StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time *

Yunyue Zhu,Dennis Shasha

Courant Institute of Mathematical Sciences
Department of Computer Science
New York University
{yunyue,shasha}@cs.nyu.edu

Abstract

Consider the problem of monitoring tens of thousands of time series data streams in an online fashion and making decisions based on them. In addition to single stream statistics such as average and standard deviation, we also want to find high correlations among all pairs of streams. A stock market trader might use such a tool to spot arbitrage opportunities. This paper proposes efficient methods for solving this problem based on Discrete Fourier Transforms and a three level time interval hierarchy. Extensive experiments on synthetic data and real world financial trading data show that our algorithm beats the direct computation approach by several orders of magnitude. It also improves on previous Fourier Transform approaches by allowing the efficient computation of time-delayed correlation over any size sliding window and any time delay. Correlation also lends itself to an efficient grid-based data structure. The result is the first algorithm that we know of to compute correlations over thousands of data streams in real time. The algorithm is incremental, has fixed response time, and can monitor the pairwise correlations of 10,000 streams on a single PC. The algorithm is embarrassingly parallelizable.

1 Introduction

Many applications consist of multiple data streams. For example,

- In mission operations for NASA's Space Shuttle, approximately 20,000 sensors are telemetered once per second to Mission Control at Johnson Space Center, Houston[16].
- There are about 50,000 securities trading in the United States, and every second up to 100,000 quotes and trades (ticks) are generated.

Unfortunately it is difficult to process such data in set-oriented data management systems, although object-relational time series extensions have begun to fill the gap [21]. For the performance to be sufficiently good however, "Data Stream Management Systems" (DSMSs) [3], whatever their logical model, should exploit the following characteristics of the application:

- Updates are through insertions of new elements (with relatively rare corrections of older data).
- Queries (moving averages, standard deviations, and correlation) treat the data as sequences not sets.
- Since a full stream is never materialized, queries treat the data as a never-ending data stream.
- One pass algorithms are desirable because the data is vast.
- Interpretation is mostly qualitative, so sacrificing accuracy for speed is acceptable.

This paper presents the algorithms and architecture of StatStream, a data stream monitoring system. The system computes a variety of single and multiple stream statistics in one pass with constant time (per input) and bounded memory. To show its use for one practical application, we include most of the statistics that a securities trader might be interested in. The algorithms, however, are applicable to other disciplines,

*Work supported in part by U.S. NSF grants IIS-9988345 and N2010:0115586.

NYU computer science department technicl report TR2002-827, June 2002. An abbreviated version of this report will appear in *Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002*

such as sensor data processing and medicine. The algorithms themselves are a synthesis of existing techniques and a few ideas of our own. We divide our contributions into functional and algorithmic. Our functional contributions are:

1. We compute multi-stream statistics such as synchronous as well as time-delayed correlation and vector inner-product in a continuous online fashion. This means that if a statistic holds at time t, that statistic will be reported at time $t + v$, where v is a constant independent of the size and duration of the stream.

2. For any pair of streams, each pair-wise statistic is computed in an incremental fashion and requires constant time per update. This is done using a Discrete Fourier Transform approximation.

3. The approximation has a small error under natural assumptions.

4. Even when we monitor the data streams over sliding windows, no revisiting of the expiring data streams is needed.

5. The net result is that on a Pentium 4 PC, we can handle 10,000 streams each producing data every second with a delay window v of only 2 minutes.

Our algorithmic contributions mainly have to do with correlation statistics. First, we distinguish three time periods:

- timepoint – the smallest unit of time over which the system collects data, e.g. second.

- basic window – a consecutive subsequence of timepoints over which the system maintains a digest incrementally, e.g., a few minutes.

- sliding window – a user-defined consecutive subsequence of basic windows over which the user wants statistics, e.g. an hour. The user might ask, "which pairs of stocks were correlated with a value of over 0.9 for the last hour?"

The use of the intermediate time interval that we call basic window yields three advantages:

1. Results of user queries need not be delayed more than the basic window time. In our example, the user will be told about correlations between 2 PM and 3 PM by 3:02 PM and correlations between 2:02 PM and 3:02 PM by 3:04 PM.

2. Maintaining stream digests based on the basic window allows the computation of correlations over windows of arbitrary size, not necessarily a multiple of basic window size, as well as time-delayed correlations with high accuracy.

3. A short basic window give fast response time but fewer time series can then be handled.

A second algorithmic contribution is the grid structure, each of whose cells stores the hash function value of a stream. The structure itself is unoriginal but the high efficiency we obtain from it is due to the fact that we are measuring correlation and have done the time decomposition mentioned above.

The remainder of this paper will be organized as follows. The data we consider and statistics we produce are presented in Section 2. Section 3 presents our algorithms for monitoring high speed time series data streams. Section 4 discusses the system StatStream. Section 5 presents our experimental results. Section 6 puts our work in the context of related work.

2 Data And Queries

2.1 Time Series Data Streams

We consider data entering as a time ordered series of triples (streamID, timepoint, value). Each stream consists of all those triples having the same streamID. (In finance, a streamID may be a stock, for example.) The streams are synchronized.

Each stream has a new value available at every periodic time interval, e.g. every second. We call the interval index the **timepoint**. For example, if the periodic time interval is a second and the current timepoint for all the streams is i, after one second, all the streams will have a new value with timepoint $i+1$. (Note that if a stream has no value at a timepoint, a value will be assigned to that timepoint based on interpolation. If there are several values during a timepoint, then a summary value will be assigned to that timepoint.)

Let s_i or $s[i]$ denote the value of stream s at timepoint i. $s[i..j]$ denotes the subsequence of stream s from timepoints i through j inclusive. s^i denotes a stream with streamID i. Also we use t to denote the latest timepoint, i.e., now. The statistics we will monitor will be denoted $stat(s_j^{i_1}, s_j^{i_2}, ..., s_j^{i_k}, j \in [p, q])$, where the interval $[p, q]$ is a window of interest. We will discuss the meaning of windows in the next section.

2.2 Temporal Spans

In the spirit of the work in [10, 11], we generalize the three kinds of temporal spans for which the statistics of time series are calculated.

1. **Landmark windows:** In this temporal span, statistics are computed based on the values between a specific timepoint called landmark and the present. $stat(s, landmark(k))$ will be computed on the subsequence of time series $s[i], i \geq k$. An unrestricted window is a special case when $k = 1$. For an unrestricted window the statistics are based on all the available data.

2. **Sliding windows:** In financial applications, at least, a sliding window model is more appropriate for data streams. Given the length of the sliding window w and the current timepoint t, $stat(s, sliding(w))$ will be computed in the subsequence $s[t-w+1..t]$.

3. **Damped window model:** In this model, recent sliding windows are more important than previous ones. For example, in the computation of a moving average, a sliding window model will compute the average as $avg = \frac{\sum_{i=t-w+1}^{t} s_i}{w}$. By contrast, in a damped window model the weights of data decrease exponentially into the past. For example, a moving average in a damped window model can be computed as follows:

$$avg_{new} = avg_{old} * p + s_t * (1-p), 0 < p < 1$$

Other statistics in a damped window model can be defined similarly.

In this paper, we will focus on the sliding window model, because it is the one used most often and is the most general.

2.3 Statistics To Monitor

Consider the stream $s_i, i = 1, ..., w$. The statistics we will monitor are

1. Single stream statistics, such as average, standard deviation, best fit slope. These are straightforward.

2. Correlation coefficients

$$corr(s,r) = \frac{\frac{1}{w}\sum_{i=1}^{w} s_i r_i - \overline{s}\ \overline{r}}{\sqrt{\sum_{i=1}^{w}(s_i - \overline{s})^2}\sqrt{\sum_{i=1}^{w}(r_i - \overline{r})^2}}$$

3. Autocorrelation: the correlation of the series with itself at an earlier time.

4. Beta: the sensitivity of the values of a stream s to the values of another stream r (or weighted collection of streams). For example, in financial applications the beta measures the risk of a stock. A stock with a beta of 1.5 to the market index experiences 50 percent more movement in price than the market.

$$beta(s,r) = \frac{\frac{1}{w}\sum_{i=1}^{w} s_i r_i - \overline{s}\ \overline{r}}{\sum_{i=1}^{w}(r_i - \overline{r})^2}$$

3 Statistics Over Sliding Windows

To compute the statistics over a sliding window, we will maintain a synopsis data structure for the stream to compute the statistics rapidly. To start, our framework subdivides the sliding windows equally into shorter windows, which we call **basic windows**, in order to facilitate the efficient elimination of old data and the incorporation of new data. We keep digests for both basic windows and sliding windows. For example, the running sum of the time series values within a basic window and the running sum within an entire sliding window belong to the two kinds of digests respectively. Figure 1 shows the relation between sliding windows and basic windows.

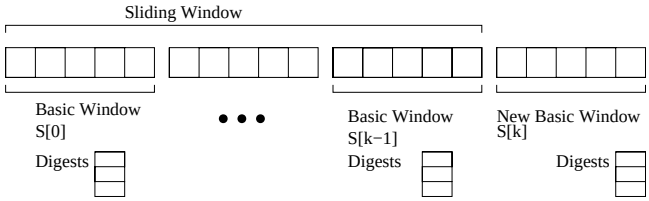

Figure 1: Sliding windows and basic windows

Table 1: Symbols

w	the size of a sliding window
b	the size of a basic window
k	the number of basic windows within a sliding window
n	the number of DFT coefficients used as digests
N_s	the number of data streams

Let the data within a sliding window be $s[t-w+1..t]$. Suppose $w = kb$, where b is the length of a basic window and k is the number of basic windows within a sliding window. Let $S[0], S[1], ..., S[k-1]$ denote a sequence of basic windows, where $S[i] = s[(t-w)+ib+1..(t-w)+(i+1)b]$. $S[k]$ will be the new basic window and $S[0]$ is the expiring basic window. The j-th value in the basic window $S[i]$ is $S[i;j]$. Table 1 defines the symbols that are used in the rest of the paper.

The size of the basic window is important because it must be possible to report all statistics for basic window i to the user before basic window $i+1$ completes (at which point it will be necessary to begin computing the statistics for window $i+1$).

3.1 Single Stream Statistics

The single stream statistics such as moving average and moving standard deviation are computationally inexpensive. In this section, we discuss moving averages just to demonstrate the concept of maintaining digest information based on basic windows. Obviously, the information to be maintained for the moving average is $\sum(s[t-w+1..t])$. For each basic window $S[i]$, we maintain the digest $\sum(S[i]) = \sum_{j=1}^{b} S[i;j]$. After b new data points from the stream become available, we compute the sum over the new basic window $S[k]$. The sum over the sliding window is updated as follows: $\sum_{new}(s) = \sum_{old}(s) + \sum S[k] - \sum S[0]$.

3.2 Correlation Statistics

Correlation statistics are important in many applications. For example, Pairs Trading, also known as the correlation trading strategy, is widely employed by major Wall Street firms. This strategy focuses on trading pairs of equities that are correlated. The correlation between two streams (stocks) is affected by some factors that are not known a priori. Any pair of streams could be correlated at some time. Much effort has been made to find such correlations in order to enjoy arbitrage profits. These applications imply that the ability to spot correlations among a large number of streams in real time will provide competitive advantages. To make our task even more challenging, such correlations change over time and we have to update the moving correlations frequently.

The efficient identification of highly correlated streams potentially requires the computation of all pairwise correlations and could be proportional to the total number of timepoints in each sliding window times all pairs of streams. We make this computation more efficient by (1) using a discrete fourier transform of basic windows to compute the correlation of stream pairs approximately; (2) using a grid data structure to avoid the approximate computation for most pairs.

We start by a quick review of DFT. We will explain how to compute the vector inner-product when the two series are aligned in basic windows. This approach is extended to the general case when the basic windows are not aligned. Then we will show our approach for reporting the highly correlated stream pairs in an online fashion.

3.3 Review of the Discrete Fourier Transform

We will follow the convention in [2]. The Discrete Fourier Transform of a time sequence $x = x_0, x_1, ..., x_{w-1}$ is a sequence $X = X_0, X_1, ..., X_{w-1} = DFT(x)$ of complex numbers given by

$$X_F = \frac{1}{\sqrt{w}} \sum_{i=0}^{w-1} x_i e^{-j2\pi Fi/w} \qquad F = 0, 1, ..., w-1$$

where $j = \sqrt{-1}$. The inverse Fourier transform of X is given by

$$x_i = \frac{1}{\sqrt{w}} \sum_{F=0}^{w-1} X_F e^{j2\pi Fi/w} \qquad i = 0, 1, ..., w-1$$

The following properties of DFT can be found in any textbook on DFT.

- The DFT preserves the Euclidean distance between two sequence x and y.

$$d(x, y) = d(X, Y)$$

- (Symmetry Property) If x is a real sequence, then

$$X(i) = X^*(w - i), \qquad i = 1, 2, ..., w-1$$

X^* is the complex conjugate of X.

Since for most real time series the first few DFT coefficients contain most of the energy($\sum x_i^2$), we would then expect those coefficients to capture the raw shape of the time series [2, 9]. For example, the energy spectrum for the *random walk* series, which models stock movements, declines with a power of 2 with increasing coefficients. And for *black noise*, which successfully models series like the water level of a river as it varies over time, the energy spectrum declines even faster with increasing number of coefficients.

3.4 Inner-product With Aligned Windows

The correlation and beta can be computed from the vector inner-product. The vector inner-product for two series $x = (x_1, ..., x_w), y = (y_1, ..., y_w)$, denoted as $\psi(x, y)$, is just the sum of the products of their corresponding elements:

$$\psi(x, y) = \sum_{i=1}^{w} x_i y_i$$

Given two series s^x and s^y, when the two series are aligned,

$$\psi(s^x, s^y) = \sum_{i=1}^{k} \psi(S^x[i], S^y[i])$$

So, we must explain how to compute the inner-product of two basic windows: $x_1, x_2, ...x_b$ and $y_1, y_2, ...y_b$.

Let $f : f_1(x), f_2(x), ...$ be a family of continuous functions. We approximate the time series in each basic window, $S^x[i] = x_1, x_2, ...x_b$ and $S^y[i] = y_1, y_2, ...y_b$, with a function family f. (We will give specific examples later.)

$$x_i \approx \sum_{m=0}^{n-1} c_m^x f_m(i), y_i \approx \sum_{m=0}^{n-1} c_m^y f_m(i)$$

$$i = 1, ..., b$$

$c_m^x, c_m^y, m = 0, ..., n-1$ are n coefficients to approximate the time series with the function family f.

The inner-product of the two basic windows is therefore

$$\sum_{i=1}^{b} x_i y_i \approx \sum_{i=1}^{b} \Big(\sum_{m=0}^{n-1} c_m^x f_m(i) \sum_{p=0}^{n-1} c_p^y f_p(i) \Big)$$

$$= \sum_{m=0}^{n-1} \sum_{p=0}^{n-1} c_m^x c_p^y \Big(\sum_{i=1}^{b} f_m(i) f_p(i) \Big)$$

$$= \sum_{m=0}^{n-1} \sum_{p=0}^{n-1} c_m^x c_p^y W(m,p)$$

where $W(m,p) = \sum_{i=1}^{b} f_m(i) f_p(i)$ can be precomputed. If the function family f is orthogonal, we have

$$W(m,p) = \begin{cases} 0 & m \neq p \\ V(m) \neq 0 & m = p \end{cases}$$

Thus,

$$\sum_{i=1}^{b} x_i y_i \approx \sum_{m=0}^{n-1} c_m^x c_m^y V(m)$$

With this curve fitting technique, we reduce the space and time required to compute inner-products from b to n. It should also be noted that besides data compression, the curve fitting approach can be used to fill in missing data. This works naturally for computing correlations among streams with missing data at some timepoints.

The sine-cosine function families have the right properties. We perform the Discrete Fourier Transforms for the time series over the basic windows, enabling a constant time computation of coefficients for each basic window. Following the observations in [2], we can obtain a good approximation for the series with only the first n DFT coefficients,

$$x_i \approx \frac{1}{\sqrt{b}} \sum_{F=0}^{n-1} X_F e^{j2\pi Fi/b} \qquad i = 1, 2, ..., b$$

3.5 Inner-product With Unaligned Windows

A much harder problem is to compute correlations with time lags. The time series will not necessarily be aligned at their basic windows. However, the digests we keep are enough to compute such correlations.

Without loss of generality, we will show the computation of the n-approximate lagged inner-product of two streams with time lags less than the size of a basic window. Given two such series, $s^x = s_1^x, ..., s_w^x = S^x[1], ..., S^x[k]$ and $s^y = s_{a+1}^y, ..., s_{a+w}^y = S^y[0], S^y[1], ..., S^y[k-1], S^y[k]$, where for the basic windows $S^y[0]$ and $S^y[k]$, only the last a values in $S^y[0]$ and the first $b-a$ values in $S^y[k]$ are included ($a < b$). We have

$$\psi(s^x, s^y) = \sum_{i=1}^{w} (s_i^x s_{a+i}^y)$$

$$= \sum_{j=1}^{k} \big(\rho(S^x[j], S^y[j-1], a) + \rho(S^y[j], S^x[j], b-a)\big)$$

where $\rho(S^1[p], S^2[q], d) = \sum_{i=1}^{d} S^1[p;i] S^2[q; b-d+i]$. For $S^1[p] = x_1, ..., x_b$ and $S^2[q] = y_1, ..., y_b$, $\rho(S^1[p], S^2[q], d)$ is the inner-product of the first d values of $S^1[p]$ with the last d values of $S^2[q]$. This can be approximated using only their first n DFT coefficients.

$$\sum_{i=1}^{d} x_i y_{b-d+i} \approx \sum_{i=1}^{d} \big(\sum_{m=0}^{n-1} c_m^x f_m(i) \sum_{p=0}^{n-1} c_p^y f_p(b-d+i)\big)$$

$$= \sum_{m=0}^{n-1} \sum_{p=0}^{n-1} c_m^x c_p^y \big(\sum_{i=1}^{d} f_m(i) f_p(b-d+i)\big)$$

$$= \sum_{m=0}^{n-1} \sum_{p=0}^{n-1} c_m^x c_p^y W(m,p,d)$$

This implies that if we precompute the table of

$$W(m,p,d) = \sum_{i=1}^{d} f_m(i) f_p(b-d+i)$$

$$m, p = 0..., n-1; d = 1, ..., \lfloor b/2 \rfloor$$

we can compute the inner-product using only the DFT coefficients without requiring the alignment of basic windows in time $O(n^2)$ for each basic window. Precomputation time for any specific displacement d is $O(n^2 d)$.

Theorem 1 *The n-approximate lagged inner-product of two time series using their first n DFT coefficients can be computed in time $O(kn)$ if the two series have aligned basic windows, otherwise it takes time $O(kn^2)$, where k is the number of basic windows.*

It is not hard to show that this approach can be extended to compute the inner-product of two time series over sliding windows of any size.

Corollary 1 *The inner-product of two time series over sliding windows of any size with any time delay can be approximated using only the basic window digests of the data streams.*

3.6 IO Performance

It might be desirable to store the summary data for future analysis. Since the summary data we keep are sufficient to compute all the statistics we are interested in, there is no need to store the raw data streams. The summary data will be stored on disk sequentially in the order of basic windows.

Let N_s be the number of streams and n be the number of DFT coefficients we use($2n$ real number), the I/O cost to access the data for all streams within a specific period will be

$$\frac{N_s k Sizeof(float)(2+2n)}{PageSize}$$

while the I/O cost for the exact computation is

$$\frac{N_s k Sizeof(float) b}{PageSize}$$

The improvement is a ratio of $\frac{b}{2+2n}$. The first 2 corresponds to two non-DFT elements of summary data: the sum and the sum of the squares of the time series in each basic window. Also the I/O costs above assume sequential disk access. This is a reasonable assumption given the time-ordered nature of data streams.

3.7 Monitoring Correlations Between Data Streams

The above curve fitting technique can be used for computing inner-products and correlations over sliding windows of any size, as well as with any time delay across streams. A frequent goal is to discover streams with high correlations. To do online monitoring of synchronized streams over a fixed size of sliding window with correlation above a specific threshold, we use an approach based on DFT and a hash technique that will report such stream pairs quickly.

First, we introduce the normalization of a series over sliding windows of size w, $x_1, x_2, ..., x_w$, as follows.

$$\hat{x}_i = \frac{x_i - \overline{x}}{\sigma_x}, \qquad i = 1, 2, ..., w$$

where

$$\sigma_x = \sqrt{\sum_{i=1}^{w} (x_i - \overline{x})^2}$$

As the following lemma suggests, the correlation coefficient of two time series can be reduced to the Euclidean distance between their normalized series[23].

Lemma 1 *The correlation coefficient of two time series $x_1, ..., x_w$ and $y_1, ..., y_w$ is*

$$corr(x, y) = 1 - \frac{1}{2} d^2(\hat{x}, \hat{y})$$

where $d(\hat{x}, \hat{y})$ is the Euclidean distance between $\hat{x}$ and $\hat{y}$.

Proof First, we notice that

$$\sum_{i=1}^{w} \hat{x}_i^2 = \sum_{i=1}^{w} \hat{y}_i^2 = 1$$

$$corr(x, y) = \frac{\sum_{i=1}^{w} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{w} (x_i - \overline{x})^2} \sqrt{\sum_{i=1}^{w} (y_i - \overline{y})^2}} = \sum_{i=1}^{w} \hat{x}_i \hat{y}_i$$

$$d^2(\hat{x}, \hat{y}) = \sum_{i=1}^{w} (\hat{x}_i - \hat{y}_i)^2 = \sum_{i=1}^{w} \hat{x}_i^2 + \sum_{i=1}^{w} \hat{y}_i^2 - 2 \sum_{i=1}^{w} \hat{x}_i \hat{y}_i$$
$$= 2 - 2 corr(x, y)$$

∎

By reducing the correlation coefficient to Euclidean Distance, we can apply the techniques in [2] to report sequences with correlation coefficients higher than a specific threshold.

Lemma 2 *Let the DFT of the normalized form of two time series x and y be $\hat{X}$ and $\hat{Y}$ respectively,*

$$corr(x, y) \geq 1 - \epsilon^2 \Rightarrow d_n(\hat{X}, \hat{Y}) \leq \epsilon$$

where $d_n(\hat{X}, \hat{Y})$ is the Euclidean distance between series $\hat{X}_1, \hat{X}_2, ..., \hat{X}_n$ and $\hat{Y}_1, \hat{Y}_2, ..., \hat{Y}_n$.

Proof As DFT preserves Euclidean distance, we have

$$d(\hat{X}, \hat{Y}) = d(\hat{x}, \hat{y})$$

Using only the first n and last $n, (n << w)$ DFT coefficients[2, 25], from the symmetry property of DFT, we have

$$\sum_{i=1}^{n} (\hat{X}_i - \hat{Y}_i)^2 + \sum_{i=1}^{n} (\hat{X}_{w-n} - \hat{Y}_{w-n})^2$$
$$= 2 \sum_{i=1}^{n} (\hat{X}_i - \hat{Y}_i)^2 = 2 d_n^2(\hat{X}, \hat{Y}) \leq d^2(\hat{X}, \hat{Y})$$
$$corr(x, y) \geq 1 - \epsilon^2 \Rightarrow d^2(\hat{x}, \hat{y}) \leq 2\epsilon^2$$
$$\Rightarrow d^2(\hat{X}, \hat{Y}) \leq 2\epsilon^2 \Rightarrow d_n(\hat{X}, \hat{Y}) \leq \epsilon$$

∎

From the above lemma, we can examine the correlations of only those stream pairs for which $d_n(\hat{X}, \hat{Y}) \leq \epsilon$ holds. We will get a superset of highly correlated pairs and there will be no false negatives. The false positives can be further filtered out as we explain later. HierarchyScan[19] also uses correlation coefficients as a similarity measure for time series. They use $\mathcal{F}^{-1}\{\hat{X}_i^* \hat{Y}_i\}$ as an approximation for the correlation coefficient. Because such approximations will not be always above the true values, some stream pairs could be discarded based on their approximations, even if their true correlations are above the threshold. Though HierarchyScan proposes an empirical method to select level-dependent thresholds for multi-resolution scans of sequences, it cannot guarantee the absence of false negatives.

We can extend the techniques above to report stream pairs of negative high-value correlations.

Lemma 3 *Let the DFT of the normalized form of two time series x and y be $\hat{X}$ and $\hat{Y}$.*

$$corr(x, y) \leq -1 + \epsilon^2 \Rightarrow d_n(-\hat{X}, \hat{Y}) \leq \epsilon$$

Proof We have

$$DFT(-x) = -DFT(x)$$

$$corr(x, y) \leq -1 + \epsilon^2 \Rightarrow \sum_{i=1}^{w} \hat{x}_i \hat{y}_i \leq -1 + \epsilon^2$$

$$\Rightarrow \sum_{i=1}^{w} (-\hat{x}_i) \hat{y}_i \geq 1 - \epsilon^2 \Rightarrow d_n(-\hat{X}, \hat{Y}) \leq \epsilon$$

∎

Now we discuss how to compute the DFT coefficients $\hat{X}$ incrementally. The DFT coefficients $\hat{X}$ of the normalized sequence can be computed from the DFT coefficients X of the original sequence.

Lemma 4 *Let $\hat{X} = DFT(\hat{x}), X = DFT(x)$, we have*

$$\begin{cases} \hat{X}_0 = 0 \\ \hat{X}_i = \frac{X_i}{\sigma_x} \quad i \neq 0 \end{cases}$$

We can maintain DFT coefficients over sliding windows incrementally[13].

Lemma 5 *Let X_m^{old} be the m-th DFT coefficient of the series in sliding window $x_0, x_1, ..., x_{w-1}$ and X_m^{new} be that coefficient of the series $x_1, x_2, ..., x_w$,*

$$X_m^{new} = e^{\frac{j2\pi m}{w}}(X_m^{old} + \frac{x_w - x_0}{\sqrt{w}})$$
$$m = 1, ..., n$$

This can be extended to a batch update based on the basic windows.

Lemma 6 *Let X_m^{old} be the m-th DFT coefficient of the series in sliding window $x_0, x_2, ..., x_{w-1}$ and X_m^{new} be that coefficient of the series $x_b, x_{b+1}, ..., x_w, x_{w+1}, ..., x_{w+b-1}$,*

$$X_m^{new} = e^{\frac{j2\pi mb}{w}} X_m^{old}$$
$$+\frac{1}{\sqrt{w}}(\sum_{i=0}^{b-1} e^{\frac{j2\pi m(b-i)}{w}} x_{w+i} - \sum_{i=0}^{b-1} e^{\frac{j2\pi m(b-i)}{w}} x_i)$$
$$m = 1, ..., n$$

The corollary suggests that to update the DFT coefficients incrementally, we should keep the following n digests for the basic windows.

$$\zeta_m = \sum_{i=0}^{b-1} e^{\frac{j2\pi m(b-i)}{w}} x_i, \qquad m = 1, ..., n$$

By using the DFT on normalized sequences, we also map the original sequences into a bounded feature space.

Lemma 7 *Let $\hat{X}_0, \hat{X}_1, ..., \hat{X}_{w-1}$ be the DFT of a normalized sequence $x_1, x_2, ..., x_w$, we have*

$$|\hat{X}_i| \leq \frac{\sqrt{2}}{2}, \qquad i = 1, ..., n, n < w/2$$

Proof

$$\sum_{i=1}^{w-1} (\hat{X}_i)^2 = \sum_{i=1}^{w} (\hat{x}_i)^2 = 1$$
$$\Rightarrow 2\sum_{i=1}^{n} \hat{X}_i^2 = \sum_{i=1}^{n} (\hat{X}_i^2 + \hat{X}_{w-i}^2) \leq 1$$
$$\Rightarrow |\hat{X}_i| \leq \frac{\sqrt{2}}{2}, \qquad i = 1, ..., n$$

∎

From the above lemma, each DFT coefficent ranges from $-\frac{\sqrt{2}}{2}$ to $\frac{\sqrt{2}}{2}$, therefore the DFT feature space R^{2n} is a cube of diameter $\sqrt{2}$. We can use a grid structure [4] to report near neighbors efficiently.

We will use the first $\hat{n}$, $\hat{n} \leq 2n$, dimensions of the DFT feature space for indexing. We superimpose an $\hat{n}$-dimensional orthogonal regular grid on the DFT feature space and partition the cube of diameter $\sqrt{2}$ into cells with the same size and shape. There are $(2\lceil\frac{\sqrt{2}}{2\epsilon}\rceil)^{\hat{n}}$ cells of cubes of diameter ϵ. Each stream is mapped to a cell based on its first $\hat{n}$ normalized DFT coefficients. Suppose a stream x is hashed to cell $(c_1, c_2, ..., c_{\hat{n}})$. To report the streams whose correlation coefficients with x is above the threshold $1 - \epsilon^2$, only streams hashed to cells adjacent to cell $(c_1, c_2, ..., c_{\hat{n}})$ need to be examined. Similarly, streams whose correlation coefficients with x are less than the threshold $-1 + \epsilon^2$, must be hashed to cells adjacent to cell $(-c_1, -c_2, ..., -c_{\hat{n}})$. After hashing the streams to cells, the number of stream pairs to be examined is greatly reduced. We can then compute their Euclidean distance, as well as correlation, based on the first n DFT coefficients.

The grid structure can be maintained as follows. Without loss of generality, we discuss the detection of only positive high-value correlations. There are two kinds of correlations the user might be interested in.

- **synchronized correlation** If we are interested only in synchronized correlation, the grid structure is cleared at the end of every basic window. At the end of each basic window, all the data within the current basic window are available and the digests are computed. Suppose that stream x is hashed to cell c, then x will be compared to any stream that has been hashed to the neighborhood of c.

- **lagged correlation** If we also want to report lagged correlation, including autocorrelation, the maintenance of the grid structure will be a little more complicated. Let T_M be a user-defined parameter specifying the largest lag that is of interest. Each cell in the grid as well as the streams hashed to the grid will have a timestamp. The timestamp T_x associated with the hash value of the stream x is the time when x is hashed to the grid. The timestamp T_c of a cell c is the latest timestamp when the cell c is updated. The grid is updated every basic window time but never globally cleared. Let S_c be the set of cells that are adjacent to c, including c. Here is the pseudocode to update the grid when stream x hashes to cell c:

FOR ALL cells $c_i \in S_c$
  IF $T_x - T_{c_i} > T_M$ //T_{c_i} is the timestamp of c_i.
    clear(c_i) //all the streams in c_i are out of date.
  ELSE IF $0 < T_x - T_{c_i} \leq T_M$
  //some streams in c_i are out of the date.
    FOR ALL stream $y \in c_i$
      IF $T_x - T_y > T_M$ *delete*(y)
      ELSE *examine_correlation*(x, y)
  ELSE //$T_x = T_{c_i}$
    FOR ALL stream $y \in c_i$
      examine_correlation(x, y)
  $T_{c_i} = T_x$ //to indicate that c_i is just updated

Theorem 2 *Given a collection of time series streams, using only the digests of the streams, our algorithms can find those stream pairs whose correlations (whether synchronized or lagged) are above a threshold without false negatives.*

Using the techniques in this section, we can search for high-value lagged correlations among data streams very fast. The time lag must be a multiple of the size of a basic window in this case. Theorem 1 states that we can approximate correlation of any time lag efficiently. The approximation methods are used as a post processing step after the hashing methods spot those promising stream pairs.

3.8 Parallel Implementation

Our framework facilitates a parallel implementation by using a straightforward decomposition. Consider a network of K servers to monitor N_s streams. We assume these servers have similar computing resources.

The work to monitor the streams has two stages.

1. Compute the digests and single stream statistics for the data streams. The N_s streams are equally divided into K groups. The server $i (i = 1, ..., K)$ will read those streams in the i-th group and compute their digests, single stream statistics and hash values.

2. Report highly correlated stream pairs based on the grid structure. The grid structure is also geometrically and evenly partitioned into K parts. A server X will read in its part, a set of cells S_X. Server X will also read a set of cells S'_X including cells adjacent to the boundary cells in S_X. Server X will report those stream pairs that are highly correlated within cells in S_X. Note that only the first n normalized DFT coefficients need to be communicated between servers, thus reducing the overhead for communication.

4 StatStream System

StatStream runs in a high performance interpreted environment called K[1]. Our system makes use of this language's powerful array-based computation to achieve high speed in the streaming data environment. The system follows the algorithmic ideas above and makes use of the following parameters:

- **Correlation Threshold** Only stream pairs whose absolute value of correlation coefficients larger than a specified threshold will be reported. The higher this threshold, the finer the grid structure, and the fewer streams whose exact correlations must be computed.

- **Sliding Window Size** This is the time interval over which statistics are reported to the user. If the sliding window size is 1 hour, then the reported correlations are those over the past hour.

- **Duration over Threshold** Some users might be interested in only those pairs with correlation coefficients above the threshold for a pre-defined period. For example, a trader might ask "Has the one hour correlation between two stocks been over 0.95 during the last 10 minutes?" This parameter provides such users with an option to specify a minimum duration. A longer duration period of highly correlated streams indicates a stronger relationship between the streams while a shorter one might indicate an accidental correlation. For example, a longer duration might give a stock market trader more confidence when taking advantage of such potential opportunities. A longer duration also gives better performance because we can update the correlations less frequently.

- **Range of Lagged Correlations** In addition to synchronized correlations, StatStream can also detect high-value lagged correlations. This parameter, i.e. T_M in section 3.7, specifies the range of the lagged correlations. For example, if the range is 10 minutes and the basic window is 2 minutes, the system will examine cross-correlations and autocorrelations for streams with lags of 2,4,8 and 10 minutes.

5 Empirical Study

Our empirical studies attempt to answer the following questions.

- How great are the time savings when using the DFT approximate algorithms as compared with exact algorithms? How many streams can they handle in real time?

- What's the approximation error when using DFT within each basic window to estimate correlation? How does it change according to the basic and sliding window sizes?

- What is the pruning power of the grid structure in detecting high correlated pairs? What is the precision?

We perform the empirical study on the following two datasets on a 1.5GHz Pentium 4 PC with 128 MB of main memory.

- **Synthetic Data** The time series streams are generated using the random walk model. For stream s,

$$s_i = 100 + \sum_{j=1}^{i}(u_j - 0.5), \qquad i = 1, 2, ...$$

  where u_j is a set of uniform random real numbers in $[0, 1]$.

- **Stock Exchange Data** The New York Stock Exchange (NYSE) Trade and Quote (TAQ) database provides intraday trade and quote data for all stocks listed on NYSE, AMEX, NASDAQ, and SmallCap issues. The database grows at the rate of 10GB per month. The historical data since 1993 have accumulated to 500GB. The data we use in our experiment are the tick data of the major stocks in a trading day. The 300 stocks in this dataset are heavily traded in NYSE. During the peak hours, there are several trades for each stock in a second. We use the price weighted by volume as the price of that stock at that second. In a second when there is no trading activities for a particular stock, we use the last trading price as its price. In this way, all the stock streams are updated every second, corresponding to a timepoint. The sliding window will vary from half an hour to two hours (1,800 to 7,200 timepoints). In practice the actual choice of the sliding windows will be up to the user. The lengths of the basic windows are half a minute to several minutes, depending on the number of streams to be monitored and the computing capacity.

5.1 Speed Measurement

Suppose that the streams have new data every second. The user of a time series stream system asks himself the following questions:

1. How many streams can I track at once in an online fashion? (Online means that even if the data come in forever, I can compute the statistics of the data with a fixed delay from their occurrence.)

2. How long is the delay between a change in correlation and the time when I see the change?

Our system will compute correlations at the end of each basic window. As noted above, the computation

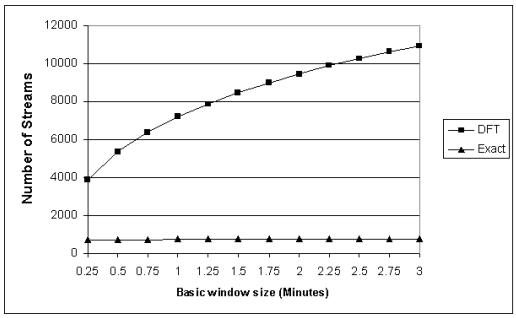

Figure 2: Comparison of the number of streams that the DFT and Exact method can handle

for basic window i must finish by the end of basic window $i + 1$ in order for our system to be considered on-line. Otherwise, it would lag farther and farther behind over time. Therefore, some of the correlations may be computed towards the end of the basic window $i + 1$. The user perceives the size of the basic window as the maximum delay between the time that a change in correlation takes place and the time it is computed.

The net result is that the answers to questions (1) and (2) are related. We can increase the number of streams at the cost of increasing the delay in reporting correlation.

Figure 2 shows the number of streams vs. the minimum size of the basic window for a uniprocessor and with different algorithms. In the DFT method, we choose the number of coefficients to be 16.

Using the exact method, given the basic window size b, the time to compute the correlations among N_s streams with b new timepoints is $T = k_0 b N_s^2$. Because the algorithm must finish this in b seconds, we have $k_0 b N_s^2 = b \Rightarrow N_s = \sqrt{\frac{1}{k_0}}$.

With the DFT-grid method, the work to monitor correlations has two parts: (1)Updating digests takes time $T_1 = k_1 b N_s$; (2)Detecting correlation based on the grid takes time $T_2 = k_2 N_s^2$. To finish these two computations before the basic window ends, we have $T_1 + T_2 = b$. Since T_2 is the dominating term, we have $N_s \approx \sqrt{\frac{b}{k_2}}$. Note that because of the grid structure, $k_2 << k_0$. Also, the computation with data digests is much more IO efficient than the exact method on the raw streams. From the equation above, we can increase the number of streams monitored by increasing the basic window size, i.e., delay time. This tradeoff between response time and throughput is confirmed in the figure. The number of streams handled by our system increases with the size of the basic window, while there is no perceivable change for the exact algorithm.

The wall clock time to find correlations using the DFT-grid method is much faster than the exact method (Figure 3a). The time is divided into two parts: detecting correlation and updating the digest

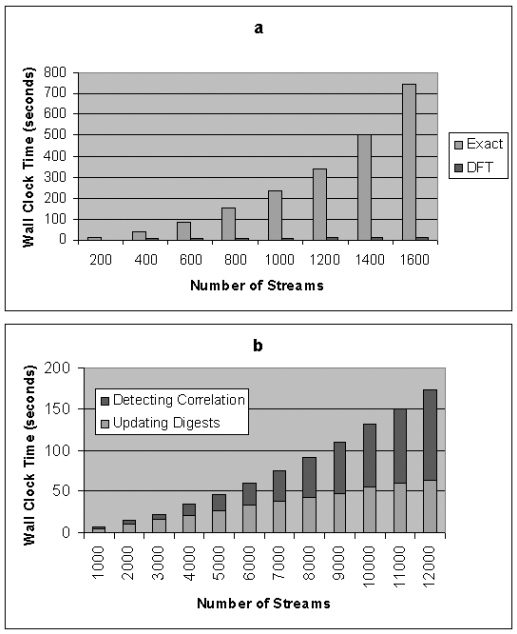

Figure 3: Comparison of the wall clock time

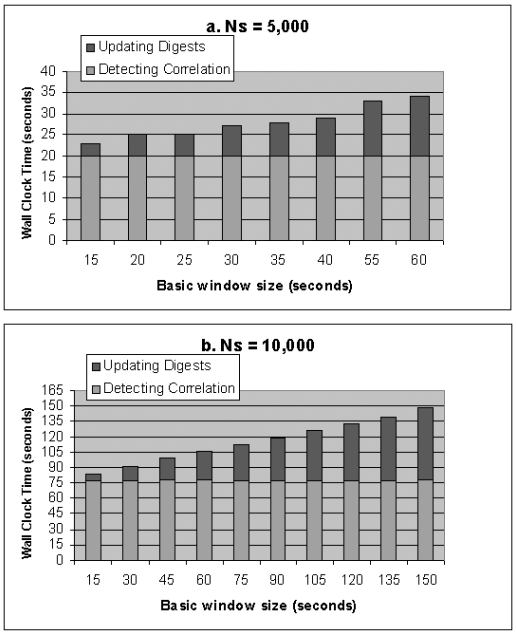

Figure 4: Comparison of the wall clock time for different basic window sizes

(Figure 3b).

The experiments on the processing time for the DFT-grid method also provide a guideline on how to choose the size of the basic window. Given a specific computing capacity, figure 4 show the processing time for different basic window sizes, when the numbers of streams are 5,000 (Figure 4a) and 10,000 (Figure 4b). Note that the wall clock time to detect correlation does not change with the size of the basic window, while the wall clock time to update the digest is proportional to the size of the basic window. Therefore the processing time is linear to the size of the basic window. Because the processing time must be shorter than the basic window time for our system to be online, we can decide the minimum basic window size. From figure 4 to monitor 5,000 streams the minimum basic window size is 25 seconds. Similarly, the basic window time should be no less than 150 seconds for 10,000 streams.

5.2 Precision Measurement

Because the approximate correlations are based on DFT curve fitting in each basic window, the precision of the computation depends on the size of the basic window. Our experiments on the two data sets (Figure 5) show that errors increase with larger basic window size and decrease with larger sliding window size, but remain small throughout. This is particularly noteworthy, because we used only the first 2 DFT coefficients in each basic window.

We also performed experiments to test the effectiveness of the grid structure. The grid structure on DFT feature space prunes out most of the low-correlated pairs of streams. The pruning power [17] is the number

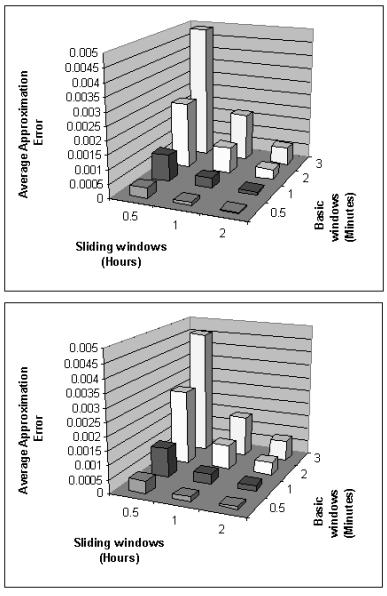

Figure 5: Average approximation errors for correlation coefficients with different basic/sliding window sizes for synthetic(above) and real(below) datasets

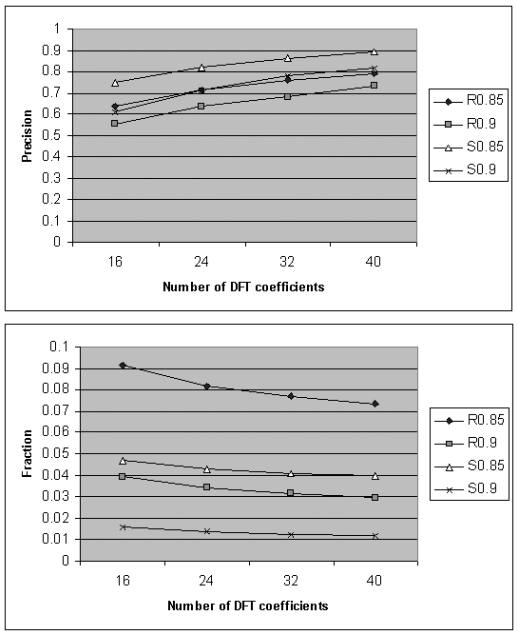

Figure 6: The precision and pruning power using different numbers of coefficients, thresholds and datasets

Table 2: Precision after post processing

Dataset	R0.85	R0.85	R0.9	R0.9	S0.85
Tolerance	0.001	0.0005	0.001	0.0005	0.0005
Precision	0.9933	0.9947	0.9765	0.9865	0.9931
Recall	1.0	0.9995	1.0	0.9987	1.0

of pairs reported by the grid, divided by the number of all potential pairs. Since our system guarantees no false negatives, the reported pairs include all the high-correlated pairs and some false positives. We also measure the quality of the system by precision, which is the ratio of the number of pairs whose correlations are above the threshold and reported by StatStream, to the number of pairs that are reported by StatStream. Figure 6 shows the precision and pruning power fraction using different numbers of coefficients and values of thresholds for different datasets. $R0.85$ indicates the real dataset with threshold of 0.85, $S0.9$ indicates the synthetic dataset with threshold of 0.9,etc. The length of the sliding window is one hour.

Table 2 shows that recall can be traded off against precision in post processing. The user may specify a tolerance t such that the system will report those pairs with approximate correlation coefficients larger than $threshold - t$.

6 Related Work

There is increasing interest in data streams. In the theoretical literature, Datar et. al [7] study the problem of maintaining data stream statistics over sliding windows. Their focus is single stream statistics. They propose an online data structure, exponential histogram, that can be adapted to report statistics over sliding windows at every timepoint. They achieve this with a limited memory and a tradeoff of accuracy. Our online basic window synopsis structure can report the precise single stream statistics for those timeponts that fall at basic window boundaries with a delay of the size of a basic window, but our multi-stream statistics also trade accuracy against memory and time.

Gehrke et al. [11] also study the problem of monitoring statistics over multiple data streams. The statistics they are interested in are different from ours. They compute correlated aggregates when the number of streams to be monitored is small. A typical query in phone call record streams is the percentage of international phone calls that are longer than the average duration of a domestic phone call. They use histograms as summary data structures for the approximate computing of correlated aggregates.

Recently, the data mining community has turned its attention to data streams. A domain-specific language, Hancock[6], has been designed at AT&T to extract signatures from massive transaction streams. Algorithms for constructing decision trees[8] and clustering [15] for data streams have been proposed. Recent work of Manku et al.[20], Greenwald et al.[14] have focused on the problem of approximate quantile computation for individual data streams. Our work is complementary to the data mining research because our techniques for finding correlations can be used as inputs to the similarity-based clustering algorithms.

The work by Yi et al.[28] for the online data mining of co-evolving time sequences is also complementary to our work. Our approximation algorithm can spot correlations among a large number of co-evolving time sequences quickly. Their method, MUSCLES can then be applied to those highly correlated streams for linear regression-based forecasting of new values.

Time series problems in the database community have focused on discovering the similarity between an online sequence and an indexed database of previously obtained sequence information. Traditionally, the Euclidean similarity measure is used. The original work by Agrawal et al. [2] utilizes the DFT to transform data from the time domain into frequency domain and uses multidimensional index structure to index the first few DFT coefficients. In their work, the focus is on whole sequence matching. This was generalized to allow subsequence matching [9]. Rafiei and Mendelzon[24] improve this technique by allowing transformations, including shifting, scaling and moving average, on the time series before similarity queries. The distances between sequences are measured by the Euclidean distance plus the costs associated with these transformations. Our work differs from them in that (1)In [2, 9, 24, 19], the time series

are all finite data sets and the focus is on similarity queries against a sequence database having a preconstructed index. Our work focuses on similarity detection in multiple online streams in real time. (2)We use the correlation coefficients as a distance measure like [19]. The correlation measure is invariant under shifting and scaling transformations. Correlation makes it possible to construct an efficient grid structure in bounded DFT feature space. This is what enable us to give real time results. [19] allows false negatives, whereas our method does not.

Other techniques such as Discrete Wavelet Transform (DWT) [5, 26, 22, 12], Singular Value Decomposition (SVD)[18] and Piecewise Constant Approximation (PCA)[27, 17] are also proposed for similarity search. Keogh et al. [17] compares these techniques for time series similarity queries. The performance of these techniques varied depending on the characteristics of the datasets, because no single transform can be optimal on all datasets. These techniques based on curve fitting are alternative ways of computing digests and could be used in our sliding window/basic window framework.

7 Conclusion

Maintaining multi-stream and time-delayed statistics in a continuous online fashion is a significant challenge in data management. Our paper solves this problem in a scalable way that gives a guaranteed response time with high accuracy.

The Discrete Fourier Transform technique reduces the enormous raw data streams into a manageable synoptic data structure and gives good I/O performance. For any pair of streams, the pair-wise statistic is computed in an incremental fashion and requires constant time per update using a DFT approximation. A sliding/basic window framework is introduced to facilitate the efficient management of streaming data digests. We reduce the correlation coefficient similarity measure to a Euclidean measure and make use of a grid structure to detect correlations among thousands of high speed data streams in real time. Experiments conducted using synthetic and real data show that StatStream detects correlations efficiently and precisely.

References

[1] http://www.kx.com.

[2] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient Similarity Search In Sequence Databases. In *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, 1993.

[3] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):109–120, 2001.

[4] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Transactions on Mathematical Software (TOMS)*, 6(4):563–580, 1980.

[5] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia*, pages 126–133, 1999.

[6] C. Cortes, K. Fisher, D. Pregibon, and A. Rogers. Hancock: a language for extracting signatures from data streams. In *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 9–17, 2000.

[7] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, page to appear, 2002.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 71 – 80, 2000.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 419–429, 1994.

[10] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Data evolution and monitoring. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California*, 2000.

[11] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proc. ACM SIGMOD International Conf. on Management of Data*, 2001.

[12] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, pages 79–88, 2001.

[13] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. In *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming (CP'95)*. Springer Verlag, 1995.

[14] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 58–66, 2001.

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

[16] E. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In *the third conference on Knowledge Discovery in Databases and Data Mining*, 1997.

[17] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. ACM SIGMOD International Conf. on Management of Data*, 2001.

[18] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 289–300, 1997.

[19] C.-S. Li, P. S. Yu, and V. Castelli. Hierarchyscan: A hierarchical similarity search algorithm for databases of long sequences. In *ICDE*, pages 546–553, 1996.

[20] G. S. Manku, S. Rajagopalan, , and B. G. Lindsay. Random sampling techniques for space efficientonline computation of order statistics of large datasets. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 251–262, 1999.

[21] L. Molesky and M. Caruso. Managing financial time series data: Object-relational and object database systems. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases*, 1998.

[22] I. Popivanov and R. J. Miller. Similarity search over time series data using wavelets. In *ICDE*, 2002.

[23] D. Rafiei. On similarity-based queries for time series data. In *ICDE*, pages 410–417, 1999.

[24] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 13–25, 1997.

[25] D. Rafiei and A. Mendelzon. Efficient retrieval of similar time sequences using dft. In *Proc. FODO Conference, Kobe, Japan*, 1998.

[26] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. A comparison of dft and dwt based similarity search in time-series databases. In *Proceedings of the 9 th International Conference on Information and Knowledge Management*, 2000.

[27] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 385–394, 2000.

[28] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. In *Proceedings of the 16th International Conference on Data Engineering*, pages 13–22, 2000.