QUERYING IN RECENT BIASED DATA USING ADAPTIVE STREAM PROCESSING

Muruga Radha Devi D.[1], **Thambidurai P.**[2],

[1]Research Scholar, Sathyabama University, Chennai, India
[2]Department of CSE, PKIET, Karaikal, India
Email: [1]emrdevi@hotmail.com

ABSTRACT

Recently a new class of data-intensive application has become widely used in which the data is modeled not as persistent relations but as *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. As a consequence, there has been a dramatically increasing amount of interest in querying and mining such data which in turn resulted in a large amount of work introducing new methodologies for indexing, classification and approximation of time series. Research in this field has focused on the development of effective transformation techniques, the application of dimensionality reduction methods and the design of efficient indexing schemes. Traditional access methods that continuously update data are considered inappropriate, due to significant update costs. The proposed method called as adaptive stream processing is based on an incremental computation of Discrete wavelet transform which is used as a feature extraction method and efficient technique for similarity query processing using sliding windows In order to prove the efficiency of the proposed method, experiments have been performed for range query and k-nearest neighbor query on real-life data sets. The results have shown that the adaptive stream processing method exhibit consistently better performance in comparison to previously proposed approaches.

Key words: Similarity query processing, data streams, sliding window, feature extraction, adaptive stream processing.

I. INTRODUCTION

A time series is a sequence of real numbers each number representing a value at a time point. In fact, any measurement that changes over time can be represented as a time series. For example the sequence could represent stock or commodity prices, sales exchange rates, weather data and bio-medical measurements. If there is a need for continuous stock monitoring as time progresses, then streaming time series are more appropriate. Streaming time series is a special case of streaming data, which nowadays are considered very important and there is an increasing research interest in the area. Traditional database methods cannot be applied directly to data streams. Therefore, new techniques and algorithms are required in order to guarantee efficient and effective query processing in terms of the CPU time and the number of disk accesses. The most important difficulty that these techniques must address is the continuous change, which poses serious restrictions. Two key aspects for achieving efficiency and effectiveness when managing time series data are representation methods and similarity measures.

A streaming time-series S is a sequence of real values s1; s2; …, where new values are continuously appended as time progresses. Formally a streaming time series S consists of a set of (tuple, timestamp) pairs. The ordering that tuples become available is induced by the timestamps. For example, a temperature sensor which monitors the environmental temperature every five minutes, produces a streaming time-series of temperature values. As another example, consider a car equipped with a Global Positioning System (GPS) device and a communication module, which transmits its position to a server every ten minutes. A streaming time series of two-dimensional points (the x and y coordinates of its position) is produced. Note that, in a streaming time-series data values are ordered with respect to the arrival time. New values are appended at the end of the series. If the sampling rate of the two time series are the same, one can omit the timestamps and consider them as sequences of d-dimensional points. Such a sequence is called the raw representation of time series.

The length of a streaming time series can be very large, since new values are continuously appended. Therefore, the similarity of two time series is expressed by means of the last values of each stream (e.g., 128, 256, 1024 values), using a sliding window approach. Each stream can be represented as a vector in a high-dimensional space. So it is desirable to develop representation techniques that can reduce the dimensionality of time series while still preserving the fundamental characteristics of a particular dataset. Dimensionality reduction techniques (e.g., Discrete Fourier Transform, Discrete wavelet Transform) can be used in order to reduce the number of dimensions, allowing efficient multidimensional access methods to be utilized. However, each vector changes over time since new values are continuously appended. The naive approach is to delete the old feature vector by updating the access method, to re-apply the dimensionality reduction technique to the new vector, and finally, to store the resulting feature vector in the access method. This process is very costly both in Central Processing Unit (CPU) time and number of disk accesses and therefore, it is inappropriate in our case. Since the dimensionality is changing for stream data, it will be beneficial to dynamically index such data to enable the support for efficient query processing. R-tree [6] based index structures have shown to be useful in indexing multi-dimensional data sets, but they are not suitable for indexing data streams since they are designed for the cases where the dimensionality is fixed. So we are using an index structure which can accommodate the changing dimensionality of data objects.

In this paper, we have used adaptive index updating techniques for efficient similarity searching on multiple data streams. This indexing method can be used both as an index and as a summary for the database, which can produce accurate answers to queries in an incremental way. When we are limited to a bounded amount of memory it is not always possible to produce exact answers for data stream queries; however, high-quality approximate answers are often acceptable instead of exact answers. One technique for producing an approximate answer to a data stream query is to evaluate the query not over the entire past history of the data streams, but rather only over sliding windows of recent data from the streams. For example, only data from the last week could be considered in producing query answers, with data older than one week being discarded.

Our Contribution:

- We have developed a novel method in order to process similarity queries in data streams with sliding windows.
- We then applied an index structure to support efficient similarity search for multiple data streams that can accommodate changing dimensionality of data objects.
- Our method is a generalized one to support queries on recent biased data also.
- Performance comparison is done between the proposed method and existing methods like sequential scan and VA$^+$ stream method.

II. BACKGROUND AND RELATED WORK

In this section, research into various similarity query processing methods are first reviewed and then we briefly describe about the traditional dimensionality reduction methods and Scalar Quantization Technique for stream processing.

A. Similarity Query processing

Analysis of time-series data is rooted in the ability to find similar series. Similarity is defined in terms of a distance metric, most often Euclidean distance or relatives of Euclidean distance. Two time sequences of same length are said to be similar if the Euclidean distance is less or equal to a given threshold.

There are two basic types of similarity queries:

- *similarity range query*: given a user time series *Q* and a distance *e*, this query retrieves all time series that are within distance *e* from *Q*.
- *similarity nearest-neighbor query*: given a user time series *Q* and a integer *k*, this query retrieves the *k* series that are closer to *Q*.

The brute force approach to answering these queries is sequential scanning which requires comparing every time series C_i to Q_i. This approach is unrealistic for large datasets. Similarity searching techniques that guarantee no false dismissals are exact, and techniques that do not have guarantee as approximate. Approximate techniques can still be very useful for exploring large databases, when the probability of false dismissal is low.

A time series $C = (c_1, c_{2, \ldots}\, c_n)$ with n data points can be considered as a point in n-dimensional space which suggests that time series could be indexed by multidimensional index structure such as R-tree[6], so we need to perform dimensionality reduction in order to exploit multidimensional index structure to index time series data. In order to guarantee no false dismissals the distance measure in the index space must satisfy the following condition

$$D_{indexspace}(A, B) <= D_{true}(A, B) \quad [1]$$

(i.e.) we can define a distance measurement on the reduced size representations that is guaranteed to be less than or equal to the true distance measured on the raw data. This is known as Contractive Property or I.ower Bounding property. It is this property that allows using representations to index the data with a guarantee of no false dismissals.[4]

The efficient processing of similarity queries requires the addressing of the following important issues:

- the definition of a meaningful distance measure *D* in order to express the similarity between two time series objects.
- the efficient representation of time series data and
- the application of an appropriate indexing scheme in order to quickly discard database objects that cannot contribute to the final answer.

It is evident from the above definitions, that in order to express similarity between two time series objects, a distance measure *D* is required. This distance measure usually ranges between 0 and 1. If two time series *u* and *v* are similar, then the value *D* (*u*, *v*) should be close to 1, whereas if they are dissimilar then *D* (*u*, *v*) should be close to 0. Similarity search can be applied for whole-match queries and subsequence-match queries as well, for static or streaming time series.

Similarity queries in streaming time series have been studied in [11] where whole-match queries are investigated by using the Euclidean distance as the similarity measure. A prediction-based approach is used for query processing. The distances between the query and each data stream are calculated using the predicted values. When the actual values of the query are available, the upper and lower bound of the prediction error are calculated and the candidate set is formed using the predicted distances. Then, false alarms are discarded.

Several alternative distance functions have been proposed in order to allow translation, rotation and scaling invariance[13]. The main shortcoming of the Euclidean distance is that all time series must have equal length, which is a significant restriction. If time series are sampled using different time intervals, then their length will not be the same, and therefore the Euclidean distance cannot be applied. In order to express similarity between time series of different lengths, other more sophisticated distance measures have been proposed. One such distance measure is *Time Warping* (TW) that allows time series to be stretched along the time axis. The time warping distance maps each element of a time series *u* to one or more elements of another time series *v*. B-K, Jagadish and Faloutsos [7] used time warping as distance function and present algorithms for retrieving similar time sequences under this function. Rafiei and Mendelzon [9] propose a set of linear transformations such as moving average, time warping and reversing. These transformations can be used as the basis of similarity queries for time series data. In addition Rafiei[10] propose the method of processing queries that express similarity in terms of multiple transformations instead of a single one.

Representing each time series as an *m*-dimensional vector may result in performance degradation during query processing, due to high computation costs of the distance function. It is preferable to compute the distance as efficiently as possible, towards increased processing performance. To attack this problem, *dimensionality reduction* is applied to the time series, in order to transform them to a more manageable representation.

B. Dimensionality Reduction Methods

Agrawal, Faloutsos and swami [1], first proposed the use of distance preserving transformation for dimensionality reduction. Popular feature extraction techniques are DFT (Discrete Fourier Transform) and DWT (Discrete Wavelet Transform) where the sequence is projected into the frequency domain (DFT) or tiling of the time frequency plane (DWT) [3]. Wavelets are often used as a technique to provide a summary representation of the data. Wavelet coefficients are projections of the given signal (set of data values) onto an orthogonal set of basis vector. Often Haar wavelets are used in databases for their ease of computation. Wavelet coefficients have the desirable property that the signal reconstructed from the top few wavelet coefficients best approximates the original signal. Note that dimensionality reduction is a lossy operation, since after the transformation of the time series some information is lost. This means that the transformed data is an approximation of the original data, and the latter must be retained in order to be available for reference. The original and the transformed data are used for query processing by means of the *filter-refinement* processing technique.

C. Stream processing using Sliding Windows

We define streaming database as a collection of multiple data streams, each of which arrives sequentially and describes an underlying signal. For example, the data feeds from a sensor network form a streaming database. The dimensionality of each data stream is always increasing in this case. Hence, theoretically the amount of data stored, if stored at all, in a streaming database tends to be infinite. This leaves us with a challenge of trying to get accurate query results from a huge database with time constraints. A set of algorithms for stream processing works on the recent past of data streams by applying a sliding window on the data stream [2]. In this way, only the last W values of each streaming time series is considered for query processing, whereas older values are considered obsolete and they are not taken into account. For example, streams that are non-similar for a window of length W (left), may be similar if the window is shifted in the time axis (right). The number of samples each time series may range from a few to hundreds or thousands.

D. Scalar Quantization Technique

Scalar quantization technique is a way to quantize each dimension independently so that a summary of the database is efficiently captured. It also serves as an index structure for efficient point, range and k nearest neighbor queries.

Indexing Based on Vector Approximation

Since conventional partitioning index methods, e.g. R-trees, grid files and their variants, suffer from dimensional curse, VA-file was used as an approach to overcome the curse and supports efficient similarity search in high-dimensional spaces. The VA-file is actually a filter-based approach of synopsis files. It divides the data space into 2^b rectangular cells where b is the total number of bits specified by the user . Each dimension is allocated a number of bits, which are used to divide it into equally populated intervals on that dimension. Each cell has a bit representation of length b which approximates the data points that fall into this cell. The VA-file itself is simply an array of these bit vector approximations based on the quantization of the original feature vectors. VA^+ file [5] is used to target non-uniform data sets, and can lead to more efficient searching. Weber et al. [12] have developed a quantitative analysis and performance study of similarity search techniques for high dimensional data sets.

VA-Stream for Indexing Streaming Database

VA-file and VA^+-file are targeted towards traditional databases, in which the dimensionality of data objects is fixed. In order to handle dynamic streaming databases, the approaches should be customized for streaming databases. We call the customized approaches as VA-Stream and VA^+-Stream, a way to generate dynamic synopsis or summarization for streaming databases with dynamic updates. Since VA-Stream and VA^+-Stream are capable of taking a snapshot of any moment of the dynamic streaming databases, a preliminary analysis can be made on the current snapshot so that the data set can be preprocessed and a better performance for efficient similarity searching can be achieved. The algorithm used to build a new VA-file in order to capture the up-to-date snapshot of the streaming databases is called VA-Stream [8].

III. ADAPTIVE STREAM PROCESSING

A streaming time series is a sequence of data values where new values are continuously appended as time progresses. Let X_{-3}, X_{-2}, X_{-1}, X_0 be a stream of stock data, where X_0 means today's data, X_{-1} means yesterday's data, and so on. The recent – biased data stream will take the following form:

$$\ldots\ldots+d(1-d)^3 X_{-3}+d(1-d)^2 X_{-2}+d(1-d) X_{-1}+dX_0$$

We look at continuous or streaming queries with either a sliding window or an infinite window. For queries with a fixed-size sliding window, the size of the synopsis stays the same, and the accuracy of this synopsis does not suffer from the evolving dimensionality. The approach for queries with an infinite window is based on an reasonable assumption that the data streams are aging. That's actually what happens in real world, where people put more emphasis on more recent activities. In the context of recent biased analysis the users are more interested in the current data than in the past data, and bigger weights will be assigned to more recent dimensions of the data. The challenge is to maintain these biased approximations continuously as new data arrives in an online manner. So a system which maintains better approximations for the recent data is useful.

A. Adaptive Stream Processing

A stream is denoted by the symbol S_x and a finite time series by the symbol S [i : j], where i is the first time instance of the time series and j is the last. The number of values of a time series is therefore $j-i+1$ and corresponds to a window of length W. S(i) is the i^{th} value of the time series. In our study, the Euclidean distance between two finite time series is used as the similarity measure. Let us assume the existence of n streaming time series, each updated over time. To determine similar streaming time series, we use only the last W values of each one and update these values when a new value becomes available. Given a query streaming time series the challenge is to determine similar time series as the data evolve with time.

The naive approach is the Sequential Scan (SS). In each update, the distances between the query streaming time series and each data streaming time series are computed. Then similar streaming time series are reported. The streaming environment poses new challenges to the applications as unbounded memory requisites, high input rates and fast response times. Therefore more sophisticated approaches are necessary to speed up the similarity process. The similarity measure is applied on the extracted features. Moreover, an indexing method is used to prune time series and therefore to avoid distance computations.

Imposing sliding windows on data streams is a natural method for approximation that has several attractive properties. If one is trying in real-time to make sense of network traffic patterns, or phone call or transaction records, or scientific sensor data, then in general insights based on the recent past will be more informative and useful than insights based on stale data. For example, queries which tracked traffic on the network backbone, would likely be applied not to all traffic over all time, but rather to traffic in the recent past. Moreover, in most cases users would expect fast response time for queries. This makes it necessary to develop an effective index structure for streaming database with very efficient update cost, so that query results can be obtained in a tolerable amount of time.

The steps involved in the adaptive stream processing are:

(a) The last W values of each stream are stored in disk.

(b) The DWT coefficients of each stream are inserted into the index.

(c) When a value becomes available the window of the stream is updated, the features of the stream are extracted incrementally and the new DWT coefficients replace the old ones.

(d) Then the adaptive update policy decides if the index needs updation or not.

(e) The query is applied to the index to retrieve candidates streaming time series.

(f) Then in a post processing step the actual distances are computed to discard false alarms.

To satisfy the limitations posed by the streaming environment, an incremental computation of DWT is used, and an adaptive update policy is used in order to avoid the degradation of the system due to the usage of the index structure and the high number of updates.

B. Incremental DWT computation

The DWT is used as the feature extraction method, which preserves the Euclidean distance between two sequences. There is a trade-off between the number of the DWT coefficients and the approximation of the distance in the time domain. If more DWT coefficients are used then the number of candidates is reduced during the query processing, thus the query procedure speeds up. Normally, every time a new value for a stream arrives, the DWT vector must be recalculated by using the last W values of the stream.

Let X be a streaming time series with values

X(0); X(1); . . . ; $X(W-1)$ and length W.

Moreover, let $DWT_0(X)$; $DWT_1(X)$; $DWT_{w-1}(X)$ denote the DWT coefficients of X. If a new value for this stream arrives, we get the sequence

T(1), T(2); . . . ; T(W) , where X(i) = T(i)

for $1 \leq i \leq W-1$ and T(W) is the new value.

C. Adaptive Update policy

Since the number of streams may be quite large, the use of an index structure is desirable to avoid the computation of the distance between the query and all the time series. If we update the index every time a new value becomes available, the overhead may be prohibitive due to additional page accesses. To avoid continuous deletions and insertions, we use an *ada--ptive update policy*. The proposed approach is an incremental way to update the index to reflect the changes in the databases, and it can eliminate the need to rebuild the whole index structure from scratch. Hence it enables faster query response time.

A parameter Δ_u is used to control the updates. If the distance between the new and the old DWT vector exceeds the value of parameter Δ_u, then the index is updated. Otherwise, no update is performed. This technique leads to considerable savings in CPU and I/O time. The last recorded DWT vector is stored in the last disk page of every streaming time series, to become available when a new value arrives.

Let X be a streaming time series. The last W values form a sequence denoted by X1[N - W + 1 : N] where N is the position of the last value of the time series. When a new value for X_1 is available, a new sequence X2 [N - W + 2 : N+1] is formed. Assume further that DWT(X1) is the last recorded DWT sequence corresponding to, X1 [N - W + 1 : N] and DWT(X2) is the DWT sequence corresponding to X2 [N - W + 2 : N+1], which is computed incrementally using the DWT (X1).

If D_E(DWT(X1); DWT(X2)) $\leq \Delta_u$, then DWT(X2) is stored as the most recent DWT (replaces DWT (X1)) but it is not inserted into the index. Assume that another value for the same stream arrives. Let X3 [N-W + 3 : N + 2] be the new time series and DWT (X3) the DWT of this sequence, which is computed incrementally using DWT (X2). DWT (X3) replaces DWT (X2) as the most recent DWT. If D_E (DWT (X3); DWT(X1)) $\leq \Delta_u$, no update is performed in the index. On the other hand, if D_E (DWT(X3); DWT(X1)) > Δ_u then DWT (X3) replaces DWT (X1) in the index, so an update is performed.

In summary, we need both the last recorded DWT vector and the previously calculated DWT vector. The first is used to decide whether an update will occur or not, and the second is used for the incremental computation of the new DWT vector.

D. Query Processing in Data Streams

The overall procedure for similarity queries in streaming series includes the following steps:

(a) feature extraction is applied on time series,

(b) the extracted features are inserted in an index structure,

(c) feature extraction is applied on the query time series,

(d) the index is used to retrieve candidate time series respecting a user-defined threshold and

(e) the distances between query and candidates time series are computed in a post processing step to discard false alarms.

Nearest-neighbor query processing

Assume that we have the DWT vector of the query stream DWT(S_q), the MBR_{LR} that is formed by the last recorded DWT vectors and a stream S_x that belongs to the MBR_{LR}. Moreover, assume that we have the last recorded DWT vector DWT$(S_x)_{LR}$ of stream S_x and the current DWT vector DWT (S_x) of stream S_x. The k^{th} nearest neighbor distance is d_k.

D_E (DWT (S_q), DWT (S_x)) $\leq d_k$

D_E (DWT (S_q), DWT (S_x)) + $\Delta_u \leq d_k + \Delta_u$

D_E (DWT (S_q); DWT (S_x)) + D_E (DWT (S_x); DWT $(S_x)_{LR}$) $\leq$ dk + Δ_u

D_E (DWT (S_q); DWT (S_x) LR) $\leq d_k + \Delta_u$;

(triangular inequality)

MinDist (DWT(S_q); MBR_{LR}) $\leq d_k + \Delta_u$

This implies that if the current DWT vector of a stream is closer to the query point than the k^{th} neighbor, then the corresponding MBR_{LR} will be inserted in the index. We can visit the first k streams and compute their real distances from the query. The maximum distance can be used for the initialization of d_k, instead of one.

Range query processing

In order for similarity range queries to produce the correct results, the user-defined distance e must be expanded by a value of Δ_u, which is the maximum Δ_u value seen so far. In this way, we guarantee that no false dismissals occur. Assume that we have the DWT vector of the query stream DWT (S_q) the MBR_{LR} that is formed by the last recorded DWT vectors and a stream S_x that belongs to the MBR_{LR}. Moreover, assume that we have the last recorded DWT vector DWT $(S_x)_{LR}$ of stream S_x and the current DWT vector DWT (S_x) of stream S_x.

$$D_E\ (DWT\ (S_q),\ DWT\ (S_x)) + \Delta_u \le e + \Delta_u$$

$$D_E\ (DWT\ (S_q);\ DWT\ (S_x)) + D_E\ (DWT\ (S_x);\ DWT\ (S_x)_{LR}) \le e + \Delta_u$$

$$D_E\ (DWT\ (S_q);\ DWT\ (S_x)\ _{LR}) \le e + \Delta_u$$

(triangular inequality)

$$MinDist\ (DWT\ (S_q);\ MBR_{LR}) \le e + \Delta_u$$

This implies that if the current DWT vector of a stream overlaps the query region, then the corresponding MBR_{LR} (a MBR that is formed by the last recorded DWT coefficients) will overlap the extended query region.

IV. EXPERIMENTAL STUDY

An important operation in streaming time series is to determine similar time series with respect to a query series. Similarity is expressed by means of the last w values of the streams. In this paper we have analyzed similarity range queries and nearest neighbor queries in such an environment The design of our experiments aims to show the advantage of the proposed approach in terms of both query response time and index building time, total CPU cost and candidate ratio.

A. Dataset

The techniques proposed in this paper are for high dimensional streaming data sets, therefore we chose our real-life data sets to have 360-day stock price movements of 500 companies, i.e., 500 data points with dimensionality 360. For e.g., in the case of *Stock data*, each dimension corresponds to daily stock prices of all companies.

We have used VA$^+$ stream and sequential scanning as competitor for our method because of the infeasibility of the well known techniques for stream data and the well known dimensionality curse which makes other choices like R-tree and its variants out of question for range and nearest neighbor search. It can be used only in cases where a fraction of the stream is updated in each time instance. Both range and k-NN queries are considered. We have studied the performance of the methods by varying several of the most important parameters such as query distance e in range queries, the value k for k-NN queries, the length of the sliding window, the number of DWT coefficients etc.,

The default values for the parameters are: the query distance 'e' that has been chosen so that 1% of the streams to be in the answer. The desirable update ratio is 0.1%. Thus only .1% of the streams will be actually updated in each time instance. The sliding window size is 256. A group of experiments was set up to evaluate the performance of the approximation generated by our approach for a snapshot of streaming databases.

B. Performance of range query and nearest neighbor queries

In this section we see the experimental results obtained from range queries where all streams are updated in each time instance. The performance of the three methods with respect to query distance 'e' is shown in figure 1. CPU cost of VA$^+$ stream is less than that of adaptive stream processing when number of queries is low. As e increases, the difference between the two methods is decreased because the number of streams that are contained in the final answer increases rapidly. Sequential scan does not require any index updates.

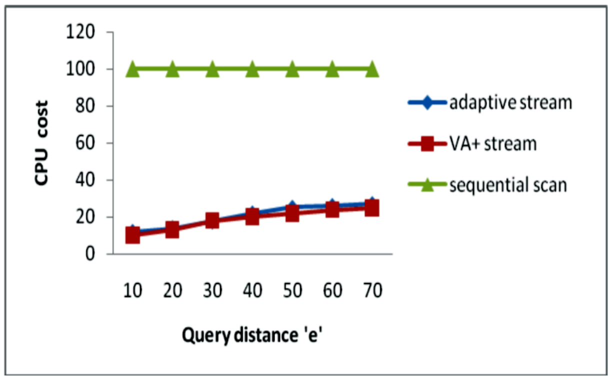

Fig. 1. Performance by increasing the query distance 'e'

The number of DWT coefficients has an important impact on the performance. As the number of coefficients increases, the distance preservation is improved and therefore less false alarms are introduced. The figure 2 shows the hit ratio with respect to the number of DWT coefficients for the stocks data set. Thus it is better to use an adequate number of DWT coefficients sacrificing low CPU cost. Achieving good hit ratio is important because hit ratio impacts the query efficiency and thus the overall method.

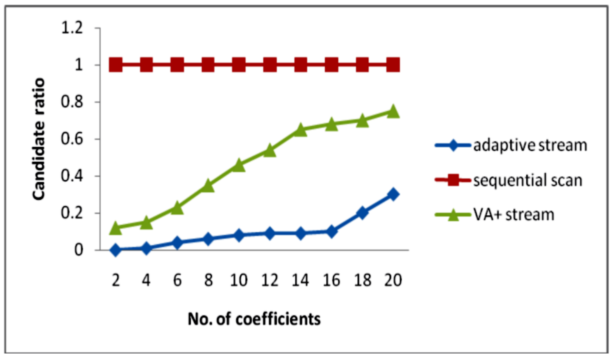

Fig. 2. Hit ratio by increasing the no. of DWT coefficients

Figure 3 shows the performance of the three methods with respect to k in k-NN query processing. From the figure it is clear that the three methods are having similar performance as in range queries.

To show the impact of window size on the performance of the approaches under test, window sizes were chosen to be 64, 128, 256, 512 and 1024 for *Stock data* and experiments were set up to process k-NN queries for the streaming database at any time positions after new dimension comes. Figure 4 shows the impact of varying sliding window size over total CPU time.

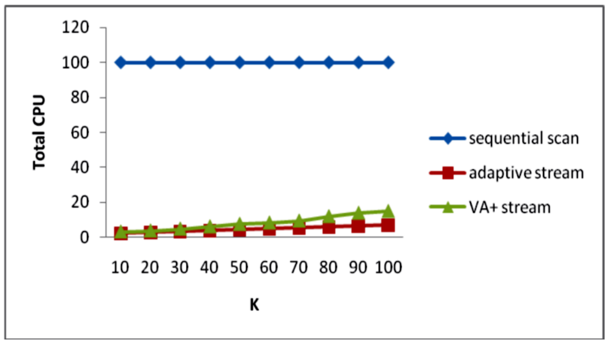

Fig. 3. Performance of the three methods with respect to different values of k

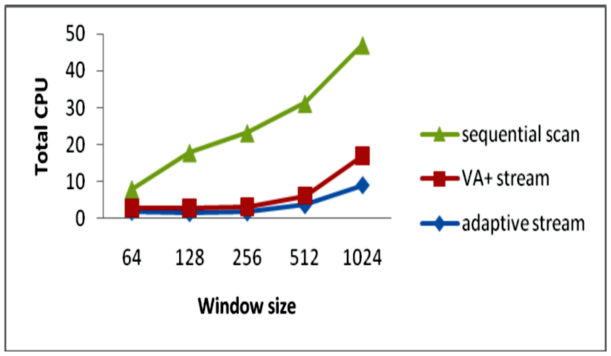

Fig. 4. Performance of the three methods by varying window size

Two metrics were used to evaluate how the generated approximation can support range and k-NN queries: vector selectivity and page ratio. Vector selectivity was used to measure how many vectors have been actually visited in order to find the k nearest neighbors of the query. Since vectors actually share pages, the vector selectivity does not exactly reflect the paging activity and query response during the similarity search. Hence, page ratio was adopted to measure the number of pages visited as a percentage of the number of pages necessary. In contrast, the vector selectivity is always 100% for sequential scan since it needs to visit all the vectors in order to find the k nearest neighbors.

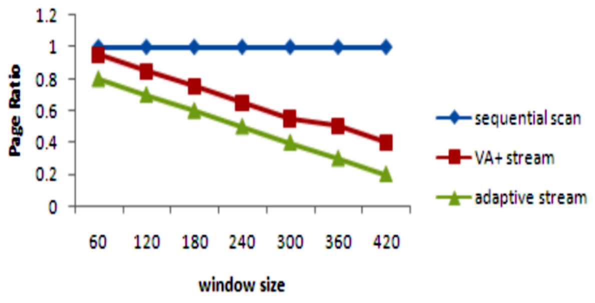

Fig. 5. (a) Page ratio Vs sliding window for stock data

The following two types of metrics were also used here for performance evaluation of k-NN queries: *average index building time* and *average query response time*. In order to get the average index building time and the average query response time, for each different window size we chose, we processed 100-NN queries at each time position after the new dimension arrived. We recorded the index building time and the average query response time over 100 queries

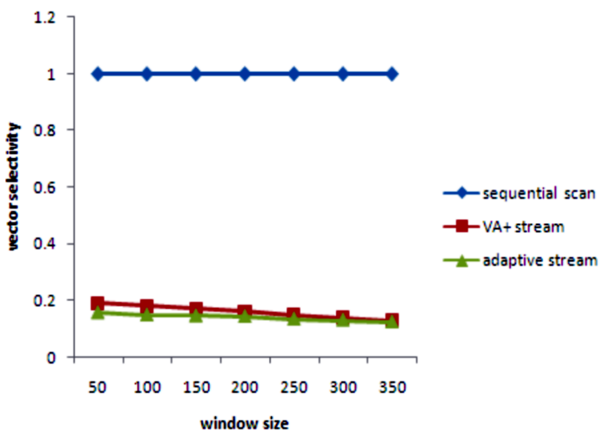

Fig. 5. (b) Vector selectivity Vs sliding window for stock data

at each time position and then computed the average index building time and the average query response time over 100 queries.

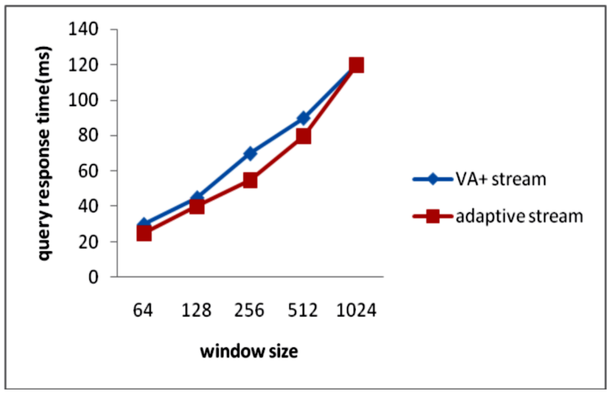

Fig. 6 (a) Query response time of two methods, for stock time series data.

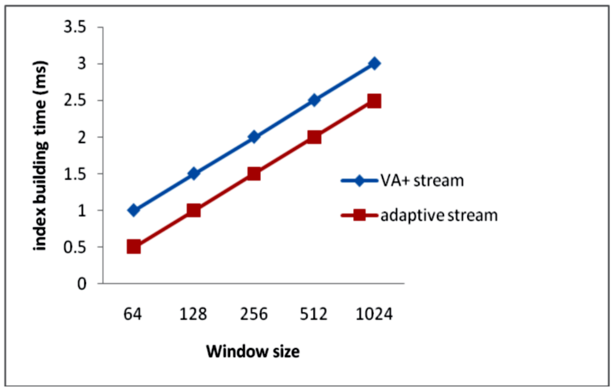

Fig. 6 (b) Compares the index building time of two methods, for stock time series data.

The index building time does not vary too much for adaptive stream and VA$^+$ Stream technique, since it is an incremental method and works almost on only one dimension.

V. CONCLUSION

We have examined similarity range queries and nearest neighbor queries using adaptive stream processing method. We have used sequential scanning and VA$^+$ stream methods as the competitor for the proposed method. We have studied the performance of the methods by varying several of the most important parameters such as query distance e in range queries, the value k for k-NN queries. Our performance evaluation establishes that the proposed technique can be used to build the index structure for streaming database in a much shorter time than available approaches. When the number of dimensions are large, our method can work both as an update-efficient index and as a dynamic summary of stream data. An advantage of adaptive stream processing is that it can handle different window sizes. A sliding window of length w is defined to capture the last w values of each streaming time series. It is expected that CPU cost for the proposed method will not be affected significantly, since the number of coefficients remains fixed. Results have shown that significant improvement is achieved in comparison to the existing approaches.

REFERENCES

[1] Agrawal, R. Faloutsos, C & Swami. A. 1993, Efficient similarity search in sequence databases, *Proceedings of the 4th* Conference on Foundations Of Data organization and Algorithms.

[2] B. Babcock, M. Datar R. Motwani, L. Callaghan, Maintaining variance and k-medians over data stream windows, *in Proceedings of the symposium on Principles of Database Systems*, 2003, pp 234 – 243.

[3] K.P. Chan and A.W. Fu, 1999, Efficient Time Series Matching by Wavelets, *Proc. Int'l Conf on Data Eng.* (ICDE 99).

[4] Eamon Keogh, Kaushik.Chakrabti, Michael Pazzani and Sharad Mehrota, 2001, Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases, *Knowledge and Information Systems, vol 3, pp 263-286.*

[5] H. Ferhatosmanoglu, E. Tuncel, D. Agarwal and El Abbadi, Vector Approximation based indexing for non-uniform high dimensional data sets. *In Proceddings of the 9th ACM Int. Conf. on Information and Knowledge Management, pages 202-209, McLean, Virginia, November 2000.*

[6] A. Guttman, R-Trees: A dynamic index structure for spatial searching, 1984, *In Proceedings of ACM SIGMOD Int. Conf. Management of Data*, Boston, USA, pp 47-57.

[7] B-K, Yi, H.V. Jagadish and C. Faloutsos, Efficient Retrieval of similar time sequences under time warping, in *Proc. Int. Conf. Data Engineering*, Orlando, Florida, USA 1998, pp 201-208.

[8] Maria Kontaki, Apostolos N Papadopoulos, Yannis Manolopoulos, Adaptive Similarity Search in streaing time series with sliding windows, *Data and Knowledge Engineering , 2007, pp 478 – 502.*

[9] D. Rafiei and A.O. Mendelzon, Similarity based queries for time series data, in *Proceedings of ACM SIGMOD Int. Conf.Management of Data, Arizona, USA 1997, pp 13-25.*

[10] D. Rafiei, On Similarity based queries for time series data, in *Proceedings of Int. Conf Data, Engineering, Sydney, Australia, 1997, pp. 410-417.*

[11] Wang and Gao L, Continually evaluation similarity based pattern queries in a streaming time series, in Proceedings of the Madison, WI, 2002.

[12] R. Weber, H.J. Schek and S. Blott, A quantitative analysis and performance study for similarity search methods in high-dimensional spaces, *In Proceedings of Int. Conf. on very large Data bases, pp 194 – 205, New York, August 1998.*

[13] Xiaoyue Wang, Eamonn Keogh, Peter Scheuermann, Querying and Mining of Time Series Data: Experimental comparison of Representations and Distance Measures.

D. Muruga Radha Devi received Bachelor's degree in Computer Science and Engineering from Bharathidasan University in 1994, Master's Degree in Computer Science and Engineering from University of Madras in 2001. Her ***research*** interests include time series analysis, efficient information retrieval and Data mining