We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

10

Clustering Parallel Data Streams

Yixin Chen
Department of Computer Science,
Washington University
St. Louis, Missouri

1. Introduction

Massive volumes of data streams can be found in numerous applications such as network intrusion detection, financial transaction flows, telephone call records, sensor streams, and meteorological data. In recent years, there are increasing demands for mining data streams. Unlike the finite, statically stored data sets, stream data are massive, continuous, temporally ordered, dynamically changing, and potentially infinite [5]. For example, Cortes et al. report that AT&T long distance call records consist of 300 million records per day for 100 million customers. For the stream data applications, the volume of data is usually too huge to be stored or to be scanned for more than once. Further, in data streams, the data points can only be sequentially accessed. Random access to data is not allowed.

Extensive research has been done for mining data streams, including those on the stream data classification [3, 20], mining frequent patterns [9, 17, 18], and clustering stream data [1, 2, 8, 9, 10, 11, 12, 13, 14, 16, 19].

In this paper, we study the clustering of multiple and parallel data streams. Our study should be differentiated from some previous studies on clustering stream data [19, 1]. Our goal is to group multiple streams with similar behavior and trend together, instead of to cluster the data records within one data stream.

There are various applications where it is desirable to cluster the streams themselves rather than the individual data records within them. For example, the price of a stock may rise and fall from time to time. To reduce the financial risk, an investor may prefer to spread his investment over a number of stocks which may exhibit different behaviors. As another application, in meteorological study and disaster prediction, it is useful to cluster meteorological data streams from different geographical regions of similar curvature trends in order to identify regions with similar meteorological behaviors. Yet another example is that a super market may record sales on different merchandizes. There may be some relationship among the sales of different merchandizes and thus the merchant can make use of the correlation to manipulate the prices to maximize the profit.

Clustering refers to partition a data set into clusters such that members within the same cluster are similar in a certain sense and members of different clusters are dissimilar. Current clustering techniques can be broadly classified into several categories: partitioning methods (e.g., *k*-means and *k*-medoids), hierarchical methods (e.g. BIRCH [22]), density-based methods (e.g. DBSCAN [15]), and grid-based methods (e.g. CLIQUE [4]). However, these methods are designed only for static data sets and can not be directly applied to data streams.

Source: Data Mining and Knowledge Discovery in Real Life Applications, Book edited by: Julio Ponce and Adem Karahoca,
ISBN 978-3-902613-53-0, pp. 438, February 2009, I-Tech, Vienna, Austria

An abundant body of researches on clustering data in one data stream has emerged. O'Callaghan et al. [19] provided an algorithm called STREAM based on *k*-means, which adopts a divide-and-conquer technique to deal with buckets to get clusters. CluStream [1] is an algorithm for the clustering of evolving data streams based on user-specified, on-line clustering queries. It divides the clustering process into on-line and off-line components. The online component computes and stores summary statistics about the data stream using micro-clustering, while the offline component does macro-clustering and answers various user questions using the stored summary statistics. In these works, the problem refers to cluster the elements of one individual data stream. These works motivate the online compression and offline computation framework used in the paper but are obviously not applicable to our problem.

Euclidean Distance vs. Correlation Distance The problem of clustering multiple data streams views each data stream as an element to be clustered, which pays attention to the similarity between streams. There are several previous works on this problem. Yang [21] uses the weighted aggregation of snapshot deviations as the distance measure between two streams, which can observe the similarity of data values but ignore the trends of streams.

Beringer et al. [6] proposed a preprocessing step which uses a discrete Fourier transforms (DFT) approximation of the original data, uses the few low-frequency (instead of all) coefficients to compute the distance between two streams, and applies an online version of the *k*-means algorithm. Such a method acts like a low-pass filter to smooth the data stream. The DFT transformation preserves the Euclidean distances. Thus, the DFT distance is equivalent to the Euclidean distance of the smoothed data streams.

A serious limitation of the above previous works is that they are based on the Euclidean distance of data records, but important trend information contained in data streams is typically discarded by clustering methods based on Euclidean distance. This is true because data streams with similar trends may not be close in their Euclidean distance. For example, in stock markets, the absolute prices of stocks from similar area or similar industry can be very different but their trends are close. To capture such similarity, it is more appropriate to use **correlation analysis**, a statistical methods on time series, which measures the resemblance of the trends of multiple data streams.

As an illustration, Figure 1 shows the trends of three stocks on Nylon, chemical fiber, and CPU chip, respectively. Although the two stocks on CPU chip and chemical fiber are closer in their data values and thus their Euclidean distance, the trends of the two stocks on Nylon and Chemical fiber are clearly more close, which can be suggested by their higher correlation coefficient. If using Euclidean distance, we may conclude that the two stocks on CPU chips and chemical fiber are more similar, which does not properly reflect the more interesting trend similarity between nylon and chemical fiber.

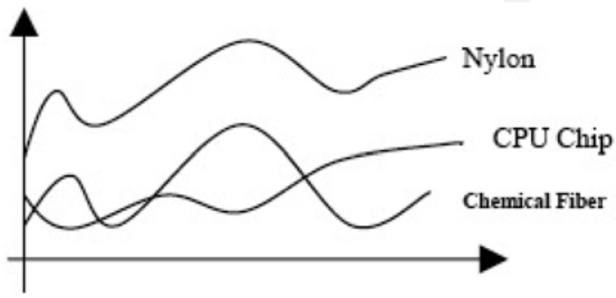

Fig. 1. Price of stocks on nylon, chemical fiber, and CPU chips.

In this paper, we propose an algorithm for clustering multiple data streams based on correlation analysis. Performing correlation analysis on data streams under the one-scan requirement poses significant technical challenges since we cannot not store the raw data. In this paper, we develop a novel scheme that compresses the data online and stores only the compressed measures, called the synopsis, in the offline system. We propose a new theory that facilitate efficient computation of correlation coefficients based on the compressed measures. The correlation coefficients are used to define distances between streams which are in turn used by a *k*-means algorithm to generate the clustering results. An attenuate coefficient is also introduced to allow the algorithm to discover the evolving behaviors of data streams and adjust the clusters dynamically.

Clustering on Demand (COD) More recently, a clustering on demand (COD) framework [7] is proposed to give approximative answers to user queries for clustering sub-streams within certain time window. The framework consists of two components, including the online maintenance phase and the offline clustering phase. The online maintenance phase provides an efficient mechanism to maintain summary hierarchies of data streams with multiple resolutions. The offline phase uses an adaptive clustering algorithm to retrieve approximations of desired sub-streams from summary hierarchies according to clustering queries.

In this paper, we also extend our clustering algorithm to support real-time COD. We propose an innovative scheme to partition the time horizon into segments and store statistical information for each time segment. We prove that the scheme enables us to accurately approximate the correlation coefficients for an arbitrary time duration, and thus allows the users to obtain clusters for data streams within the requested time window.

The paper is organized as follows. After introducing the basic concepts and problem definitions in Section 2, we propose our algorithm in Section 3. We then discuss the extension to COD in Section 4. In Section 5, we show experiment results on synthetic data sets and real data sets which demonstrate the high accuracy, efficiency, and scalability of our algorithm compared with others. We conclude the paper in Section 6.

2. Background

In this section, we introduce the background of the work and several basic concepts.

2.1 Clustering data streams

A data stream X is a sequence of data items $x_1, \ldots, x_k$ arriving at discrete time steps $t_1, \ldots, t_k$. We assume that there are n data streams $\{X_1, \ldots, X_n\}$ at each time step m, where $X_i = \{x_{i1}, \ldots, x_{ik}\}$, $1 \le i \le n$, and x_{ij} $(j = 1, \ldots, m)$ is the value of stream X_i at time j.

The problem of clustering multiple data streams is defined as follows. Given the time horizon for clustering L and the number of clusters k, the clustering algorithm partitions n data streams into k clusters $C(L) = \{C_1(L), \ldots, C_k(L)\}$ that minimizes some objective function measuring the quality of clustering in the period $[t - L + 1, t]$, where t is the time when the analysis is performed. The given clusters $C_j(L)$, $j = 1, \ldots, k$, should satisfy:

$$\bigcap_{j=1}^{k} C_j(L) = \varnothing \text{ and } \bigcup_{j=1}^{k} C_j(L) = \{X_1(L), \ldots, X_n(L)\},$$

where $X_i(L) = \{x_{i(t-L+1)}, \ldots, x_{it}\}$, $i = 1, \ldots, n$.

2.2 Attenuation coefficient

In stream data analysis, in order to recognize the evolving characteristics of data streams, newer data records are often given more weights than older ones. Therefore, we use an *attenuation coefficient* $\lambda \in [0, 1]$ to gradually lessen the significance of each data record over time. Suppose t is the current time and a data point x_i is received at time i, then, in our analysis, we replace the original value of x_i with

$$x_i(t) = \lambda^{t-i} x_i. \tag{1}$$

Applying this adjustment to every element in the data streams in the time horizon $[t - L + 1, t]$, we replace the original values by

$$\begin{aligned} & \{x_{t-L+1}(t), \ldots, x_t(t)\} \\ = \; & \{\lambda^{L-1} x_{t-L+1}, \lambda^{L-2} x_{t-L+2}, \ldots, \lambda x_{t-1}, x_t\}. \end{aligned} \tag{2}$$

2.3 Time segment

We first consider fixed-length clustering and then extend the algorithm for arbitrary-length clustering in COD in Section 4. Given a fixed length, at any time t, we report in real time the clustering results for the data streams in the time horizon $[t - L + 1, t]$. To support efficient processing, we partition the data streams of length L into m *time segments* of equal length $l = L/m$. Whenever a new segment of length l accumulates (Figure 2), we re-compute the clustering results.

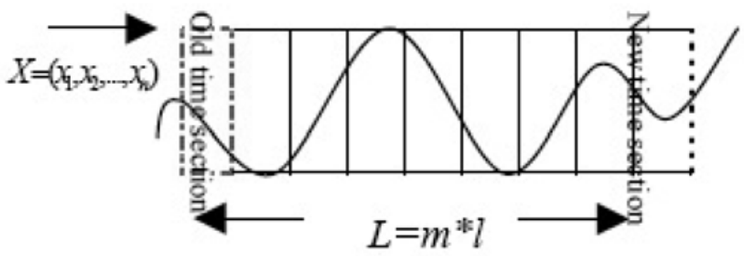

Fig. 2. A fixed length L is divided into m segments of size l.

3. The CORREL-cluster algorithm

In the following, we propose a general framework called **CORREL-cluster**, a correlation-based clustering algorithm for data streams. Unlike the widely-used Euclidean distance which only measures the discrepancy of the data values, our correlation-based distance considers two data streams with similar trends to be close to each other.

We first overview the overall framework of the proposed CORREL-cluster algorithm. The framework of the algorithm is in Figure 3.

CORREL-cluster continuously receives new data records from all the data streams at each time step (Line 5 in Figure 3). It keeps outputing the clusters for the most recent data streams of fixed length L. For every l time steps, it first computes a compressed representation of the data streams for the time segment $[t - l + 1, t]$ (Line 7), discards the raw data, and updates the compressed representation of the data streams for the target

clustering time [$t-L+1$, t] (Lines 9-10). It then calls a correlation-based k-means algorithm (Line 11) to compute the clustering results. Since the number of clusters k may be changing, CORRELcluster also employs a new algorithm to dynamically adjust k in order to recognize the evolving behaviors of the data streams (Line 12). The algorithm is schematically shown in Figure 4.

```
1.  procedure CORREL-cluster
2.  t = 0;
3.  while data stream is not terminated
4.      set t = t + 1;
5.      read new data record x_kt(t), k = 1 ··· n, one from each of
        the n data streams;
6.      if (t mod l == 0) then    /* form a new segment. */
7.          calculate the CCR of the time segment [t-l+1, t];
8.          update other m-1 CCRs, where m = L/l;
9.          if t == L then compute initial CCR_L, the compressed
            correlation representation for [t-L+1, t];
10.         else incrementally update CCR_L;
11.         call correlation_k_means();    /* compute clusters /*
12.         call adjust_k();
13.         output the clustering result;
14.     end if
15. end while
16. end_procedure
```

Fig. 3. The overall process of CORREL-cluster.

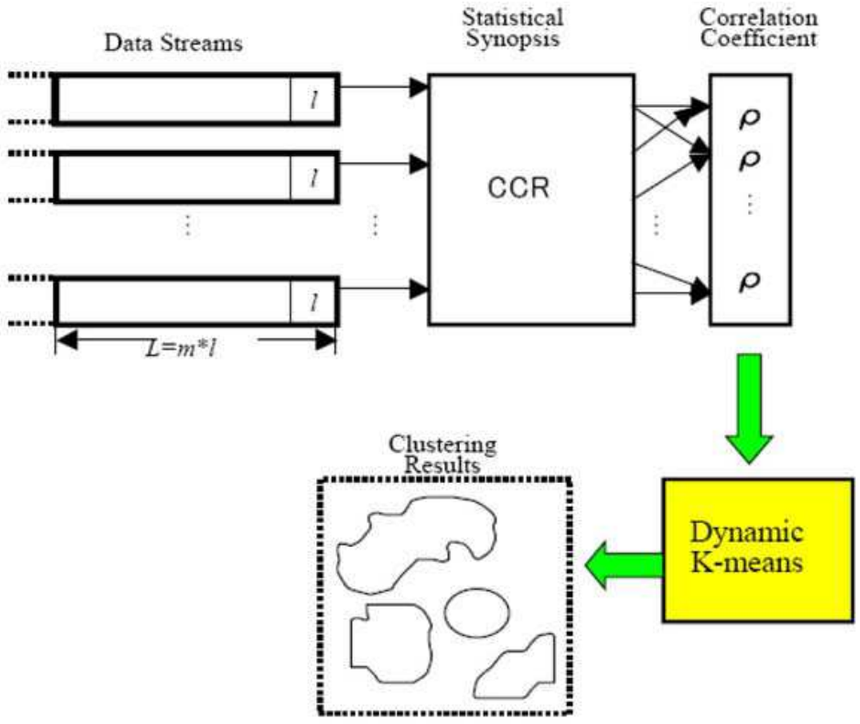

Fig. 4. Illustration of CORREL-cluster.

3.1 Correlation analysis of data streams

Before describing the details of our algorithm, we first overview the concepts of correlation analysis.

We first define the correlation coefficients for two data streams. For two data streams $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$, the correlation coefficient between them is defined as

$$\rho_{XY} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{j=1}^{n}(y_j - \bar{y})^2}}, \tag{3}$$

where $\bar{x} = (\sum_{i=1}^{n} x_i)/n$ and $\bar{y} = (\sum_{i=1}^{n} y_i)/n$.

From the definition, we can see that $|\rho_{XY}| \leq 1$. A large value of $|\rho_{XY}|$ indicates strong correlation between streams X and Y, and $\rho_{XY} = 0$ means X and Y are uncorrelated.

Since it is often impossible to store all the past raw data in the stream, we need to compress the raw data and only retain a synopsis for each time segment of each data stream.

The following theorem shows that, to compute the correlation coefficient we only need to save $\Sigma_i x_i$, $\Sigma_i x_i^2$ for each stream X and $\Sigma_i x_i y_i$ between any two streams to compute the correlation coefficients between any two streams.

Theorem 3.1 *Correlation coefficient ρ_{XY} between two sequences X and Y can be calculated using the information $\Sigma_i x_i$, $\Sigma_i x_i^2$, $\Sigma_i y_i$, $\Sigma_i y_i^2$, and $\Sigma_i x_i y_i$.*

Proof. In (3), the numerator can be rewritten as:

$$\begin{aligned}
&\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \\
=\ &\sum_{i=1}^{n}(x_i y_i - \bar{x}y_i - x_i\bar{y} + \bar{x}\bar{y}) \\
=\ &\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n}\bar{x}y_i - \sum_{i=1}^{n} x_i\bar{y} + \sum_{i=1}^{n}\bar{x}\bar{y} \\
=\ &\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y} - \bar{x}\bar{y} + \bar{x}\bar{y} \\
=\ &\sum_{i=1}^{n} x_i y_i - \frac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i.
\end{aligned} \tag{4}$$

And in the denominator, we have

$$\begin{aligned}
\sum_{i=1}^{n}(x_i - \bar{x})^2 &= \sum_{i=1}^{n}(x_i^2 - 2\bar{x}x_i + \bar{x}^2) \\
&= \sum_{i=1}^{n} x_i^2 - n\bar{x}^2 \\
&= \sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2.
\end{aligned} \tag{5}$$

Similarly, we have

$$\sum_{i=1}^{n}(y_i-\bar{y})^2=\sum_{i=1}^{n}y_i^2-\frac{1}{n}\left(\sum_{i=1}^{n}y_i\right)^2. \quad (6)$$

Therefore, we can compute both the numerator and denominator in (3) using $\Sigma_i x_i$, $\Sigma_i x_i^2$, $\Sigma_i y_i$, $\Sigma_i y_i^2$, and $\Sigma_i x_i y_i$.

Based on the above result, in CORREL-cluster, for any time segment, we store a compressed synopsis for the parallel data streams instead of the raw data.

Definition 3.1 (Compressed Correlation Representation (CCR)) Given n data streams $X_1, \ldots, X_n$, suppose the current time is t, the time segment length is l, then the we store the following quantities in $CCR = (\vec{S}, \vec{Q}, \mathbf{C}, t)$, where the components of the vectors $\vec{S}, \vec{Q}$ and matrix **C** are defined as:

$$S_i=\sum_{k=t-l+1}^{t}x_{ik}(t),\quad i=1,\cdots,n$$

$$Q_i=\sum_{k=t-l+1}^{t}x_{ik}^2(t),\quad i=1,\cdots,n$$

$$C_{ij}=\sum_{k=t-l+1}^{t}x_{ik}(t)x_{jk}(t),\quad i,j=1,\cdots,n,i<j$$

Based on Theorem 3.1, CCR provides enough information to compute the correlation coefficients between any two streams in n data streams $X_1, \ldots, X_n$.

Theorem 3.2 *For two streams X_i and X_j, $i, j = 1..n$, their correlation coefficients can be computed as*

$$\rho_{X_iX_j}=\frac{C_{ij}-\frac{1}{n}S_iS_j}{\sqrt{(Q_i-\frac{1}{n}S_i^2)(Q_j-\frac{1}{n}S_j^2)}}$$

The theorem can be seen from equations (3), (4), (5), and (6).

3.2 Update of the CCR synopsis

For a given time segment, at the current time t_c, the CCR for the n streams is

$$\left\{\sum x_{ik}(t_c),\sum x_{ik}^2(t_c),\sum x_{ik}(t_c)x_{jk}(t_c)\right\}.$$

Then at any time $t > t_c$, the data values are updated to $\{x_{ik}(t)\}$ instead of $\{x_{ik}(t_c)\}$. To perform online clustering analysis, we also need to update the CCR.

Let $t - t_c = \Delta_t$ and λ be the attenuation coefficient, then

$$x_{ik}(t)=\lambda^{\Delta_t}x_{ik}(t_c).$$

Therefore, we can update the saved information as follows.

$$\begin{aligned}\vec{S}'_i &= \sum_{k=t_c-l_c+1}^{t_c} x_{ik}(t) = \sum_{k=t_c-l_c+1}^{t_c} \lambda^{\Delta_t} x_{ik}(t_c) \\ &= \lambda^{\Delta_t} \sum_{k=t_c-l_c+1}^{t_c} x_{ik}(t_c) = \lambda^{\Delta_t} \vec{S}_i\end{aligned} \tag{7}$$

$$\begin{aligned}\vec{Q}'_i &= \sum_{k=t_c-l_c+1}^{t_c} x_{ik}^2(t) = \sum_{k=t_c-l_c+1}^{t_c} [\lambda^{\Delta_t} x_{ik}(t_c)]^2 \\ &= \lambda^{2\Delta_t} \sum_{k=t_c-l_c+1}^{t_c} x_{ik}^2(t_c) = \lambda^{2\Delta_t} \vec{Q}_i\end{aligned} \tag{8}$$

$$\begin{aligned}\mathbf{C}'_{ij} &= \sum_{k=t_c-l_c+1}^{t_c} x_{ik}(t) x_{jk}(t) = \sum_{k=t_c-l_c+1}^{t_c} \lambda^{2\Delta_t} x_{ik}(t_c) x_{jk}(t_c) \\ &= \lambda^{2\Delta_t} \sum_{k=t_c-l_c+1}^{t_c} x_{ik}(t_c) x_{jk}(t_c) = \lambda^{2\Delta_t} \mathbf{C}_{ij}\end{aligned} \tag{9}$$

We note that such an update happens not at every time step, but every l steps (Line 8 in Figure 3). Whenever a new time segment comes in, we compute the new CCR. Also, there are $m = L/l$ existing segments and thus m CCRs. We discard the oldest CCR, and update the other $m-1$ segments according to the above formulae.

3.3 Aggregating *CCR* to CCR_L

In CORREL-cluster, for a user specified clustering length L, we need to cluster streams within the time $[t - L + 1, t]$. Therefore, for each pair of streams (X, Y), we need to compute the correlation coefficients for $X[t - L + 1, t]$ and $Y[t - L + 1, t]$. Since we compute for each time segment with length l a CCR, we need to combine them into CCR_L, the CCR for the time window $[t - L + 1, t]$.

To formalize the problem, we have m time segments and m CCRs. Let $CCR(v)$ be the CCR for time segment $[t - vl + 1, t - (v - 1)l]$, for $v = 1..m$, we denote the components of $CCR(v)$ as

$$CCR(v) = (\vec{S}(v), \vec{Q}(v), \mathbf{C}(v)), \tag{10}$$

and CCR_L as

$$CCR_L = (\vec{S}_L, \vec{Q}_L, \mathbf{C}_L), \tag{11}$$

We have, at time t:

$$\begin{aligned}\vec{S}_{L_i} &= \sum_{k=t-L+1}^{t} x_{ik}(t) = \sum_{v=1}^{m} \Big(\sum_{k=t-vl+1}^{t-(v-1)l} x_{ik}(t) \Big) \\ &= \sum_{v=1}^{m} \vec{S}_i(v), \quad \forall i = 1, \cdots, n,\end{aligned} \tag{12}$$

which can be compactly written as

$$\vec{S}_L = \sum_{v=1}^{m} \vec{S}(v). \tag{13}$$

Similarly, we have:

$$\vec{Q}_L = \sum_{v=1}^{m} \vec{Q}(v), \quad \text{and} \quad \mathbf{C}_L = \sum_{v=1}^{m} \mathbf{C}(v). \tag{14}$$

We use the above equations to compute CCR_L when we receive the first m time segments (Line 9 of Figure 3).

For later updates (Line 10), we do not need to redo the summation. In fact, we can incrementally update CCR_L. Given the existing m CCRs: $CCR(i)$, $i = 1, ...,m$, and the newly generated $CCR(new) = (\vec{S}(new), \vec{Q}(new), \mathbf{C}(new))$. We update the CCR_L as:

$$\vec{S}'_L = \vec{S}_L + \vec{S}(new) - \vec{S}(1) \tag{15}$$

$$\vec{Q}'_L = \vec{Q}_L + \vec{Q}(new) - \vec{Q}(1) \tag{16}$$

$$\mathbf{C}'_L = \mathbf{C}_L + \mathbf{C}(new) - \mathbf{C}(1). \tag{17}$$

We also update the saved CCRs by

$$CCR(i) = CCR(i+1), \quad i = 1, \cdots, m-1 \tag{18}$$

$$CCR(m) = CCR(new). \tag{19}$$

3.4 Dynamic *k*-means algorithm

We use a k-means clustering algorithm to generate the cluster for data streams in the user-specified window $[t-L+1, t]$. In the k-means algorithm, the distance $d(X, Y)$ between two data streams X and Y is measure using the reciprocal of the correlation coefficient:

$$d(X, Y) = 1/\rho_{XY}. \tag{20}$$

The clustering quality is measured by an objective function

$$G = \sum_{i=1}^{k} \sum_{j=1}^{n} (1/\rho_{X_j C_i}), \tag{21}$$

where ρ_{XjCi} is the correlation coefficient between data stream X_j and cluster center C_i, which is a stream from X_1 to X_n.

Let n be the number of streams to be clustered. The correlation-based k-means algorithm is shown in Figure 5.

In practice, a major advantage of the algorithm is that it typically takes very few steps to converge. This is due to the fact that the clusters are not changing very fast over a time gap l.

Consider two consecutive clustering calls at time t and $t+l$, then the time windows for clustering, $[t-L+1, t]$ and $[t+l-L+1, t+l]$, significantly overlap with each other. As a result, each clustering often converges in a few steps if we start from the previous clustering result.
As the data streams change over time, new clusters may emerge and existing clusters may disappear. A drawback of the conventional k-means algorithm is that the user needs to specify the number of clusters k.
To capture this dynamic evolution of data streams, we continuously update the number of clusters k every time a new time segment with length l is received. We assume that when k is updated regularly and frequently, it will not change abruptly. Therefore, if the number of clusters given by the previous clustering is k, we will only consider $k - 1$, k or $k + 1$ to be the current number of clusters k'. Then we choose k' as the one that produces the smallest objective function G. That is,

$$G_{k'} = \min\{G_{k-1}, G_k, G_{k+1}\}.$$

Algorithm correlation-k-means(k, $Center_k$, R_k)
Input: number of the clusters k, sets of the center points $Center_k$, current clustering result R_k;
Output: updated clustering results R_k and its objective function value G_k;
begin

1. **repeat**
2. **for** $i = 1$ **to** n
   calculate the correlation distances between stream X_i and centers of k clusters;
   assign X_i to the cluster with the shortest distance;
   end for
3. compute the new center of each cluster, update the set of centers $Center_k$;
4. **until** no change of clustering result
5. calculate the objective function G_k

end

Fig. 5. The algorithm for clustering.

When considering $k' = k - 1$, we initialize the clusters by merging the two clusters that are closest in their centers; when considering $k' = k + 1$, we initialize the clusters by forming a new cluster as the stream in existing clusters that is the farthest from the its cluster center. After the initial clustering is given, we run k-means until it converges in order to measure the quality of the adjusted clusters. The adjust k algorithm is shown in Figure 6.

4. Clustering on demand

Our above discussion only considers clustering data streams over a time period of fixed length L. In some applications, the length of the time period depends on users' demands. Here, we extend our clustering algorithm to support clustering on demand (COD), i.e. clustering over any time horizon at user's request [7]. We call the extended algorithm CORREL-COD. The CORREL-COD algorithm has an online component and an offline component. The online component calculates the summary information in CCRs, whereas the offline part performs clustering.

Algorithm adjust_k(k, $Center_k$, R_k)
Input: number of the clusters k, sets of the center points $Center_k$, current clustering result R_k
Output: updated number of clusters k', updated clustering result $R_{k'}$ and its objective function value $G_{k'}$
begin

1. Calculation of R_{k+1}
   (a) Among all the clusters, choose the data stream X which is the farthest from its cluster center, set a new cluster with X as its center;
   (b) $Center_{k+1} = Center_k \cup \{X\}$
   (c) correlation-k-means($k+1$,$Center_{k+1}$,R_{k+1})
2. Calculation of R_{k-1}
   (a) Choose two closest clusters, suppose their centers are C_1 and C_2, respectively
   (b) combine these two clusters into a new cluster, compute the center C_3 of the new cluster
   (c) $Center_{k-1} = Center_k \cup \{C_3\} - \{C_1, C_2\}$
   (d) correlation-k-means($k-1$,$Center_{k-1}$,R_{k-1})
3. choose the one with the best G from R_{k-1}, R_k, R_{k+1}, set k' and $R_{k'}$ accordingly;

end

Fig. 6. The algorithm for adjusting k.

We assume that the maximum time horizon over which the user will demand is L. Namely, we only need to preserve information for time period $[t - L + 1, t]$.
We perform k-means clustering in the offline part to meet the user's demands. The offline part first receives summary information from the online processor and calculates correlation coefficients and distances between streams, then performs the k-means algorithm to cluster the data streams. Since we cannot afford to store the raw data, we divide the time horizon into multiple segments and store the CCR synopsis for each segment. Therefore, for an arbitrary time horizon, it is often impossible to extract information over the exact horizon. We assemble a combination of partitioned time segments in such a way that minimizes the difference between the user-specified time horizon w and the best approximative time horizon w' over which we can get CCR synopsis.
Let $L = 2^l$ and m be the maximum number of segments the memory can store. We need to design appropriate **partitioning scheme** on the lengths of segments. One way is to divide L into m parts of equal length L/m. The maximum difference between w and w' will be L/m. Since all segments have the same length, there may be excessive loss for the recent segments that contain newer, and hence more interesting and valuable, information. Another way is to assign segment lengths as 1, 2, 2^2, 2^3, . . . , 2^{l-1}, which means that we store CCRs for time segments $[t - 1, t]$, $[t - 4, t - 2]$, $[t - 8, t - 5]$, ..., $[t - L + 1, t - L/2]$. The segments containing newer data will have shorter lengths, leading to higher clustering accuracy for newer data. However, in this way, the maximum difference between w and w' is will be $L/2$, which is excessively large.
We propose a new scheme that places more weights on recent data while making the difference $|w - w'|$ as small as possible. We assume $m > \log L$. In our scheme, we arrange

segments with lengths of 1, 2, 2^2, 2^3, . . . , 2^{l-1}, respectively. Let $m' = m - \log L = m - l$, and S_i denote the segment with length 2^i. For the m' unassigned segments, we assign them according to the following rule.

We first remove S_{l-1}, replace the region covered by S_{l-1} by two more S_{l-2}, and then reduce m' by one. If $m' > 0$, we will keep splitting a larger segment into two smaller segments. When there are still S_{l-2} segments, we split the most recent S_{l-2} into two S_{l-3}, and then reduce m' by one, until all S_{l-2} segments are removed or $m' = 0$. If all S_{l-2} segments are removed and $m' > 0$, we will split the most recent S_{l-3} into two S_{l-4} and reduce m' by one. We repeat this process until $m' = 0$.

Given L, m, and $m' = m - \log L$, it is easy to show that using our scheme, the maximum length of any segment is 2^k, where

$$k = \text{argmax}_i(m' \leq C_i) - 1 \tag{22}$$

where

$$C_i = 2^{l-i+2} - (l - i + 4). \tag{23}$$

Let T_i the number of segments of type S_i, we can prove that (the proof is omitted):

$$T_k = C_{k+1} - m' + 1 \tag{24}$$

$$T_{k-1} = 2(2^{l-k} - T_k) - 1. \tag{25}$$

$$T_i = 1,\ i = 0, 1, \ldots, k-2 \tag{26}$$

$$T_i = 0 \text{ for } i > k. \tag{27}$$

Example: Suppose $L = 1024$, $m = 20$, $l = 10$. Then $m' = m - \log L = 10$. By (23), we get $C_9 = 3$, $C_8 = 10$, $C_7 = 25$. This gives $k = 7$ since $\text{argmax}_i(m' \leq C_i) = 8$. Then by (24), (25), and (26), we get

$$T_7 = 10 - 10 + 1 = 1,\ T_6 = 2(2^{(10-7)} - 1) - 1 = 13$$

and

$$T_5 = T_4 = T_3 = T_2 = T_1 = T_0 = 1.$$

Theorem 4.1 *Using the above partitioning scheme, we have m segments in total.*

Proof. The number of segments is

$$\begin{aligned}
\sum_{i=1}^{k} T_i &= T_k + 2(2^{l-k+1} - T_k) - 1 + (k-2) \\
&= 2^{l-k+2} - T_k + k - 3 \\
&= 2^{l-k+2} - (C_k - m' + 1) + k - 3 \\
&= 2^{l-k+2} - 2^{l-k+2} + (l - k + 4) + m' + k - 4 \\
&= m' + l = m.
\end{aligned}$$

Theorem 4.2 *Using the above partitioning scheme, the total length of the m segments is L − 1.*

Proof. The total length of all segments is

$$\begin{aligned}\sum_{i=0}^{k} 2^i T_i &= T_k 2^k + T_{k-1}2^{k-1} + 2^{k-2} + 2^{k-3} + \cdots + 2 + 1 \\ &= T_k 2^k + [2(2^{l-k} - T_k) - 1]2^{k-1} + 2^{k-1} - 1 \\ &= 2^l - 1 = L - 1.\end{aligned}$$

Theorem 4.3 *Suppose that the user demands a query for segments of length $r \le L$. Using the above partitioning scheme, suppose r' is the total length of a set of selected segments that is closest to r. Then $r' - r \le 2^k$, where $k = \text{argmax}_i(m\_ \le C_i) - 1$.*

Proof. To form a set of the most recent segments with a total length closest to r, we can incrementally add S_1, S_2, ... to the set until the total length of the selected segments r' is larger than r. Suppose the longest segment in the resulting set is S_g, Since $r < L$we have $g \le k$, and $r' - r \le 2^g \le 2^k$.

The above results show that our partitioning schemes achieves two goals. First, we assign shorter segments to newer data and thus give newer data higher precision. Second, the largest possible difference between the length of the approximative combination of segments and the user requested length is lowered to 2^k, which is much smaller than $L/2$.

5. Experimental results

To evaluate the performance of our algorithms, we test it using both synthetic data and real data on a PC with 1.7GHz CPU and 512 MB memory running Window XP. The systems are implemented using Visual C++ 6.0.

5.1 Testing data

We generate the synthetic data in the same way as in [6]. For each cluster, we first define a prototype $p(\cdot)$ which is a stochastic process defined by a second-order difference equation:

$$\begin{aligned}p(t + \Delta t) &= p(t) + p'(t + \Delta t) \\ p'(t + \Delta t) &= p'(t) + u(t), \quad t = 0, \Delta t, 2\Delta t, \ldots\end{aligned}$$

where $u(t)$ are independent random variables uniformly distributed in an interval $[-a, a]$. The data streams in the cluster are then generated by "distorting" the prototype, both horizontally (by stretching the time axis) and vertically (by adding noise). The formulation for a data stream $x(\cdot)$ is defined as:

$$x(t) = p(t + h(t)) + g(t),$$

where $h(\cdot)$ and $g(\cdot)$ are stochastic processes generated in the same way as the prototype $p(\cdot)$. The constant a that determines the smoothness of a process can be different for $p(\cdot)$, $h(\cdot)$, and $g(\cdot)$, such as 0.04, 0.04, 0.5, respectively.

We can then generate different clusters by generating different prototype function $p(\cdot)$. For each prototype function, we randomly distort the prototype to general multiple data streams in that cluster.

The real data set (Figure 7) that we use contains average daily temperatures of 169 cities around the world, recorded since January 1, 1995 to present. Each city is regarded as a data stream and each stream has 3,416 points.

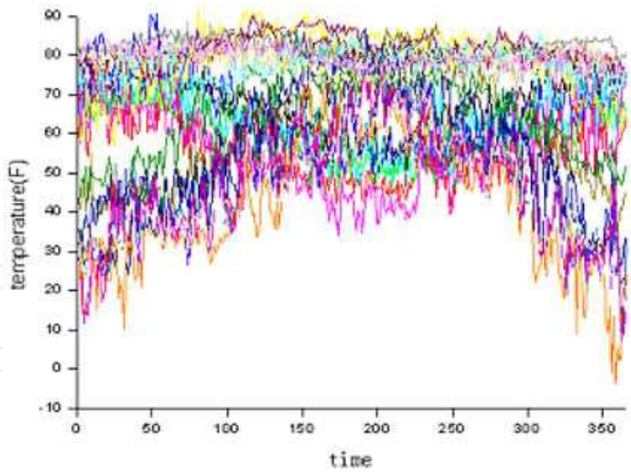

Fig. 7. Daily temperatures for 169 cities around the world.

5.2 Performance analysis on CORREL-cluster

5.2.1 Clustering results

We ran CORREL-cluster on the real data set to cluster cities based on the recorded daily temperatures. We set L = 360 and l = 30. The input of the algorithm is the daily temperatures of cities around the world. CORREL-cluster gave five clusters each of which contains cities mostly in the same continent and belonging to the same temperature zone. The correct rate is around 85% to 89%. The results are shown in Figures 8-12, where each graph shows one cluster.

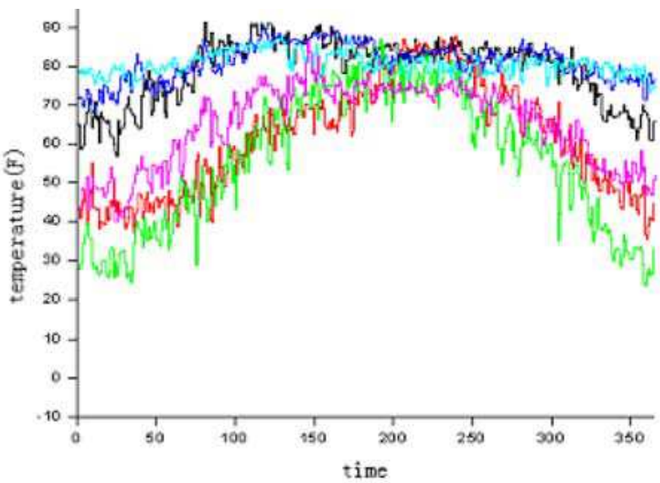

Fig. 8. Cluster 1: cities in Asia

5.2.2 Quality

We evaluate the quality of the clustering from CORREL-cluster by comparing with that from DFT-cluster [6] (30 DFT coefficients). Figure 13 shows a comparison of the quality of clustering on the real city-temperature data set by CORREL-cluster and DFT-cluster for various number of segments. The quality is measured by correct rate, the ratio of the number of cities that are correctly labelled to the total number of cities.

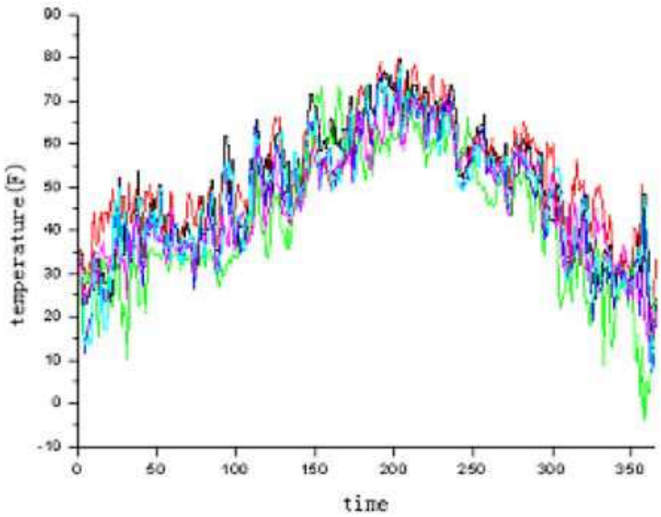

Fig. 9. Cluster 2: cities in Europe.

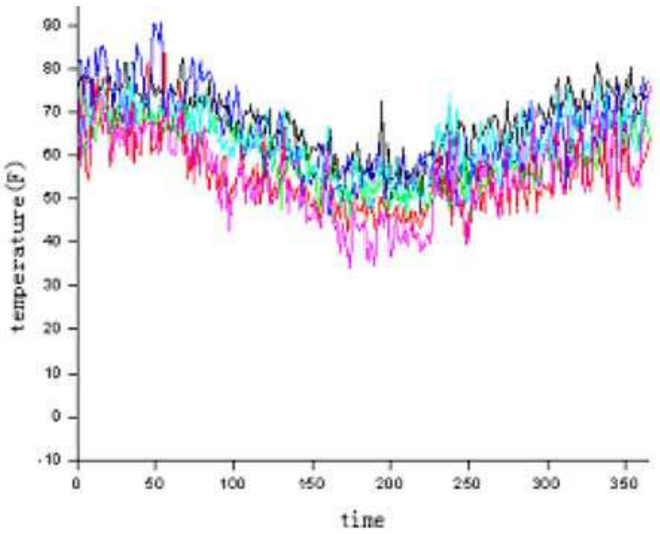

Fig. 10. Cluster 3: cities in Oceania.

Since we use a fixed time horizon L = 360, the larger the number of segments is, the more frequently clustering is executed. Thus, for both algorithms, the quality improves when the number of segments increases. However, as we can see from Figure 13, CORREL-cluster always has a better quality than DFTcluster.

5.2.3 Speed

Since the clustering on the real data set is too fast, we use synthetic data sets to test the processing speed of CORREL-cluster. We generate 6 synthetic data sets each containing 100 data streams. Each data stream has 65,536 data elements. Again, we compare with DFT-cluster (250 DFT coefficients). The experimental results show that the executing time for CORREL-cluster is shorter than that of DFT-cluster for every synthetic data set. Figure 14 shows that, the average processing time per segment for CORREL-cluster isn 0.928 seconds whereas 1.2 seconds for DFT-cluster using 250 DFT coefficients. DFT-cluster needs even

longer processing time when more coefficients are used. When using 1500 DFT coefficients, DFT-cluster takes in average over 7 seconds. Reducing the number of DFT coefficients can save time but will lead to worse quality. As we see in Figure 15, DFT-cluster with 250 DFT coefficients has much worse quality than CORREL-cluster on these synthetic data sets.

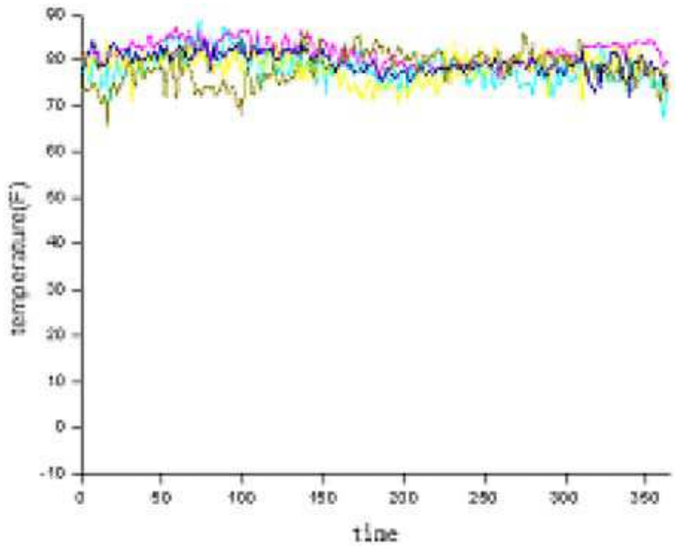

Fig. 11. Cluster 4: cities in Africa.

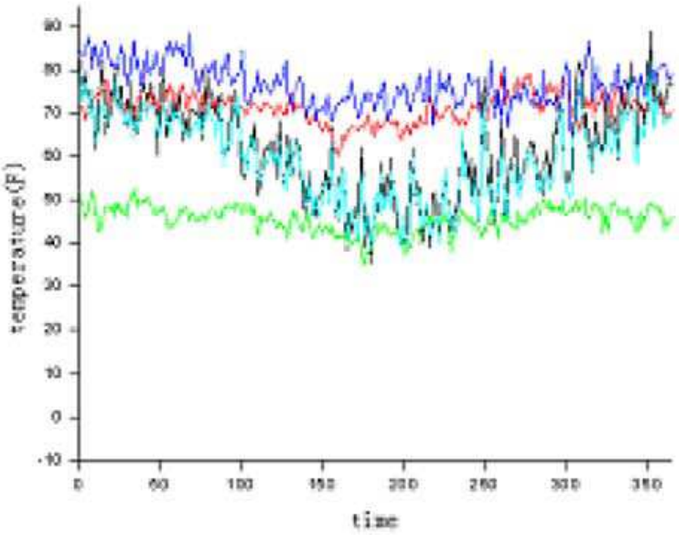

Fig. 12. Cluster 5: cities in South America.

5.2.4 Dynamic number of clusters

CORREL-cluster requires an initial value k for the number of clusters. We study its sensitivity to k by trying different values of k. Figure 16 shows the clustering results of CORREL-cluster on a synthetic data set with three clusters for different initial values of k, including 2, 3, 4, 6, 8, 10 and 20. We see that the number of clusters given by CORREL-

cluster soon becomes the same regardless the initial value, which indicates that the initial value of k has little influence on the clustering performance. This stability is due to our adjust $k()$ algorithm that adaptively changes the number of clusters in the process of clustering. This adaptiveness can be seen in Figure 16. For example, five clusters are found at time 32, four at time 38, and three at 52.

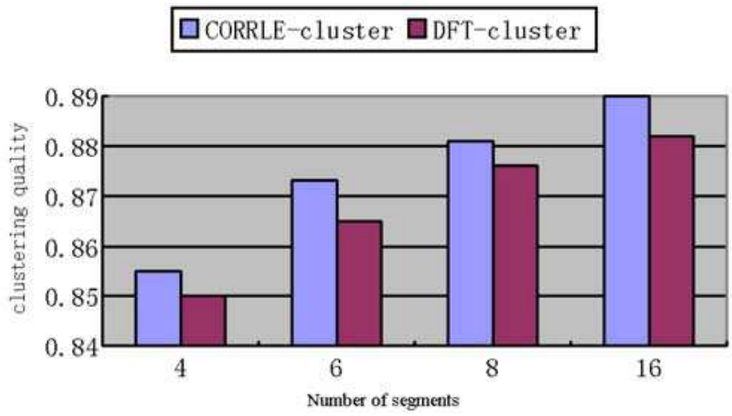

Fig. 13. Clustering quality of CORREL-cluster and DFT-cluster on real data.

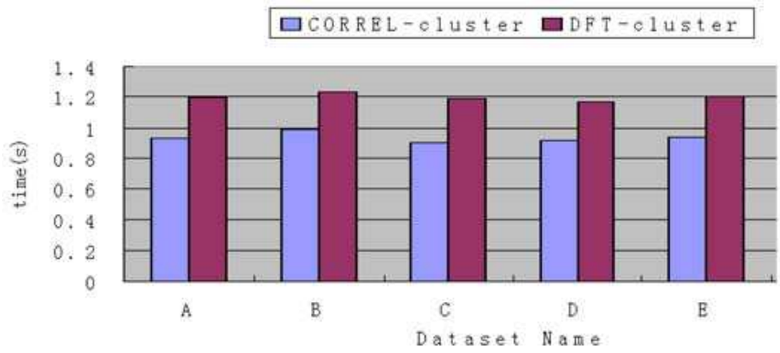

Fig. 14. Computation time of CORREL-cluster and DFT-cluster on synthetic data.

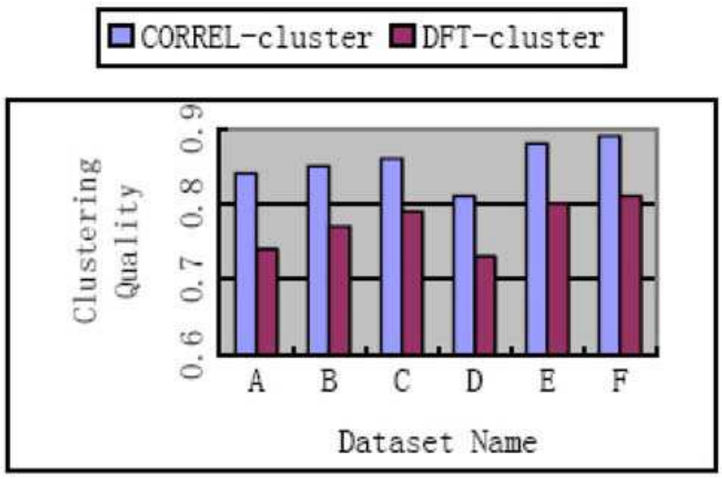

Fig. 15. Quality of CORREL-cluster and DFT-cluster on synthetic data.

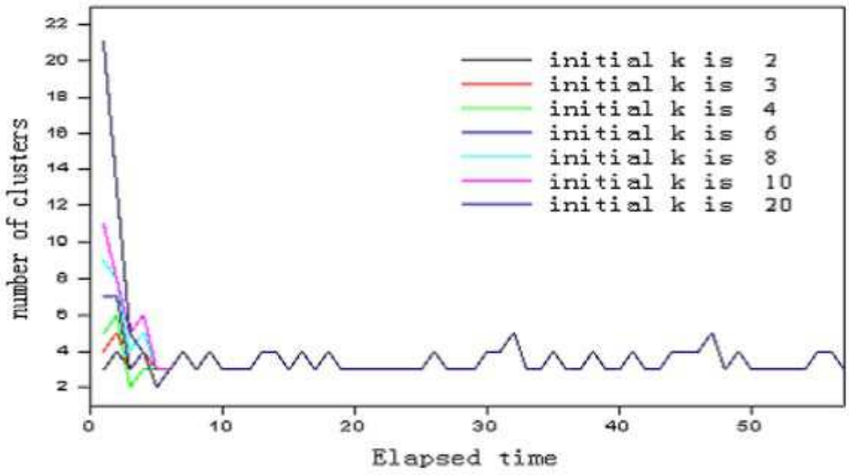

Fig. 16. Number of clusters found by CORREL-cluster for different initial values of *k*.

5.3 Performance analysis on CORREL-COD

5.3.1 Scalability

To evaluate the scalability of the online processing, we test our CORREL-COD algorithm and ADAPTIVEcluster [7] using several randomly generated synthetic data sets of sizes varying from 1,000 to 10,000. As we see in Figure 17, the execution time for both algorithms increases linearly with the number of data points, but CORREL-COD is always more efficient than ADAPTIVE-cluster.

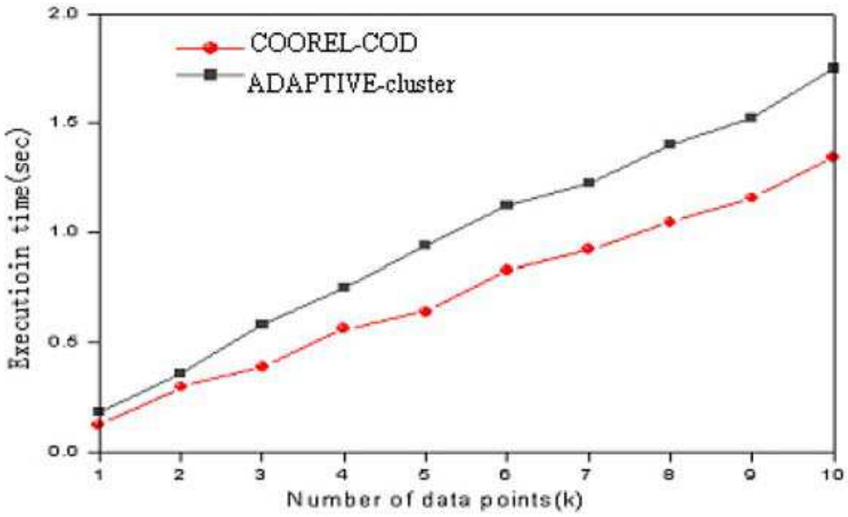

Fig. 17. Comparison of the scalability of CORRL-COD and ADAPTIVE-cluster.

5.3.2 Quality

We measure the quality of clustering using $G_{rawdata}/G_{COD}$, where $G_{rawdata}$ denotes the objective function obtained by clustering the raw data directly without segmentation and G_{COD} denotes the objective function by clustering based on the summary information retrieved by the online processor.

Figure 18 shows the quality of clustering on two simulated data sets for different number of segments. We see that the larger the number of segment is, the more precise our algorithm achieves. This is because the difference between the user specified length and the length of the approximated statistical information becomes smaller when the number of segments increases. From Figure 8, we can see the clustering quality is always above 95%, which means that results by our COD algorithm are close to the optimal results that can be obtained from raw data.

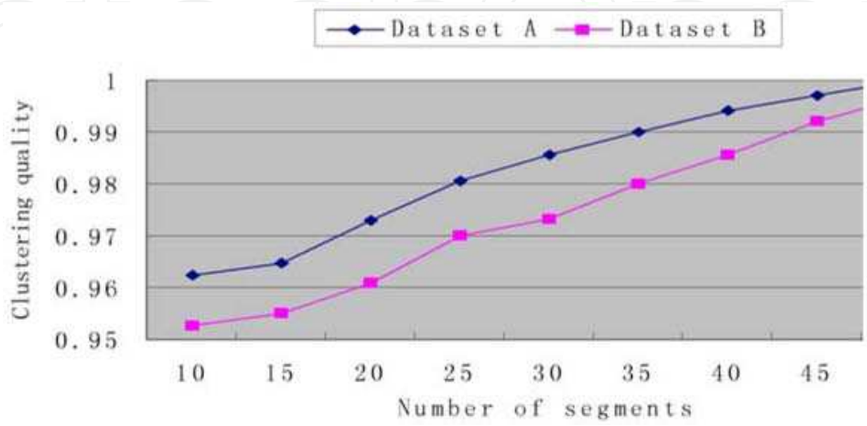

Fig. 18. Clustering quality for different numbers of segments.

6. Conclusions

When our aim is to mine the similarity on the trends of data streams, correlation coefficient is a more appropriate measure for similarity between data streams than Euclidean distance used by previous data stream clustering methods. In this paper, we have developed algorithms CORREL-cluster and CORRELCOD for clustering multiple data streams based on correlation coefficients, which supports online clustering analysis over both fixed and flexible time horizons. Since data streams have high speed and massive volume, we can not retain the raw data to perform correlation analysis. We have proposed a compression scheme that supports an one-scan algorithm for computing the correlation coefficients for. We have developed an adaptive algorithm to dynamically determine the number of clusters so that CORREL-cluster can adjust to the evolving behaviors of data streams. Moreover, we have developed a novel partitioning algorithm to support clustering of arbitrary length per user's request. Experimental results on real and synthetic data sets show that our algorithms have high clustering quality, efficiency, and scalability.

7. References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. of conference of very large databases*, pages 81–92, 2003.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for projected clustering of high dimensional data streams. In *Proc. of conference of very large databases*, pages 852–863, 2004.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. On-Demand Classification of Evolving Data streams. In *Proc. Of International Conference on Knowledge Discovery and Data Mining*, 2004.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of the ACM SIGMOD Conference*, pages 94–105, Seattle, WA, 1998.

[5] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the twenty-second ACM symposium on Principles of database systems*, pages 234–243, 2003.

[6] J. Beringer and E. Hüllermeier. Online-clustering of parallel data streams. *Data and Knowledge Engineering*, 58(2):180–204, 2006.

[7] B.R. Dai, J.W. Huang, M.Y. Yeh, and M.S. Chen. Adapative clustering for multiple evolving streams. *IEEE Transaction On Knowledge and data engineering*, 18(9), 2006.

[8] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proc. of the Eighteenth International Conference on Machine Learning*, pages 106–113, 2001.

[9] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*. AAAI/MIT, 2003.

[10] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Annual IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[11] S. Guha, R. Rastogi, and K. Shim. CURE:An effcient Clustering algorithm for large databases. In *ACM SIGMOD Conference*, pages 73–84, 1998.

[12] M.R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical report, Digital Systems Research Center, 1998.

[13] G. Hulten, L. Spencer, and P. Domingos. Mining time changing data streams. In *Proc. of ACM SIGKDD*, pages 97–106, 2001.

[14] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pages 102–111, 2002.

[15] J. Sander X. Xu M. Ester, H.-P. Kriegel. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, Portland, Oregon, 1996. AAAI Press.

[16] S. Madden and M. Franklin. Fjording the stream: an architecture for queries over streaming sensor data. In *Proc. of ICDE*, pages 555–566, 2002.

[17] G. Manku and R. Motwani. Approximate Frequency counts over data streams. In *Proceedings of conference of very large databases*, 2002.

[18] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of frequent and top-k elements in data streams. In *Proc. Of International Conference on Database Theory*, 2005.

[19] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proc. of 18th International Conference on Data Engineering*, pages 685– 694, 2002.

[20] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining Concept-Drifting data streams using ensemble classifiers. In *Proc. Of International Conference on Knowledge Discovery and Data Mining*, 2003.

[21] J. Yang. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *Proc. of IEEE Int'l Conf. Data Mining*, pages 695–697, 2003.

[22] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. of ACM SIGMOD international conference on Management of data*, pages 103– 114, 1996.

Data Mining and Knowledge Discovery in Real Life Applications

Edited by Julio Ponce and Adem Karahoca

ISBN 978-3-902613-53-0
Hard cover, 436 pages
Publisher I-Tech Education and Publishing
Published online 01, January, 2009
Published in print edition January, 2009

This book presents four different ways of theoretical and practical advances and applications of data mining in different promising areas like Industrialist, Biological, and Social. Twenty six chapters cover different special topics with proposed novel ideas. Each chapter gives an overview of the subjects and some of the chapters have cases with offered data mining solutions. We hope that this book will be a useful aid in showing a right way for the students, researchers and practitioners in their studies.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Yixin Chen (2009). Clustering Parallel Data Streams, Data Mining and Knowledge Discovery in Real Life Applications, Julio Ponce and Adem Karahoca (Ed.), ISBN: 978-3-902613-53-0, InTech, Available from: http://www.intechopen.com/books/data_mining_and_knowledge_discovery_in_real_life_applications/clustering_parallel_data_streams

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen