

2. Related Work

Streams partitioning refers to dividing data streams into multiple sub-streams by classification or clustering. Major previous studies focused on this problem can be categorized into following classes.

2.1. Partition Based on Traditional Clustering

Simple partitioning is based on a few features of data streams, such as quantities or attributes. They always have simple implementation and low time complexity. For example, shuffle grouping is a simple partitioning method that evenly divides data items of a stream into sub-streams in a round-robin manner [2]. It is applied to many distributed systems for processing data streams, such as Storm and Flink. Field grouping is another simple partitioning method based on attribute values for streams with relational data items [5]. It divides a stream into sub-streams by putting data items together whose value of one or several specific attributes are the same. This group of methods usually process streams with simple data structure and light processing tasks.

Hierarchical clustering methods are based on the binary tree structure. They include agglomerative and divisive clustering methods. Agglomerative clustering regards each data item as a cluster block at the beginning and continuously agglomerates these small clusters into larger ones in repeated iterations. Divisive clustering starts by assuming a single cluster contains whole data and dividing the clusters into smaller ones in each step. Typical hierarchical clustering methods are ODAC [6], HUE-Stream [7], HASTREAM [8]. Hierarchical clustering methods are usually simple and do not need to set the number of clusters in advance. Besides, they also provide users with more convenient and intuitive clustering results. The disadvantages are that they often have high computational complexity and are very sensitive to outliers.

Block clustering divides a data set into a predetermined number of clusters based on similarity to the cluster centroids. The most common one is the K-means proposed by MacQueen [9]. There are many other variants based on K-means for stream partitioning, including StreamKM++ [3], incremental k-means [10], CluStream [11], etc. The advantage is that they have an easy implementation in general. But only hyper-spherical clusters can be determined with these methods.

Density-based clustering is to divide the data set into a large number of micro-clusters. Each micro-cluster is composed of a group of data that is very close to each other. Each micro-cluster stores a summary in the form of a vector. These micro-clusters are combined to form the final cluster according to the density accessibility and connectivity theory. The representative clustering method is DenStream, proposed by Cao et al. [4], which introduces the concepts of core micro-clusters and outlier micro-clusters and gives an outlier micro-clusters recognition strategy based on pruning thought, which integrates the reachable core micro-clusters to achieve clustering. However, the disadvantages of this method are high computational complexity and difficulty in dealing with overlapping clusters.

2.2. Partitioning Based on Spectral Clustering

Partitioning based on spectral clustering takes data items as points, takes the similarity between points as the weight of edges to generate undirected graphs, and converts stream partitioning into clustering based on the undirected graphs. Spectral clustering methods can be classified into the minimum cut, average cut, ratio cut, etc., by their cut rules [12]. By the construction of similarity graphs, they can be divided into single-view spectral clustering and multi-view spectral clustering [15] [16]. By way of graph partitioning, they can be divided into 2-way clustering [13] [14] and k-way clustering [17] [18]. 2-way spectral clustering divides the graph into two parts based on the relationship between the second largest eigenvalue of the Laplace matrix of the graph and then recursively applies the same procedure to the sub-graphs in a hierarchical way until the number of clusters is enough or the recursive conditions are violated [14]. This group of methods is unsuitable for stream clustering because of their large computation and low efficiency. k-way spectral clustering first selects several main eigenvectors of the Laplace matrix that contain classification information in a heuristic way and then uses these eigenvectors to map the original data points to a spectral space. This paper focuses on k-way spectral clustering for stream partitioning.

Ng et al. [17] was the first to propose k-way spectral clustering. They use k eigenvectors of the Laplace matrix to partition a graph into k clusters, which is faster 2-way spectral clustering. Thereafter, researchers made a lot of improvements based on Ng's work. For example, Li et al. [18] proposed a semi-supervised learning spectral clustering algorithm. Its basic idea is to set prior information constraints based on a small number of known label data so as to optimize the similarity matrix and improve the accuracy of clustering. Chen and Feng [19] present a novel k-way spectral clustering algorithm called discriminant cut (Dcut). However, these methods rely on constructing accurate similarity graphs. Nie et al. [20] [21] proposed an adaptive spectral clustering method. It constructs an optimization problem to explore the minimum distance among neighbor vertices of the similarity graph. By ingenious mathematical transformation, an algorithm of adaptive spectral clustering was presented to optimize the similarity matrix. It considers both processing speed and accuracy. Thus, we introduce this method into data stream clustering, taking its advantages in high-dimensional data clustering to improve the partitioning of big data streams.

Spectral clustering has higher accuracy than traditional methods for clustering high-dimensional data items because it can identify non-convex manifolds. At present, a few studies have already applied spectral clustering for data stream clustering. However, these studies directly use the distance between data to construct the similarity matrix without considering the effect of data locality. Besides, it is difficult to identify clusters of nonlinear or non-convex manifolds accurately. To eliminate data locality while ensuring processing efficiency, we attempt to adopt adaptive spectral clustering to partition big data streams before processing.

3. Principle of Partitioning Based on Adaptive Spectral Clustering

3.1. Assumptions of the Big Data Stream

Assume the big data stream we study is a sequence of continuously arriving, high-dimensional data items, denoted as $S=\{d_1, d_2, \dots, d_n, \dots\}$. Any data item d_i in S contains m attributes, denoted as $d_i(a_1, a_2, \dots, a_m)$. The attributes of data items in S are determined by the specific application background in practical application. These data items continuously enter a data center and are processed.

Denote every N data items received as a data set named a window. Without losing generality, stream S is processed by a window model. That is, the data items are processed window by window separately with their arrival. The size of the window has a great impact on the processing speed. We set a small window size of N

to ensure processing speed. The data set received in the *t*th window is S_t={d_1^t, d_2^t, …, d_N^t}. The data stream collected during the life cycle is S=S_1∪S_2∪…∪S_t∪…. The data items in *S* can be clustered into several groups. Assumed that data items in different clusters are processed by a specific task, and each task is independent of others. Usually, data items are placed on multiple computing nodes in a data center. Thus, the data stream needs to be partitioned into sub-streams before processing. This paper proposes an approach based on adaptive spectral clustering for stream partitioning and distributed caching in a data center.

3.2. Principle of Adaptive Spectral Clustering

Assume that the data received by the *t*h window is S_t={d_1^t, d_2^t, …, d_N^t}∈$\boldsymbol{R}^{N\times m}$, where *m* is the number of attributes contained in the data items, *N* is the number of data items set for a window, and the value of *N* is preset by experience. Following the adaptive spectral clustering [20] [21] [22], the first step is to construct a similarity graph with data items of S_t. Taking data items in S_t as vertices, the similarity between data items as the weight of edges, and then the similarity graph G^t corresponding to S_t can be constructed. Denote the adjacency matrix formed by the similarity between all data items in the similarity graph G^t as W^t. How to calculate the similarity between data items, that is, the weight of the edges in the graph, is the key to constructing the similarity graph. The most common way to set the similarity between data items is the Gaussian kernel [24]. Specifically, the similarity between data items $d_i^t$and d_j^t is expressed as:

$$w_{i,j}^t = exp(\frac{||d_i^t-d_j^t||_2^2}{2\sigma^2}) \quad (1)$$

where σ is the adjusting parameter, $w_{i,j}^t$∈[0,1]. Generally, a small pre-set threshold ω0 is needed in this method. That is, if $w_{i,j}^t$<ω0, there is no connection between $d_i^t$and d_j^t; if $w_{i,j}^t$>ω0 the weight between $d_i^t$and d_j^t is set to be $w_{i,j}^t$. The similarity graph is constructed, which is G^t= (St, W^t). Then, the clustering of S_t is transformed into the partitioning of similarity graph G^t.

Assume D is the degree matrix of G^t, D∈ $R^{N\times N}$. Let the degree of d_i^t be *deg*$_i$ and *deg*$_i$ =$\sum_j^N(w_{i,j}^t)$, then the Laplace matrix of G^t is L=D-W^t. Let *I* be the identity matrix with the same dimension as *L*. It can be standardized as Formula (2).

$$L=D^{-\frac{1}{2}}(D-W^t)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}W^tD^{-\frac{1}{2}} \quad (2)$$

Assume graph G^t needs to be partitioned into k subgraphs which are $G_1^t, G_2^t,\dots G_k^t$, $G^t = G_1^t \cup G_2^t \cup\dots G_k^t$. For any two subgraphs G_i^t and G_j^t, $G_i^t \cap G_j^t = \emptyset$. We utilize normalized cut in the adaptive spectral clustering. This cut criterion comprehensively considers the total weight of the edges inside the subgraph after clustering and segmentation, as well as the weight of the cut edges between subgraphs. It reflects the similarity of the vertices in the subgraph and measures the difference of the data between subgraphs, which can be described as Formula (3).

$W(G_i^t,\overline{G_i^t})$ in Formula (3) represents the sum weights of cut edges between subgraphs, and $vol(G_i^t)$ represents the total weight of the internal edges of G_i^t. Let F∈ $R^{N\times k}$ be an indicator matrix, each row mapping a subgraph. According to [25], the partitioning objective which meets Formula (3) can be standardized as Formula (4).

$$\begin{cases} \min\limits_F Tr(F^TLF) \\ \quad F^TF = I \end{cases} \quad (4)$$

To optimize problem (4), we utilize eigenvectors of the Laplace matrix to solve this problem. Problem (4) obtains its optimal solution when *F* is a matrix consisting of *k* minimum eigenvectors of *L*. That is, the partitioning of S_t can be transformed into optimizing graph G^t using eigenvectors of the Laplace matrix *L*.

Generally, spectral clustering sets the similarity between data items with the Gaussian kernel and judges whether there is an edge by a similarity threshold. However, constructing a similarity graph with this method has poor accuracy because it ignores the locality of the data. The obtained graph can only reflect the similarity between the two points without considering the mutual influence of their neighbor points. The edges between vertices are not only determined by the distance between data items but also by the surrounding vertices, especially in the case of many non-hyperspherical clustering manifolds. Partitioning a stream is to divide the data items into different parts and continuously cache them to computing nodes. For S_t, it should be divided into *k* clusters, which can be denoted as S_t= $C_1^t \cup C_2^t \dots\cup C_k^t$. Then the entire stream partitioning can be denoted as S={$C_1^1 \cup C_2^1 \ldots \cup C_k^1\}\cup\{C_1^2 \cup C_2^2 \ldots \cup C_k^2\} \ldots \{C_1^t \cup C_2^t \ldots \cup C_k^t\}$…. This partitioning process based on spectral clustering includes two steps: (1) partitioning the data set arriving at each window to obtain *k* clusters. (2) Continuous placing the clusters of each window on q computing nodes respectively. We will discuss an adaptive similarity matrix and a continuous caching strategy with the progressive change of the data stream.

4. Optimization of Adaptive Similarity Matrix

Clustering a sub-stream St arrived in a window includes three steps. (1) Construct the similarity graph G^t. (2) Calculate the eigenvectors matrix of G^t and its Laplace matrix. (3) Cluster with the Laplace matrix and eigenvectors of G^t. To eliminate the data locality, we formulate a problem for the optimal similarity matrix and present a self-learning process with the idea of minimum weights of all neighbors.

4.1. Optimization in a Single-Window

Assume the optimal similarity graph that has eliminated the data locality is G^t. If edges are connected between two vertices in graph G^t, the two vertices are called neighbors. We regard the weights of edges between vertices in $G^{t\prime}$ which has not been optimized, as the probability of being neighbors. Intuitively, the smaller the distance between two vertices, the more likely they are to be neighbors. Thus, the basic idea to optimize the similarity matrix is to obtain the neighbor probability. When the sum of weighted distances between all vertices is the minimum, the corresponding neighbor probability is optimal[20] [21] [22][23].

For S_t={d_1^t, d_2^t, …, $d_{N_t}^t$}, its similarity matrix is denoted as W^t, and W^t ∈N×N. Let $w_{i,j}^t$ be the *j*th element in row *i* of W^t, representing the similarity between d_i^t and d_j^t. Denote $\boldsymbol{w}_i^t$ as the vector composed of elements in the *i*th row. The weighted distance between d_i^t and d_j^t can be expressed as $\left\|d_i^t - d_j^t\right\|_2^2 w_{i,j}^t$. By [20] [21], the similarity graph G^t should meet the following objective.

$$\min_{w_i^T\mathbf{1}=1,0\le w_{i,j}\le 1} \sum_{i,j=1}^{N}(\left\|d_i^t - d_j^t\right\|_2^2 w_{i,j}^t + \varepsilon(w_{i,j}^t)^2) \tag{5}$$

In Formula (5), ε is a regular parameter. To solve *w* in Formula (5), we can obtain the optimal similarity matrix.

4.2. Optimization of Similarity Matrix

Formula (5) is difficult to be solved directly. We need to do some transformation for it. According to [26], we can get the following property. For G^t and its Laplace matrix L, the number of 0 among its eigenvalues equals the number of G^t's subgraphs. Thus, when the number of 0 in its eigenvalues is *k* will meet the partitioning requirements. Denote rank (L) as the rank of L. Adding the constraint rank (L)=N-k to the Formula (5), and we can convert it into Formula (6).

$$\begin{cases} \min\limits_{W^t} \sum_{i,j=1}^{N}(\left\|d_i^t - d_j^t\right\|_2^2 w_{i,j}^t + \varepsilon(w_{i,j}^t)^2) \\ (\boldsymbol{w}_i^t)^{\boldsymbol{T}}\mathbf{1} = 1, 0 \le w_{i,j}^t \le 1, rank(L) = N - k \end{cases} \tag{6}$$

The solution of Formula (6) is to explore the W^t when the sum of the k minimum eigenvalues of L approaches 0. The constraint rank (L)=N-k in Formula (6) still can not participate in the calculation directly. So, further transformation is needed. Let L^* be the optimized Laplace matrix, $\lambda_i(L^*)$ be the *i*th smallest eigenvalue of L^*, $\boldsymbol{f}_i^*$ be the eigenvector corresponding to $\lambda_i(L^*)$, F^* be the matrix composed of eigenvectors corresponding to the k ascending eigenvalues of L^*. L^* is positive semi-definite and $\lambda_i(L^*)$≥0. According to Ky Fan's research [27], we have that $\sum_{i=1}^{k}\lambda_i(L^*) = \min\limits_{F^*} Tr(F^{*T}L^*F^*)$, $F^{*T}F^* = I$, where I is a unit matrix. For a large enough parameter β, when $\sum_{i=1}^{k}\lambda_i(L^*)$ approaches 0, the W^t approaches optimal. Thus, Formula (6) can be converted into Formula (7).

$$\begin{cases} \min\limits_{W^t} \sum_{i,j=1}^{N}(\left\|d_i^t - d_j^t\right\|_2^2 w_{i,j}^t + \varepsilon(w_{i,j}^t)^2) + \beta Tr(F^{*T}L^*F^*) \\ (\boldsymbol{w}_i^t)^{\boldsymbol{T}}\mathbf{1} = \mathbf{1}, 0 \le w_{i,j}^t \le 1 \end{cases} \tag{7}$$

The value range of regularization parameters ε is (0,+∞), which is difficult to obtain. Thus, Formula (7) still needs more transformation. Let $e_{i,j}^d = \left\|d_i^t - d_j^t\right\|_2^2$, $e_{i,j}^{f^*} = \left\|\boldsymbol{f}_i^* - \boldsymbol{f}_j^*\right\|_2^2$, where $\boldsymbol{f}_i^* \in R^{1\times k}$ is the eigenvector corresponding to *i*th smallest eigenvalue of matrix F^*. Let $\boldsymbol{e}_i \in R^{N\times 1}$ be a vector composed of N

elements, whose jth element is $e_{i,j} = e^{d}_{i,j} + \beta e^{f^*}_{i,j}$. Denote $\boldsymbol{w}^t_i \in R^{n\times 1}$ as the vector composed of a row in W^t. Then, Formula (7) can be transformed into the following form.

$$\min_{(\boldsymbol{w}^t_i)^T\mathbf{1}=1, 0\le w^t_{i,j}\le 1} \left\| \boldsymbol{w}^t_i + \frac{1}{2\varepsilon}\boldsymbol{e}_i \right\|^2_2 \tag{8}$$

With KKT [28] and Formula (8), we can obtain the following expression.

$$w^t_{i,j} = -\frac{1}{2\varepsilon_i} e^d_{i,j} + \eta_i \tag{9}$$

In Formula (9), ε_i and η_i are the adjustment parameters. To reduce the computational complexity and eliminate data locality, only b nearest neighbors of d^t_i are used to calculate the weights. Then, we can obtain $\varepsilon_i = \frac{b}{2} e^d_{i,b+1} - \frac{1}{2}\sum_{j=1}^{b} e^d_{i,b}$, $\eta_i = \frac{1}{b} + \frac{1}{2b\varepsilon_i}\sum_{j=1}^{b} e^d_{i,j}$. Since is the average of $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$, we have

$$\varepsilon = 1/n \sum_{i=1}^{n} \left(\frac{b}{2} e^d_{i,b+1} - \frac{1}{2}\sum_{j=1}^{b} e^d_{i,b}\right) \tag{10}$$

Parameter b is a quantity of practical, which is easy to obtain by training. So, parameter ε can be calculated with b. By the above analysis, we design the algorithm CSM (Calculate Similarity Matrix) to obtain the similarity matrix W^t. The pseudocode is as follows.

```
Algorithm 1: CSM
Input: $S_t=\{d^t_1, d^t_2, \ldots, d^t_{N_t}\}, k, \varepsilon, \beta, \theta_0$; // $\theta_0$ is convergence threshold
Output: $W^t \in R^{n\times n}$;            //similarity matrix
1:  CSM ($S_t, k, \varepsilon, \eta, \beta$)//
2:  {  $W^t=(w^t_{i,j})_{i,j=1,2\ldots N}$←random(1);        // initialize with random values in (0,1)
3:      $L$←init_Laplacian($W^t$);                  //calculating the Laplace matrix
4:      $\theta$←∞;
5:      while $\theta > \theta_0$
6:      {  // $k$ minimum eigenvector
           $F$←get_least_k_eigenvectors($L$);
7:         $w^t_{i,j} \leftarrow -\frac{1}{2\varepsilon_i} e^d_{i,j} + \eta_i$;                   //by Formula(9)
8:         $W^t$ ← optimiz_similarity_matrix($W^t$);
9:         $L$← optimiz_Laplacian_matrix ($W^t$);
10:         $\theta \leftarrow \sum_{i=1}^{k} \lambda_i (L)$;
12:     }
13:     return $W^t$;
14:  }
```

In Algorithm 1, parameter k is known in advance, ε is trained by Formula (10), η can be calculated by ε, and β is a large enough value. With the above parameters, the similarity matrix W^t can be obtained by running Algorithm 1.

5. Stream Partitioning and Distributed Caching

5.1. Partitioning with Adaptive Spectral Clustering

With Algorithm 1, we get the similarity matrix of the sub-stream received during a single window. To partition the entire data stream, it is necessary to continuously partition the data items that arrive at a series of windows. The key to calculating the similarity matrices is to estimate the parameters ε of each window. We observed that a smooth transition exists in adjacent windows. Thus, exponential smoothing is used to estimate the parameters of adjacent windows through historical parameters ε. With the above ideas, we design the partitioning algorithm ASCS (Adaptive Spectral Clustering for Stream). The pseudocode is as follows.

```
Algorithm 2:  ASCS
Input: $S=\{S_1, S_2, \ldots\}, k, \varepsilon^0, \eta^0\ \beta, \theta_0$; //
Output: $C=\{C^1 \cup C^2 \ldots \cup C^t \ldots\}$;  //$C$ is the cluster set of $S$, $C^t=\{C^t_1, C^t_2, \ldots, C^t_k\}$
1:  ASCS($S, k, \varepsilon^0, \eta^0, \beta, \theta_0$)
2:  {  $\varepsilon^0, \eta^0$←init(), continue_flag←1; // continue_flag indicates whether the stream ends
3:      while continue_flag=1
4:    { $S_t$←receive_current_window($S$);
5:        while $S \neq \emptyset$
6:        {  $t$=get_window_No($S_t$);
7:            if $t$ >1
               ////parameters estimating
8:              $(\varepsilon^{t+1}, \eta^t)$←neighbor_window_estimate($\varepsilon^t$, $\eta^t$);
9:            $W^t$←CSM($S$, $k$,$\varepsilon^t$,$\eta^t$, $\beta$);   //by Algorithm 1
10:           $D$←($deg_i$ ←$\sum_{j=1}^{n} w_{i,j}^t$)$_{i=1,2,\ldots,N}$;  //Degree matrix
11:           $L$← $I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$;          // standard Laplace matrix
12:           $F$← get_least_k_eigenvectors($L$);
13:           $M$← standardize($F$);            //
14:           $C^t$={$C_1^t$, $C_2^t$, …, $C_k^t$}←get_k_clusters($S_t$, $W^t$, $M$); //clustering
15:           $C$← $C \cup C^t$;
16:       }
17:       continue_flag←0;
18:   }
19:   return $C$;
20: }
```

5.2. Distributed Caching of the Big Data Stream

The data items in the same cluster often have the same characteristics and are processed by the same tasks. Denote *vol*(C_i) as the computational resources when cluster C_i is processed. We place all clusters of each window on computing nodes according to the idea of "Min-Min" [29]. The stream partitioning and distributed caching algorithm PASC (Partitioning based on Adaptive Spectral Clustering) is based on Algorithm 2. The pseudocode is as follows.

Algorithm 3: PASC

Input: P={p_1, p_2, …,p_q}, S={S_1, S_2, …}, k, ε^0, η^0 β, θ_0; // P is the set of computing nodes, C^t is the cluster set of S_t, C_j^t is the jth cluster in C^t

Output: D_1, D_2, …, D_q;   //D_i is the data set to be cached on p_i

```
1:  PASC()
2:  {  $C$ ←ASCS ($S$, $k$, $\varepsilon^0$, $\eta^0$, $\beta$, $\theta_0$);        //call Algorithm 2
3:     $C^t$ ←get_current_clusters($C$);        //clusters of the current window
4:     while $C^t \neq \emptyset$
5:     {  $C_j^t$ ← get_min_vol_cluster($C^t$);    //
6:        $P_c$ ← find_nodes_with_min_cache($C_j^t$); //
7:        if |$P_c$|=1
8:        {   $p_i$ ← get_a node($P_c$);
9:            $D_i$ ←  $C_j^t$ ∪ get_existing_cache_data($p_i$);
10:       }
11:       else  if |$P_c$|>1
12:       {  // node with minimum weight between $C_j^t$ and other clusters
$p_i$ ← get_node_with_min_inter_vol($P_c$, $C_j^t$);
13:           $D_i$ ←  $C_j^t$ ∪ get_existing_cache_data($p_i$);
14:       }
15:       Remove($C^t$, $C_j^t$);
16:    }
17:    return $D_1$, $D_2$, …, $D_q$;
18: }
```

6. Experiments

6.1. Experimental Setup

Computing environment: The experiments were conducted on a computing cluster composed of four PCs, in which the CPU of each PC is Intel i5-11500H@2.50 GHz, 16 GB memory, 1T main hard disk, and these machines are connected through a high-speed local area network. All machines are installed with CentOS 6.5 (64-bit) operating system. The programs of the proposed algorithm are implemented on Apache Storm. The above software and hardware environment constitute a heterogeneous data center for our experiments.

Data sets: Three data sets were used in the experiments , which are a public data set named SHAPE [30], denoted as D1; a news data set of different topics collected on the Sina website by using crawler tools, denoted as D2; a collaborative filtering data set in [31], denoted as D3. Among them, data items of D1 are read at random time intervals to simulate the data streams that arrive randomly. D2 and D3 are read data at the interval of the time stamp of the data items generated.

6.2. Experiment and Analysis

The representative cluster-based partitioning methods for data streams are the block-based clustering method SKKM [3], the density-based clustering method DSC [4], and the hierarchical clustering method ODAC [6]. The experiments compare the proposed method PASC with the existing methods mentioned above in terms of clustering accuracy and computing resource utilization of subsequent processing.

Experiment 1: Parameter prediction of exponential smoothing

This experiment was designed to evaluate the accuracy of parameter prediction with the smooth exponential method in the continuous windows. In this experiment, we compare the parameters ε^t of some windows predicted by smoothing with those that are trained by the method explained by Formula (10). The PASC algorithm was run on simulated data streams with data sets D1, D2, and D3, respectively. The experimental results are shown in Figure 1. The error of parameter ε^t between the predicted value and the trained value is less than 0.1 in most windows. Thus, it can support the parameter setting in continuous windows.

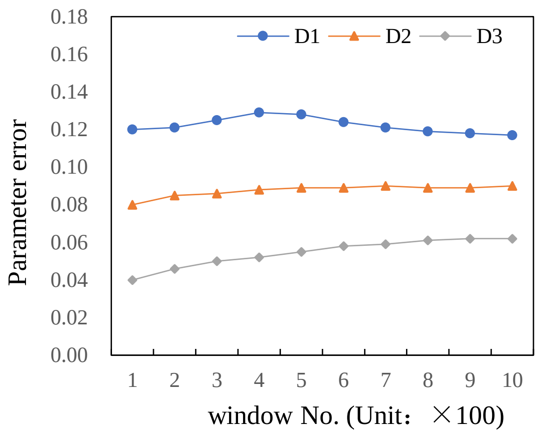


Fig. 1 Parameter prediction for the smooth exponential

Experiment 2 :

Following [32], the accuracy of clustering is the ratio between the number of correct clusters and the clusters, which can be expressed by the following Formula.

$$ACC=\frac{\sum_{i=1}^{n}\delta(y_i,map(c_i))}{n} \tag{11}$$

where y_i is the real cluster label of data item d_i, and c_i is the result obtained by our clustering algorithm. Function $\delta(x, y) = 1$ when $x=y$, otherwise 0. This experiment was also conducted with data sets D1, D2, and D3 to simulate the data streams. The algorithms of SKKM, DSC, ODAC, and PASC run on these streams, respectively. The results are shown in Fig. 2. The four groups of histograms show the clustering accuracy of these algorithms. On the whole, the accuracy of PASC on D1, D2, and D3 data sets is more than 90%, while that of SKKM, DSC, and ODAC is less than 85%, which is significantly lower than that of PASC. This shows that the PASC is better than other methods in terms of partitioning accuracy.

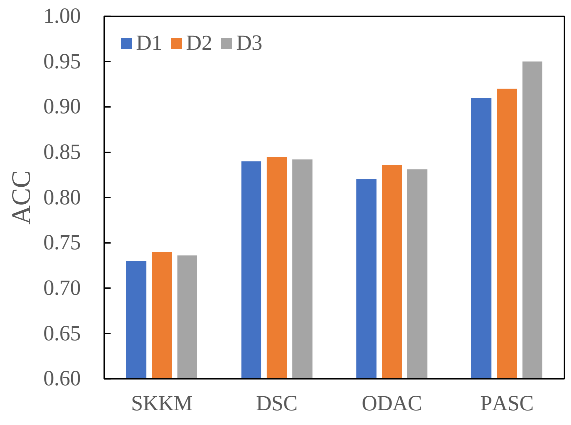
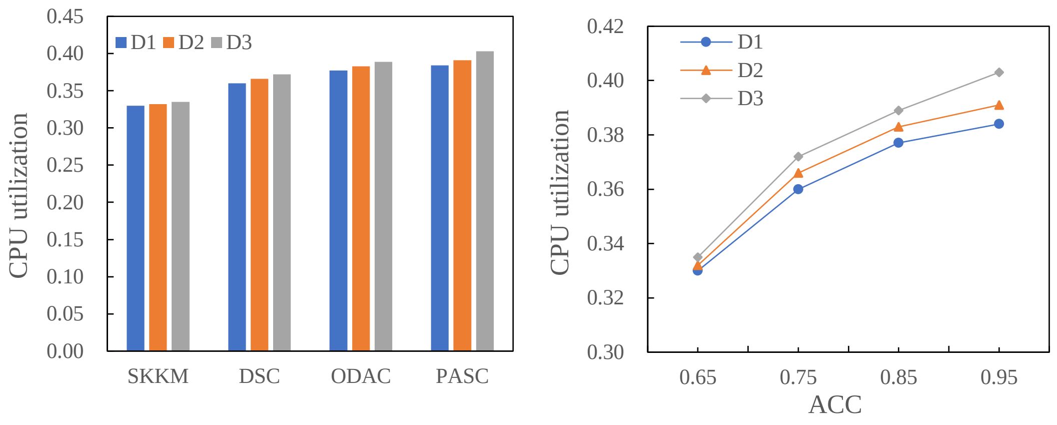


Fig. 2 Comparison of partitioning accuracy

Experiment 3: Average utilization of computational resources

In this experiment, SKKM, DSC, ODAC, and PASC also run on streams simulated with Data sets D1, D2, and D3. After being processed by these algorithms, the streams are put forward to execute some subsequent tasks. We collect the average CPU utilization when processing the subsequent tasks. Several aggregation tasks are executed on stream D1. Stream simulated by D2 runs the tasks of public opinion analysis. Tasks of collaborative filtering run on stream by D3. The experimental results are shown in Fig. 3. Fig. 3 (a) shows that the utilization of computational resources processed by PASC is higher than the other three methods. In Fig. 3(a), DSC and ODAC are nearly the same in the utilization of computational resources because their accuracies of partitioning are close. Fig. 3 (b) shows that the accuracy of partitioning is positively correlated with CPU utilization, which means that our partitioning methods can improve the utilization of computational resources.



(a)Computing resource utilization (b) Relationship between ACC and resource utilization

Fig. 3 Comparison of resource utilization

7. Conclusions

Partitioning and caching are important works before the processing of big data streams. Thus, this paper proposed a partitioning and caching approach based on adaptive spectral clustering. It is a clustering and optimal placing problem for big data streams on a series of windows. We modelled the data items received in each window with a similarity graph, formulated a problem of optimizing the similarity matrix, and designed an algorithm to generate the similarity matrix adaptively. Then, a partitioning and caching method based on the similarity matrix is proposed. Finally, a series of comparative experiments with SKKM, DSC, and ODAC are conducted to verify the proposed method. Experimental results show that the clustering accuracy and computing resource utilization are better than the existing methods, indicating that the proposed method can effectively improve the clustering accuracy and thus improve the processing performance of big data streams.

8. Acknowledgements

This work was supported by the National Natural Science Foundation of China under grant No. 62072337; the National Key R&D Program of China under grant No. 2019YFB1704100; the National Social Science Foundation of China under grant No. 17BTQ086; the Subproject of National Seafloor Observatory System of China under grant No. 2970000001/001/016.

9. References

[1] S. Blair, J. Costello. Slipstream: High-performance lossless compression for streaming synchronized waveform monitoring data. International Conference on Smart Grid Synchronized Measurements and Analytics, 2022, pp. 1-6.

[2] Chen, F. Zhang, H. Jin. Pstream: a popularity-aware differentiated distributed stream processing system. IEEE Transactions on Computers, 2020, 70(10): 1582-1597.

[3] M. R. Ackermann , M. Mrtens , C. Raupach, et al. StreamKM++: a clustering algorithm for data streams. Proceedings of the Workshop on Algorithm Engineering and Experiments, 2012, pp. 173-187.

[4] F. Cao, M. Estert, W. Qian, et al. Density-based clustering over an evolving data stream with noise. Proceedings of the 2006 SIAM International Conference on Data Mining, 2006, pp. 328-339.

[5] J. Xu, Z. Chen, J. Tang, et al. T-storm: traffic-aware online scheduling in storm. IEEE 34th International Conference on Distributed Computing Systems, 2014, pp. 535-544.

[6] P. P. Rodrigues, J. Gama, J. P. Pedroso. ODAC: hierarchical clustering of time series data streams. Proceedings of the 2006 SIAM International Conference on Data Mining, 2006, pp. 499-503.

[7] W. Meesuksabai, T. Kangkachit, K. Waiyamai. Hue-stream: evolution-based clustering technique for heterogeneous data streams with uncertainty, 2011, pp. 27–40.

[8] M. Hassani, P. Spaus, T. Seidl. Adaptive multiple-resolution stream clustering. In: Machine learning and data mining in pattern recognition, 2014, pp. 134–148.

[9] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. 5th Symposium on Mathematical Statistics and Probability,1967, pp. 281-297.

[10] C. Ordonez. Clustering binary data streams with k-means. Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, 2003, pp. 12-19.

[11] C. C. Aggarwal, S. Y. Philip, J. Han, et al. A framework for clustering evolving data streams. Proceedings 2003 VLDB conference, 2003, pp. 81-92.

[12] D. Hamad, P. Biela. Introduction to spectral clustering. 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, 2008, pp. 1-6.

[13] S. Barnard, H. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 711-718.

[14] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. Czechoslovak Mathematical Journal, 1975, 25(4): 619-633.

[15] S. F. Hussain, M. Mushtaq, Z. Halim. Multi-view document clustering via ensemble method. Journal of Intelligent Information Systems, 2014, 43(1): 81-99.

[16] H. Wang, Y. Yang, B. Liu. GMC: Graph-based multi-view clustering. IEEE Transactions on Knowledge and Data Engineering, 2019, 32(6): 1116-1129.

[17] A. Ng, M. Jordan, Y. Weiss. On spectral clustering: Analysis and an algorithm. Advances in Neural Information Processing Systems, 2001,14:849-856.

[18] J. Li, J. Wei, M. Ye, et al. Privacy-preserving constrained spectral clustering algorithm for large-scale data sets. IET Information Security, 2020, 14(3): 321-331.

[19] Chen W, Feng G. Spectral clustering with discriminant cuts. Knowledge-Based Systems, 2012, 28(1): 27-37.

[20] Q. Wang, Z. Qin, F. Nie, et al. Spectral embedded adaptive neighbors clustering. IEEE Transactions on Neural Networks and Learning Systems, 2018, 30(4): 1265-1271.

[21] F. Nie, X. Wang, H. Huang. Clustering and projected clustering with adaptive neighbors. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014: 977-986.

[22] X. Zhu, J. Gan, G. Lu, et al. Spectral clustering via half-quadratic optimization. World Wide Web, 2020, 23: 1969-1988.

[23] H. U. Qiankun, D. Shifei. p-Spectral clustering algorithm with optimization of local similarity. Journal of Frontiers of Computer Science & Technology, 2018, 12(3): 462-471.

[24] U. V. Luxburg. A tutorial on spectral clustering. Statistics and Computing, 2007, 17(4): 395-416.

[25] S. X. Yu and J. Shi. Multiclass spectral clustering. IEEE International Conference on Computer Vision, 2003, pp. 313-313.

[26] M. Hein, J. Y. Audibert, U. V. Luxburg. Graph Laplacians and their convergence on random neighborhood graphs. Journal of Machine Learning Research, 2007, 8(9): 1325-1370.

[27] K. Fan. On a theorem of Weyl concerning eigenvalues of linear transformations I. Proceedings of the National Academy of Sciences, 1949, 35(11): 652-655.

[28] S. Boyd, L. Vandenberghe. Convex optimization. Cambridge: Cambridge University Press, 2004.

[29] K. Etminani, M. Naghibzadeh. A min-min max-min selective algorihtm for grid task scheduling. 2007 3rd IEEE/IFIP International Conference in Central Asia on Internet, 2007, pp. 1-7.

[30] P. Franti. (2015). Clustering Datasets. [Online]. Available: http://cs.uef.fi/sipu/datasets/

[31] S. Ubukata, S. Takahashi, A. Notsu, et al. Basic consideration of collaborative filtering based on rough c-means clustering. Joint 11th International Conference on Soft Computing and Intelligent Systems and 21st International Symposium on Advanced Intelligent Systems, 2020, pp. 1-6.

[32] D. Cheng, Q. Zhu, J. Huang, et al. A novel cluster validity index based on local cores. IEEE Transactions on Neural Networks and Learning Systems, 2018, 30(4): 985-999.