Cypress: Managing Massive Time Series Streams with Multi-Scale Compressed Trickles

Galen Reeves
Dept. of EECS
University of California
Berkeley, CA, USA
greeves@eecs.berkeley.edu

Jie Liu, Suman Nath, Feng Zhao
Microsoft Research
One Microsoft Way
Redmond, WA, USA
{liuj,sumann,zhao}@microsoft.com

ABSTRACT

We present Cypress, a novel framework to archive and query massive time series streams such as those generated by sensor networks, data centers, and scientific computing. Cypress applies multi-scale analysis to decompose time series and to obtain sparse representations in various domains (e.g. frequency domain and time domain). Relying on the sparsity, the time series streams can be archived with reduced storage space. We then show that many statistical queries such as trend, histogram and correlations can be answered directly from compressed data rather than from reconstructed raw data. Our evaluation with server utilization data collected from real data centers shows significant benefit of our framework.

1. INTRODUCTION

Increasing instrumentation of physical and computing processes has given us unprecedented capabilities to collect data. Applications for data center management, environmental monitoring, scientific experiments, and mobile asset tracking produce massive time series streams from various sensors. Typical data stream systems [1, 22] can process such signals in real time for interesting queries. Such systems, however, are inadequate for many applications as the intended queries to run on a stream may not be known when the data is produced. Therefore, such applications require a stream archiving and analysis system (SAAS) that can archive data for a long period of time and efficiently support various statistical and data mining queries on historic data.

Both archiving and query processing in a SAAS can be extremely challenging. For example, consider a data center monitoring application. Since data centers are large capital investments for online service providers, they are closely monitored for operating conditions and utilizations. A data center may contain tens of thousands of servers. Assume that 100 performance counters are collected from each server to monitor its utilization. In addition, for each server, 10 physical sensors are used to monitor its power consumption and operation environment (e.g. temperature around it, etc.). Then, a data center with 50,000 servers yields 55 million concurrent data streams and, with a mere 30-second sampling rate, more than 15 billion records (or about 1TB data) a day. While the most recent data are used in real-time monitoring and control, the historical archived data are used for tasks such as capacity planning, workload placement, pattern discovery, and fault diagnostics. As we will discuss in Section 2, many of these tasks require queries for computing pair-wise correlation, histogram, and first-order trend of time series data over last several months. Due to the sheer volume of the data, archiving it in its raw form for several months may consume prohibitively large storage space, while querying it may be impractically slow.

In this paper, we consider the problem of space-efficient archival of a large number of massive time series and fast processing of several statistical and data mining queries on that archived data. Traditional database systems address space-efficient archival and query processing separately. For example, many database systems compress data for space-efficiency; however, queries are run on uncompressed data and hence data must be uncompressed before a query is processed. Such an approach is not feasible for our target applications since the decompression overhead would make query latency even worse. We therefore incorporate compression and query processing in the same framework; i.e., our queries run directly on compressed data. This, in addition to avoiding expensive decompression before a query, enables fast query processing since a query runs on a much smaller amount of data. Several existing systems support efficient queries on compressed data; however, they are not general enough for our target applications. For example, StatStream computes correlation directly from compact Fourier representation of streams (via FFT); however, the technique is not effective for correlating noisy data or for preserving important spikes in data. Systems that compress data using polynomial approximation [11, 25] can preserve spikes, but do not efficiently support correlation queries.

Our framework, called *Cypress*, preprocesses and decomposes each data stream into a small number of substreams, which we call *trickles*, and answers common queries directly from a set of the trickles rather than reconstructing the original streams. The multi-scale lossy compression of Cypress is motivated by two important observations. First, typical sensor streams are not just sequences of random data. Many of them are numerical values that reflect underlying physical or computing processes. Although noisy, the infor-

VLDB '09, August 24-28, 2009, Lyon, France

mation they contain is usually sparse, as we will elaborate in section 2. Such data can be effectively compressed with data reduction techniques such as FFT, random projections, etc. Moreover, as we will show later in this paper, our core queries of interest, e.g., correlation, histograms, trends, etc., can be efficiently answered directly from such compressed data.

Second, a sensor stream can be decomposed into different components that have different compressibility and usefulness in query processing. Thus, compressing an entire signal with one generic data reduction technique may be suboptimal. For example, Figure 1(a) shows CPU utilization of a server (more details in Section 2), which consists of some noise, occasional spikes, and a high-level trend. Space-efficient compression of the entire signal with techniques such as FFT will capture the high-level trend of a signal, but will fail to capture spikes and noises, both of which are important for several important queries we consider. On the other hand, answering a query from the raw signal or its high-level trend may sometimes be incorrect. For example, pair-wise correlation of two raw signals may be dominated by their high level trends, possibly answering queries relying on correlation of noisy components of two signals incorrectly. We elaborate these in Section 2.

To address the above limitations of systems that do not consider different frequency components of a signal separately, Cypress preprocesses sensor data streams using multiple filters. For ease of discussion, we focus on 1-dimensional numerical data streams (or signals) in this paper, although the framework can be extended to multi-dimension and multimedia data streams as well. Many real sensor data streams are oversampled to avoid missing abnormal events. Thus, the "energy" of the signals are usually concentrated at low frequency bands. FFT is a common way to identify what frequency bands have significant presence in the signal. The high-level trends of such signals can be efficiently represented by filtering and downsampling the signals. We call these filtered and downsampled signals the *LoF trickles*. Downsampled signals can lose abrupt events, or spikes, in the signals. For this reason, we tease out spikes and store them separately as *spike trickles*. Notice that LoF signals are sparse in the frequency domain and spikes are sparse in the time domain. So, the storage space requirement for archiving them is small.

After removing LoF and spikes, the residuals' values, as well as their Fourier coefficients, are small. Although less significant in either domains, they may still contain useful information, especially when comparing across signals. For example, the similarity in residuals may reinforce or complement the similarity in other trickles. For these residual signals, we apply Johnson-Lindenstrauss style random projections [2, 19] (or sketches) for further dimensionality reductions. We call these sketches *HiF trickles*. We show that random projections preserve correlations among signals, and analyze the accuracy of correlation coefficients and projection length. Random projections are also *universal*, in the sense that if, in the future, transformations are discovered such that the residuals have a sparse representation in a new domain, the signals can be fully recovered from the projections, according to recent results from the theory of compressive sensing [2, 8, 14, 17].

Apart from the multi-scale compression framework, we study how common queries can be answered directly from trickles. We show that LoF + Spikes can approximate histograms well. We further analyze the relationship between the accuracy of correlation of two signals directly from their sketches and sketch lengths. We give an accuracy upper bound for using Gaussian random projections. We also show that random projections are more robust than other compression techniques, such as downsampling, when dealing with wideband, nonstationary noise-like data.

Although the multi-scale analysis of the Cypress framework is related to previous methods based on Wavelet decompositions [12, 13], the different types of compression techniques used at different scales in the Cypress framework provide stronger guarantees for various queries of interest. For instance, Wavelet methods could fail to capture all the spikes. The scheme presented in the paper [12] makes only probabilistic guarantees on preserving all spikes, and hence is not suitable for applications that do not want to lose any spikes. Furthermore, although high frequency wavelet coefficients can capture noise, retaining all such coefficients does not provide any compression benefit. In contrast, sketching provides a robust solution to extract, from noisy data, small number of features useful for answering correlation-type queries.

Also, although the Cypress framework uses some of the same principals (such as sparsity and random projections) as in compressed sensing and random sketching [2, 8, 14, 17] there are several key differences. First, it is known in advance where the sparsity in the data is located, and thus we use direct methods (e.g. filtering followed by thresholding) to efficiently take advantage of the sparsity in both the time and frequency domains. Using random projections on the raw data streams would be overly expensive (in terms of reconstructing the signals) and also be unreliable because the non-sparse "noise" components are sufficiently large to render reconstructions highly inaccurate. A second difference is that when we apply random projections to the high frequency components, the goal is not compression with respect to a sparse basis (no such basis is known), but rather a robust "sufficient statistic" for measuring correlations.

The contributions of the paper are the following:

- We present the design of Cypress, a multi-scale filtering and compression framework for massive data streams.
- We show that statistics such as trends, histograms, and correlations can be answered from trickles without reconstructing raw data.
- We provide theoretical analysis on the relationship between the accuracies of correlation analysis and the sketch lengths.
- Through realistic data sets, we evaluate the performance and accuracy of the Cypress framework.

The rest of the paper is organized as follows. In section 2, we give a running example of a data center monitoring application and describe common queries on performance counters. In section 3 we present our Cypress framework. In section 4, we show how histogram queries can be efficiently answered by our framework. In Section 5, we analytically show the accuracy of correlation of two signals directly from their random sketches, as a function of sketch length. We evaluate the Cypress framework to performance counters collection

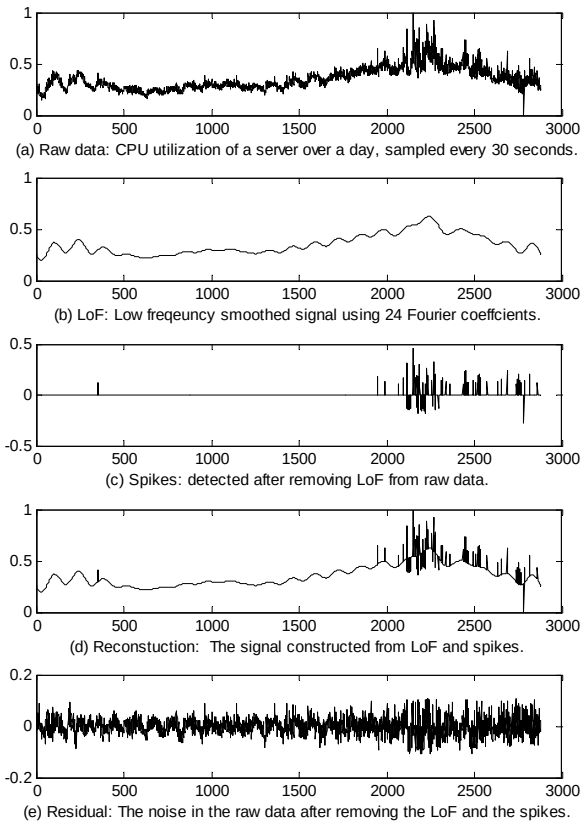

Figure 1: Decomposition of the raw CPU utilization signal (a) into low-frequency (b), spike (c), and high frequency (e) components. Each component is compressed as a separate trickle.

from about 800 servers in a Microsoft online service in section 6. In section 7 we discuss related work. We conclude the paper in Section 8.

2. A RUNNING EXAMPLE

We introduce a running example using performance counters from a production Internet service with millions of users. There are three types of servers — A, B, and C. We use 50 server for each type (150 servers total) in the data set. All servers are stateful, but in different ways. Type A servers are client facing, behind load balancers. They maintain long living TCP connections with clients. Type B and C servers are internal servers behind type A servers.

Without losing generality, we use CPU utilization as an example throughout our discussion. The CPU utilization on each server is affected by many factors, such as total number of users in the system, load balancing algorithms that determine the number of users on each server, background tasks, and, in rare cases, software bugs, etc. In our example, processor utilizations are collected every 30 seconds from each server. So, each server generates 2880 CPU utilization samples per day. Figure 1(a) shows a typical CPU utilization trace for a type A server over 1 day.

As mentioned before, Cypress archives this data as a collection of trickles: the LoF trickles represent the high level trend of the signal (Figure 1,b), the spike trickles representing the spikes (Figure 1,c), and the HiF trickles representing the residual noisy signal (Figure 1,e). While our target queries can be directly answered using the trickles, LoF and spike trickles also allow approximate reconstruction of the original signal (without the noise component), as shown in Figure 1(d). Thus, queries that may not be directly supported by the trickles can be answered on this reconstructed signal.

We now show how typical queries are answered by using different trickles. Some example queries that a data center operator wants to make on the CPU data are as follows.

- Q1 (capacity planning): What is the average growth rate for the service over last three months?
- Q2 (server provisioning): How many servers have reached 80-percentile utilization in the last Christmas season?
- Q3 (dependency analysis): How behaviors of type A servers correlated with those of type B servers?
- Q4 (load balancing): Do servers within a cluster receive balanced load?
- Q5 (anomaly detection): Are utilization spike patterns on servers in one cluster also exhibited by other servers?

We notice that given the spatial and temporal span of the data, all these queries are statistical in nature. Small approximation errors in answers are acceptable as they do not effect decision making quality.

Q1 is a trend query. In this application, the number of users in the system is known to scale linearly with total CPU utilization [5]. So, average user growth can be approximated by computing daily average of CPU utilizations across all servers. This can be done by using the LoF trickles and linear regressions.

Q2 is a histogram query over a time window and across all servers. In Section 4, we will show that this can be answered directly from LoF trickle and spikes.

Q3, Q4, and Q5 are all correlation queries, seeking similarity in the data directly or features contained in the data. However, they need to be answered using different components of the signal (and hence different trickles). To illustrate Q3, Figure 2(a) shows the correlation coefficients computed across 150 servers using the raw data over a day. The image is a visualization of a 150×150 matrix, indexed by the servers. That is, the $(i, j)^{th}$ element of the matrix is the correlation coefficient of the values (2880 samples) between the i^{th} server and the j^{th} server. The diagonal elements are 1's. We can clearly see the two clusters of B servers, and that A servers' behaviors are strongly correlated. Type C servers are less correlated, indicating that the access patterns to C servers are not uniform. In addition, those were four abnormal servers that were idling for the day. It is interesting to observe that Q3 can also be answered accurately with much fewer data points. For example, Figure 2(b) shows the correlation coefficients computed from the LoF trickle from the 150 servers. Even with only 24 data points per server, we can still clearly differentiate server types, their strong correlation within the clusters A and B, and the abnormal servers.

In contrast, Q4 needs to be answered using the high frequency components of the signals (or, the Spike trickles). Figure 2(c) and (d) shows two clusters within Type B servers,

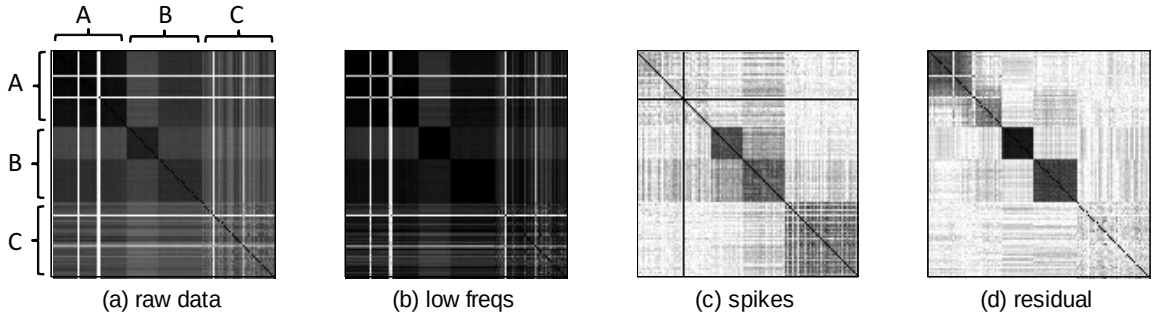

Figure 2: The correlation coefficients across 150 servers, computed using different components of signals. Darker pixels represent higher correlation coefficients (e.g., a black pixel represents a corr. coeff. of 1).

implying the presence of two load-balanced clusters. The presence of these clusters is not visible in Figure 2 (a) and (b), suggesting that Q4 cannot be answered from the low frequency signals. On the other hand, Figure 2(d) pronounces the clusters more clearly than (c), and hence Q4 is best answered from residual noises (i.e., HiF trickle).

Finally, Q5 needs to be answered from the Spikes trickles of the signals, as other components do not capture the spikes. As shown, some correlations in type C servers are visible only in Figure 2(c).

3. MULTI-SCALE COMPRESSION FRAMEWORK

The key for data compression is to seek structure (e.g. sparsity) in the data streams and to tease out data that do not contribute to the accuracy of query answers. Throughout this section, the starting point is considered to be a one-dimensional raw data stream represented by a real-valued discrete time signal $x(n)$. It is assumed that this signal corresponds to uniformly spaced samples of a continuous time process (e.g. CPU utilization) with initial sampling rate f_s samples per second (i.e. there are $1/f_s$ seconds between each sample). Compression is achieved by representing the signal $x(n)$ with fewer than $1/f_s$ values per second while still maintaining the ability to answer queries of interest.

In section 3.1 below we review standard properties from signal processing and perform spectrum analysis the CPU data. The upshot of this analysis is that a great deal of the signal's "energy" can be persevered using filtering and down-sampling. Next, in Section 3.2 we introduce the full Cypress framework which goes beyond the basic ideas of filtering and down-sampling and allows for high resolution answers to various type of queries.

3.1 Spectrum Analysis

Fourier-type transforms allow us to analyze time-series in the frequency domain. For example, Figure 3 shows the frequency spectrum measured over four days of CPU utilization computed after removing the mean. It is clear that most of the "energy" in the signal is stored in low frequency components which correspond to the LoF trickle.

The mechanism to keep the signals in the time domain is through filtering and downsampling. For completeness of discussion, we review the following basic properties. A discrete time signal $x(n)$ can be equivalently represented by

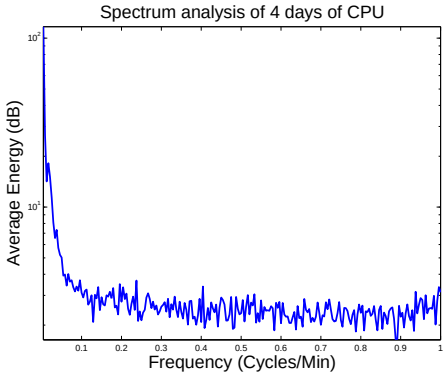

Figure 3: Spectrum analysis of CPU utilization over four days after removing the daily mean.

it discreet time Fourier transform (DTFT)

$$X(f) = \sum_n x(n) e^{-i\,n\,2\pi f/f_s} \tag{1}$$

where the frequency parameter f has units of cycles/seconds. Note that $X(f)$ is periodic with period f_s. A discrete time signal $x(n)$ is said to have bandwidth f_c if $f_c < f_s/2$ and the magnitude of $X(f)$ is equal to zero for all frequencies f such that $f_c < |f| < f_s - f_c$. As we show below, the significance of the bandwidth is that it determines the minimum sampling rate required to accurately represent the signal.

Downsampling $x(n)$ by a factor of L consists of keeping every L'th sample and disregarding the others. The DTFT of the downsampled signal $x_L(n) = x(nL)$ corresponds to the sum of L shifted versions of $X(f)$,

$$X_L(f) = \frac{1}{L} \sum_{k=0}^{L-1} X(f - (k/L) f_s), \tag{2}$$

and the new sampling rate of the downsampled signal is given by f_s/L. If the bandwidth f_c is greater than $f_s/2L$, then the shifted versions overlap and aliasing occurs. However, if $f_c < f_s/2L$ then aliasing does not occur and the original sequence $x(n)$ can be recovered by first stretching $x_L(n)$ by a factor L (i.e. inserting $L-1$ zeros between each sample of $x_L(n)$) and then ideal low-pass filtering the resulting signal with cutoff frequency f_c. These key aspects are

summarized in the following observations.

OBSERVATION 3.1. *If a discrete time signal $x(n)$ has bandwidth $f_c < f_s/2L$, then downsampling $x(n)$ by a factor L does not introduce any aliasing. That is, downsampling preserves all frequency components.*

Note that the band-limited condition is important. Downsampling a wideband signal (e.g. raw data) does not preserve its spectrum.

OBSERVATION 3.2. *If a discrete time signal $x(n)$ has bandwidth $f_c < f_s/2$, then $x(n)$ can be upsampled by a factor L by passing the L-stretched signal through an ideal low-pass filter with cutoff frequency f_c.*

A key approach in our system then, is to separate the time series into low and high frequency components. From Figure 3 it is clear that relatively low sampling rate is needed to retain the majority of the energy in the data. The dominance of the resulting LoF components can be seen by reconstructing one days of CPU utilization based on various sampling rates. Figure 1(b) shows the low frequency components in the time domain corresponding to sampling rates of 48. With only two samples per hour (48 samples per day), it is already possible to see the trend in the data.

3.2 The Cypress Framework

Our Cypress framework applies a sequence of filters and statistical analysis to the raw data to identify sparsity in the raw signals. The top part of Figure 4 shows the flow of the overall compression process. We use an example CPU utilization trace mentioned in Section 2 to help explain the process. Plots (A)–(G) in the figure shows how the data look like after various steps.

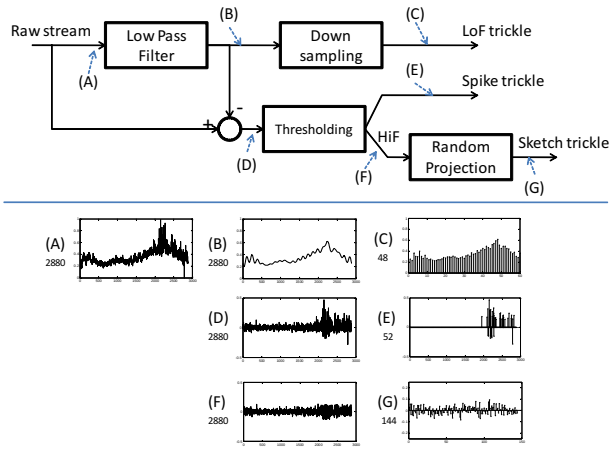

Figure 4: The Cypress framework for multi-scale compression. The top part shows the flow of the process. The plots (A) – (G) show the effects on an example signal (CPU utilization) flowing through the system. The numbers below the plot labels are the signal lengths. In this case, we reduce the 2880 sample raw data into three trickles of 48, 52, and 144, respectively, achieving a data reduction of about 91.5%.

3.2.1 LoF Trickles

In order to obtain the LoF representation of the signal, the raw data (A in Figure 4) first goes through a low pass filter with cut off frequency $f_s/2M$, where M is an application specific parameter that is tunable based on the nature of the data streams. It can be experimentally determined by performing spectrum analysis on examples of the data streams and analyzing the energy concentration in the spectrum. For example, we use $M = 60$ in the filtering of the CPU utilization data.

The effect of low pass filtering is to create a band-limited signal (B) (in Figure 4) with bandwidth $f_s/2M$ so that downsampling by a factor M to signal (C) does not create aliasing. That is, if required in the future, we can reconstruct a full length smooth signal (B) by first M-stretching signal (C) and then running it through the same low pass filter.

The downsampled signal (C) captures low frequency trends in the original data. For example, the average signal value over one day based on the LoF trickle (C) requires only 48 data points and is essentially the same as the average over the same time period based on the 2880 data points of the raw stream (A).

To answer histogram type of queries, we prefer to store LoF representation in the time domain, rather than the frequency domain, although they are equivalent in terms of information contents and the accuracy of approximating the original signal. This decision differentiates our approach from StatStream [22].

3.2.2 Spike Trickles

The difference (D) between the raw stream (A) and its smoothed approximation (B) is a zero mean "noise" like signal. So, it does not contribute much to trending analysis. However, any significant variation from the standard deviation is potentially interesting for several useful queries (e.g., Q2 and Q5 in section 2).

Cypress detects the spikes of a signal by applying a threshold to the "noise", which is the signal obtained by subtracting the bandpass signal (B) from the original signal (A). The threshold can be application specific. In this example we use 3σ as the threshold, where σ is the standard deviation of the "noise".

Spikes are typically sparse, due to the Gaussian like distribution of the noise. In our example, there are 52 spikes in (E). They captures moments where the CPU utilization has abrupt changes, including when it reaches 100% and 0.

3.2.3 HiF Trickles

After removing LoF and spikes, the residual is bounded in value but has a wide frequency band. It does not contribute much to the statistical analysis of a single stream, such as trending, histogram, or rare-event analysis. However, as we will see further in section 5, they still contain rich information, especially when correlating across data streams. So, for applications where the residual cannot be simply ignored, we need to find ways to archive and process them efficiently.

General purpose compression of wide band signals is challenging since the entropy in the signal is high. However, since the purpose of keeping the residuals are for correlation analysis beyond the low frequency bands, we apply *Johnson-Lindenstrauss* style compressive random projections (also called *sketches*). We call such sketches HiF trickles. A

compressive random projection is a linear transformation $A \in \mathbb{R}^{K \times N}$ with $K < N$, where the elements of the matrix are drawn i.i.d from an appropriate distribution (e.g. Gaussian or Bernoulli). For a vector with dimension (i.e. length) N, the projected vector is of dimension K. Random projections are universal dimension reduction techniques that preserve the relative distance between vectors [17] .

While LoF and spike trickles are generated in a streaming fashion, HiF trickles are generated in batches. Like StatStream, we group n successive residual data points into a *basic window* of length N, and compute the HiF trickle for the entire basic window. A key question to answer when applying random projections is the projection length K. Obviously, the shorter the length is, the better data reduction rate can be achieved and the less data need to processes when computing for queries. However, shorter projections can also sacrifice query answer quality. Further discussions on the tradeoff between projection length and correlation accuracy is given in section 5.

In summary, the Cypress framework breaks each signal x into three kinds of streams based on their time scales (or frequencies): a LoF signal x_L, a spike signal x_S, and a HiF signal x_H. When the signals are archived, the three kinds of signals are further converted into three kinds of trickles: a LoF trickle $\tilde{x}_L$ via downsampling, a spike trickle $\tilde{x}_S = x_S$ as is, and compressed HiF trickle $\tilde{x}_H$ via random projections of x_H.

3.2.4 The System Model

Similar to StatStream [22], Cypress distinguishes between three time periods: i) timepoint: the smallest unit of time over which the system collects data, e.g. second; ii) basic window: a consecutive subsequence of timepoints that Cypress processes in a batch to maintain three different types of trickles, e.g., several hours; and iii) query window: a user-defined consecutive subsequence of basic windows over which the user wants statistics, e.g. a day or a week. As each signal streams into Cypress, a basic window worth of data is buffered, which is then decomposed into individual trickles.

A query in Cypress is issued on a particular trickle set, chosen based on the query. For example, a histogram query can be made on the LoF trickles only, if the querier wants to ignore the spikes and residual noise. Therefore, different trickles are archived in different database tables. To support queries on an arbitrary query window, trickles within each trickle table are indexed on their signal IDs and timestamps. Cypress uses the Multi-skip List (MSL) data structure [11] for this purpose, which supports efficiently retrieving trickles for a given set of signal IDs and within a time range. For lack of space, we do not discuss the details of Cypress's storage component in this paper.

4. HISTOGRAM QUERIES

Histograms are a commonly used analysis tool for archived data that provide valuable insight in the distributions of signal values. For instance, if the signals are viewed as random processes, then histograms are essentially a smoothed estimates of the underlying probability distributions. Accordingly, the ability to accurately and efficiently compute histograms is an important component of our compression framework.

In this section, we address one dimensional histograms that correspond to an analysis window of length N, and we use the vector notation $\mathbf{x} = [x(n), \cdots, x(n+N-1)]$. We refer to both $x(n)$ and $\mathbf{x}$ as signals, where is it clear that $x(n)$ is a stream and $\mathbf{x}$ is a vector.

For a given signal $\mathbf{x}$, and given set of B intervals (or bins) $(b_0, b_1], (b_1, b_2], \cdots, (b_{B-1}, b_B]$, we define the histogram to be the vector $H(\mathbf{x}) = [H_1(\mathbf{x}), \cdots, H_B(\mathbf{x})]$ where $H_i(\mathbf{x})$ is the number of elements in $\mathbf{x}$ whose value falls in the ith bin.

It is desirable to answer histogram queries based on as few tickles as possible and from as few samples within each tickle as possible. Figure 5 shows a histogram with 10 bins (10% CPU utilization per bin) computed from raw signal (Figure 5 (a)) and a reconstructed signal (Figure 5 (b)).

We let $\mathbf{x}$ denote the original signal and let $\tilde{\mathbf{x}} = \mathbf{x}_L + \mathbf{x}_S$, where $\mathbf{x}_L$ and $\mathbf{x}_S$ denote the LoF and spike components of $\mathbf{x}$. We further define the *Histogram Error Ratio* (HER) to be the total differences in each bin counts divided by the total number of samples in the signal:

$$\mathrm{HER}(\mathbf{x}) = \frac{\sum_{i=1}^{B} |H_i(\mathbf{x}) - H_i(\tilde{\mathbf{x}})|}{\sum_{i=1}^{B} H_i(\mathbf{x})} \quad (3)$$

For example, the HER of the histograms shown in Figure 5 is 6.4%.

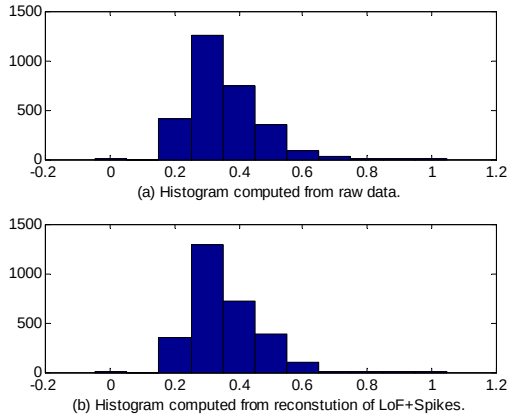

Figure 5: Comparison of histograms computed from raw signal and that computed from (LoF+Spike) reconstruction.

To understand the accuracy of this histogram approximation, we need to answer two questions:

Effects of discarding HiF signals. Since spikes are kept with full precision in the trickles, it is fair to compare the effect of HiF signals $\mathbf{x}_H$ on the histogram of LoF signals $\mathbf{x}_L$.

The HiF signal $\mathbf{x}_H$ has the following statistical properties:

- The mean of $\mathbf{x}_H$ is approximately zero since we removed a small number of spikes from an otherwise zero mean signal.
- $|\mathbf{x}_H| < d$, where d is the threshold for detecting spikes.

If we assume that $\mathbf{x}_H$ and $\mathbf{x}_L$ correspond to independent random processes, then the probability density function (pdf) of $\mathbf{x}_H + \mathbf{x}_L$ is the given by the convolution of

the pdf of $\mathbf{x}_H$ and of $\mathbf{x}_L$. So, adding $\mathbf{x}_H$ to $\mathbf{x}_L$ effectively blurs the histogram by spreading each bin in the histogram to its neighbor bins. The degree of blurring depends on the resolution of the bins and the pdf of the two random variables. For example, assume that $\mathbf{x}_L$ has n samples in the bin $(a, b]$, and $b - a > 2d$. If both $\mathbf{x}_L$ and $\mathbf{x}_H$ have uniform distributions, then $\mathbf{x}_L + \mathbf{x}_H$ will spread over three ranges $(a - d, a]$, $(a, b]$, and $(b, b + d]$, with $\frac{d}{b-a+2d}n$, $\frac{b-a}{b-a+2d}n$, and $\frac{d}{b-a+2d}n$ samples. In other words, the error scales linearly with d, the range of the HiF signal. If the pdf of $\mathbf{x}_H$ is close to a Gaussian distribution with fast decay of probability from the (zero) mean, which is true for many noisy signals, then the amount of spreading over is much smaller. In addition, not all spreading leads to errors in histograms. The spreading over from neighboring bins partially cancel out each other. So the overall error introduced by neglecting $\mathbf{x}_H$ can be small.

When $\mathbf{x}_H$ and $\mathbf{x}_L$ are not independent, (e.g. the pdf of $\mathbf{x}_H$ depend on the values that $\mathbf{x}_L$ takes), then the error may be more concentrated in certain bins than in others.

Approximate the histogram of LoF signal from LoF trickles. If $H(\mathbf{x}_L)$ is an acceptable approximation of $H(\mathbf{x}_L + \mathbf{x}_H)$, how can we compute $H(\mathbf{x}_L)$ efficiently? The archived trickle $\tilde{\mathbf{x}}_L$ is a downsampling of $\mathbf{x}_L$. Is it possible to approximate $H(\mathbf{x}_L)$ from the shorting length trickle $\tilde{\mathbf{x}}_L$ without fully reconstructing $\mathbf{x}_L$.

Note that upsampling of $\tilde{\mathbf{x}}_L$ (as defined in section 3.1) is the inverse of downsampling of $\mathbf{x}_L$. Since we are aggressive in terms of selecting cut off frequencies when filtering $\mathbf{x}$ to obtain $\mathbf{x}_L$, there are many levels of intermediate upsampling of $\tilde{\mathbf{x}}_L$ that progressively approximate $\mathbf{x}_L$.

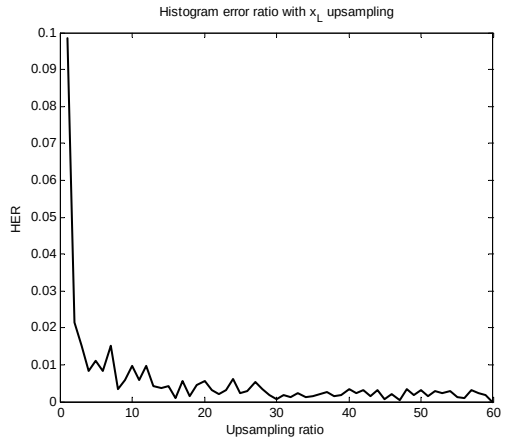

Figure 6: Histogram Error Ratio computed from upsampling the LoF trickle used in the running example.

Figure 6 shows the HER when use various rates to upsample the LoF trickle. We see that without upsampling, the histogram of the trickle directly gives about 10% error. A small upsampling, say less than 4 can significantly reduce the error to about 1%.

Ultimately, the resolution (i.e. bin size) and accuracy of histograms are application dependent. For example, for CPU utilization data, it does not give more information if we consider bin size less than 5%.

From the above discussions, we see the tradeoff between compression rate and the accuracy of answering histogram queries. The larger the cut off frequency is used to separate LoF, the better that the LoF signal approximates the original signal. The smaller threshold is used to detect spikes, the less blurring is introduced when eliminating HiF. However, the larger cut off frequency means more samples need to be archived in LoF trickle. Smaller threshold also causes more samples being classified as spikes.

5. CORRELATION QUERIES

Correlations are a useful measure of the similarity of two data streams and different insights are gained by performing correlation analysis on LoF, Spike, and Hi trickles separately. Intuitively this makes sense because the trickles correspond to different time-scales and behaviors. For example, LoF correlations may correspond to long-term trends while Spike and HiF correlations correspond to short-term behavioral relationships.

As we have shown in Sections 3, compressible signals such as the LoF and Spike trickles can be represented accurately from a relatively small number of samples and it follows that correlation analysis can be performed directly on their compressed representations. However, the HiF trickle is not assumed to be compressible, and hence the main challenge is to find a compressed version or "sketch" of the HiF trickle that is sufficient for approximating correlation analysis. Since compression is achieved by discarding some of the information content in the HiF trickle, any sketching method carries some risk that the resulting correlation estimates will be highly inaccurate. In this section, we show that random linear projections provide a robust method for sketching, and we analyze the tradeoff between accuracy in the correlation estimates and the length of the sketch.

To be explicit, the correlations we discuss are the sample correlations between two signals, say x_1 and x_2, over an analysis window of length N. Using the vector notation $\mathbf{x}_1 = [x_1(n), \cdots, x_1(n+N-1)]$ and $\mathbf{x}_2 = [x_2(n), \cdots, x_2(n+N-1)]$, the sample correlation is given by

$$\rho_{i,j} = \frac{\langle \mathbf{x}_1, \mathbf{x}_2 \rangle}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|}.$$

where $\langle \cdot, \cdot \rangle$ is the inner product and $\| \cdot \|$ is the Euclidian norm.

In the Cypress framework, the HiF trickles are buffered and processed in blocks of length N creating a sequence of vectors $\mathbf{x}_i \in \mathbb{R}^N$. The goal is to find a sketching method $f : \mathbb{R}^N \mapsto \mathbb{R}^K$ with $K << N$ that can be applied independently to each vector $\mathbf{x}_i$ such that for any pair (i, j) the correlation $\rho_{i,j}$ can be estimated accurately from the sketches $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$. The following sections discus linear sketching functions of the form $f(\mathbf{x}) = A\mathbf{x}$ where A is a $K \times N$ sketching matrix.

5.1 Risk of Downsampling

Uniform downsampling by a factor N/K can be expressed as a linear projection with a $K \times N$ matrix A where the rows of A correspond to K distinct rows of the $N \times N$ identity matrix. Hence it is clear that downsampling represents a particular instance of linear sketches.

If it is assumed that the elements of two streams $x_1(n)$ and $x_2(n)$ are drawn i.i.d. over n from a bivariate distribution, then the downsampled vectors $A\mathbf{x}_1$ and $A\mathbf{x}_2$ corresponds to

a set of K i.i.d. pairs. Hence, it is clear that accuracy of statistics computed on $A\mathbf{x}_1$ and $A\mathbf{x}_2$ depends only on the number of samples K and not on the length of the original vectors N.

Moreover, if the independence assumption above is relaxed to allow for sequences $x_i(n)$ with short term correlations, then it follows that uniform downsampling should have the same properties mentioned above provided that the duration between samples is long enough to ensure that they are sufficiently independent. We refer to either the i.i.d. or the identically distributed with short term correlation settings as *idealized noise*.

To the extent that the HiF trickles correspond to idealized noises, uniform downsampling provides an efficient and accurate sketching method for preserving sample correlations. However, if the trickles do not correspond to idealized noise, then there is a risk that downsampled sketches will "miss" the important contents in the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ the resulting estimate $\hat{\rho}_{1,2}$ will be very different from the true sample correlation $\rho_{1,2}$. To illustrate this point, consider the following example of non-stationary noise.

Consider the setting where the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ correspond to the HiF trickles of two servers S_1 and S_2 on the same day. Now, assume that the short term behaviors of S_1 and S_2 are independent in general, but are highly correlated during some event of interest E. Clearly, the sample correlation over the day depends on the how often the event E occurs. However, since these occurrences of E could be spread arbitrarily (i.e. not uniformly) throughout the day, there is a risk that uniform sampling will miss a significant portion of these occurrences. While one solution may be to use a non-uniform downsampling pattern, such a method has the risk that the samples will not be sufficiently independent.

As the above example helps to show, for any fixed linear projection A, there exists a class of signals for which A will perform badly, and hence robust guarantees are difficult. In the following section, we illustrate that one solution to this problem is to use randomized linear projections which offer good performance with known reliability.

5.2 Robust Guarantees from Random Projections

This section shows that random linear projections are a robust method of sketching with probabilistic error guarantees. To begin, it is useful to observe that for any (i, j) the sample correlation between $\mathbf{x}_i$ and $\mathbf{x}_j$ does not depend on the magnitudes $\|\mathbf{x}_i\|$ and $\|\mathbf{x}_j\|$. Hence, without loss of generality, it may be assumed that all signals are normalized to unit magnitude so that sample correlation is related to Euclidian distance by

$$\rho_{i,j} = 1 - \frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2. \tag{4}$$

A great deal of work [17, 15, 2, 6, 21, 20, 7, 16] has considered the problem of estimating pairwise distances from lower-dimensional projections. Below, we present a key result that is used in various proofs [2] of the Johnson - Lindenstrauss Lemma [17].

THEOREM 5.1. *[2] Given any sequence of vectors $\mathbf{x}_i \in \mathbb{R}^N$ and any $\epsilon > 0$, let A be a $K \times N$ matrix chosen independently of $\mathbf{x}_i$ with entries i.i.d. zero mean Gaussian with variance $1/d$ where*

$$K > \frac{2}{\epsilon^2/2 - \epsilon^3/3}\log(2/\delta). \tag{5}$$

Then for each pair (i, j) the bounds

$$(1-\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq \|A\mathbf{x}_i - A\mathbf{x}_j\|^2 \leq (1+\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 \tag{6}$$

hold with probability at least $1 - \delta$.

Thus, a suitably chosen set of random linear projections will, with high probability, preserve the Euclidian distance between any two vectors. Using the relation (4), it is straightforward to see that under the same conditions given in Theorem 5.1, the correlation estimate $\hat{\rho}_{i,j} = 1 - \frac{1}{2}\|A\mathbf{x}_i - A\mathbf{x}_j\|^2$ obeys the bound $|\hat{\rho}_{i,j} - \rho_{i,j}| < 2\epsilon$ with probability at least $1 - \delta$. While this bound is tight in the scaling sense as K becomes large, it is loose for the relatively small values of K needed to provide significant compression in the Cypress framework. In the following result we improves the error bound by a factor of two by considering the estimation of the correlation directly.

THEOREM 5.2. *Given any sequence of vectors $\mathbf{x}_i \in \mathbb{R}^N$ with unit norm and any $\epsilon > 0$, let A be a $K \times N$ matrix chosen independently of $\mathbf{x}_i$ with entries i.i.d. zero mean Gaussian with variance $1/K$ where K satisfies (5). Then, for each pair (i, j) the estimated correlation $\hat{\rho} = \langle A\mathbf{x}_i, A\mathbf{x}_j\rangle$ obeys $|\hat{\rho} - \rho| < \epsilon$ with probability at least $1 - \delta$.*

PROOF. The normalized vectors $\sqrt{\frac{D}{2(1+\rho)}}A(\mathbf{x}_i + \mathbf{x}_j)$ and $\sqrt{\frac{D}{2(1-\rho)}}A(\mathbf{x}_i - \mathbf{x}_j)$ have Gaussian distribution $\mathcal{N}(0, I_{d\times d})$ and are independent due to the orthogonality of $\mathbf{x}_i + \mathbf{x}_j$ and $\mathbf{x}_i - \mathbf{y}_j$. Hence, the random variables $Y_1 = \frac{D}{2(1+\rho)}\|A(\mathbf{x}_i + \mathbf{x}_j)\|^2$ and $Y_2 = \frac{D}{2(1-\rho)}\|A(\mathbf{x}_i - \mathbf{x}_j)\|^2$ are i.i.d. chi-square (χ^2) with D degrees of freedom, and the estimate can be written as

$$\begin{aligned}
\hat{\rho} &= \langle A\mathbf{x}_i, A\mathbf{x}_j\rangle \\
&= \frac{1}{4}\left[\|A(\mathbf{x}_i + \mathbf{x}_j)\|^2 - \|A(\mathbf{x}_i - \mathbf{y}_j)\|^2\right] \\
&= \frac{1}{2D}\left[(1+\rho)Y_1 - (1-\rho)Y_2\right] \\
&= \rho + \frac{1+\rho}{2}(Y_1/D - 1) - \frac{1-\rho}{2}(Y_2/D - 1)
\end{aligned}$$

Since a χ^2 variable with D degrees of freedom has mean D, it follows that $\mathbb{E}(\hat{\rho}) = \rho$. Using the fact that Y_1 and Y_2 are independent and applying a Chernoff bound directly to the estimate $\hat{\rho}$ shows that for all $\rho \in [-1, 1]$, the following bound holds

$$\begin{aligned}
&\Pr\left(\left|\frac{1+\rho}{2}(Y_1/D - 1) - \frac{1-\rho}{2}(Y_2/D - 1)\right| > \epsilon\right) \\
&< 2\exp(-D[\epsilon^2/4 - \epsilon^3/6])
\end{aligned}$$

Hence, if we choose D satisfying (5) then the stated result follows □

From the proof of Theorem 5.2, it also possible to get an approximate "upper bound" on the estimation error for Gaussian random projections. Note that if $|\rho| = 1$ then

$$\Pr(|\hat{\rho} - \rho| > \epsilon) = \Pr(|Z_K - \mathbb{E}(Z_K)| > \epsilon K) \tag{7}$$

where Z_K is a χ^2 random variable with K degrees of freedom. With some exception (such as very small values of ϵ), this behavior corresponds to the worst case choice of ρ, and thus Equation (7) provides an improved upper bound for various regimes of ϵ and K.

It is worth noting the correlation estimate $\langle A\mathbf{x}_i, A\mathbf{x}_j \rangle$ is used above for the convenience of getting a closed-form bound. In practice, normalization or truncation to the interval $[-1, 1]$ should improve the accuracy. Also, although Gaussian projections can be shown to attain good performance, other constructions are worth considering. For example, sparse or binary matrices allow for fast computation [2] and, as the next section show, matrices with orthonormal rows can guarantee that no correlation is underestimated.

5.3 Avoiding False Negatives

For a given signal $\mathbf{x}_i$, consider the problem of identifying all other signals $\mathbf{x}_j$, $j \neq i$, that are similar to $\mathbf{x}_i$ in the sense that their sample correlation $\rho_{i,j}$ exceeds some threshold ρ^*. For each estimate $\hat{\rho}_{i,j}$ there are two kinds of error events: a false negative occurs if the true correlation is large $\rho_{i,j} \geq \rho^*$ but $\hat{\rho}_{i,j} < \rho^*$ and a false positive occurs if the true correlation is small $\rho_{i,j} < \rho^*$ but $\hat{\rho}_{i,j} \geq \rho^*$. Generally, one may consider a tradeoff between the two types of errors by adjusting the degree to which $\hat{\rho}_{i,j}$ over or underestimates the true correlations.

In the setting where this similarity index is used a "first cut" metric to determine which signals $\mathbf{x}_j$ merit further consideration, it is important to avoid false negatives. For example, StatStream builds a grid structure with estimations of pair-wise correlations of signals, to prune the search space for highly correlated signal pairs. To use our techniques with a similar approach, the correlation estimations cannot have false negatives (but they may have some false positives). Note that a false negative occurs if the norm of the difference signal $\mathbf{y} = \mathbf{x}_i - \mathbf{x}_j$ increases after sketching, that is if $\|A\mathbf{y}\| > \|\mathbf{y}\|$. Since the ratio $\|A\mathbf{y}\|/\|\mathbf{y}\|$ is upper bounded by the largest singular value of A, denoted $s_1(A)$, it follows that false negatives will not occur if $s_1(A) \leq 1$. Note that any $K \times N$ sketching matrix with orthonormal rows, such as any subset of the Fourier matrix for example, has all K non-zero singular values equal to one and satisfies this condition.

For the i.i.d. Gaussian sampling matrix A considered in Theorem (5.2) the event $\|A\mathbf{y}\| > \|\mathbf{y}\|$ occurs with probability approximately one half because the variance of each element is normalized such that the rows a_k, $k = 1, \cdots, K$ of A have expected magnitude $\mathbb{E}\|a_k\|^2 = N/K$. If instead the variance of each element is $1/N$, then A is comparable to a sketching matrix with orthonormal rows in the sense that $\mathbb{E}\|a_k\|^2 = 1$ and $\mathbb{E}\langle a_k, a_l \rangle = 0$, for $k \neq l$. The probability of a false negative for these rescaled sampling matrices is easily upper bounded using using Theorem 5.2 with $\epsilon = N/K$.

6. EVALUATION

We evaluate the performance and accuracy of the Cypress framework using 800 data streams, a super set of the example described in section 2, over a week. There are 16M records in total. In some cases, we extrapolate the results to 50,000 servers, which is the target application size of Cypress. We first show the performance of filtering and compression in terms of execution time and compression ratio, then evaluate the performance and accuracy of running histogram and correlation queries. For execution time measurements, we use a PC with Intel Core 2 Duo 2.4GHz and 2GB memory, running Windows Vista.

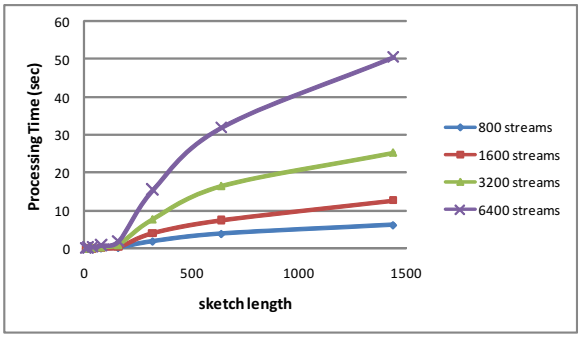

Figure 7: Execution time for sketching.

6.1 Multi-scale compression

There are several parameters that effect the performance of Cypress

- The number N_F of cut off frequency, and thus the downsampling size, which determines the compression ratio of the LoF trickle.
- The spike detection threshold T_s, which impacts on how many spikes will be archived and the bound on residuals.
- The sketch length K, which determines the compression ratio of the HiF trickle.

Obviously, these parameters also effect the accuracy of query answers.

Execution Time. Among the operations that Cypress takes, low pass filtering and thresholding are streaming operators, which consist of several dozens of multiplications, additions, subtractions, and comparisons for every new sample streamed in. Even with an implementation that bulk processes raw data once a day and uses ideal low pass filter by performing pairs of FFT and IFFT, the test PC finished filtering, downsampling, and spike detection for 800 data streams in 1.66 seconds.

Sketching is a relatively expensive operation for two reasons: 1) the random projection matrix is dense, and 2) a single projection from N dimension to K dimension requires $N \cdot K$ multiplications and $N \cdot K$ additions. Figure 7 shows the execution time for sketching 800, 1600, 3200, and 6400 streams, with sketch length 10, 20, 40, 80, 160, 320, 640, and 1440. We see a significant jump in execution time when the sketch length goes from 160 to 320. A possible cause is extra memory allocation or swapping.

Compression Ratio. The storage space used for archiving trickles is affected by both N_F, the number of LoF stored, T_s, the threshold used to detect spikes, and K, the sketch length. The contribution of LoF and sketches to the archiving space is solely determined by N_F and K, but the storage space for spikes can vary. More interestingly, a smaller T_s clearly causes more spikes to be detected. But a larger N_F can cause the LoF signal to approximate raw data better, which may reduce the number of spikes.

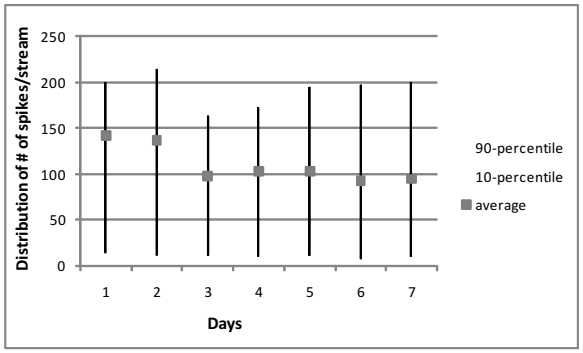

Figure 8: The distribution of the number of spikes detected in 800 streams, over 7 days.

We use the CPU utilization from the first day as the data set to compare storage space reduction. Table 1 shows the storage space required for the LoF and Spike trickles under various parameters. Assuming a sketch length of 144, the total compression ration, defined as the storage space used by raw data divided by that of Cypress is between 3× to 8×.

There is also a trade off among using different N_F and T_s. For example, in Table 1 using 384 LoF samples/day ($N_F = 192$) and 10% CPU utilization threshold requires more storage space than using 192 LoF samples/day ($N_F = 96$) but 8% threshold. How to choose these parameters also depends on query answer accuracy, which we will elaborate in later sections.

Table 1: Storage space requirement of different trickles as a percentage of raw data computed from 800 streams over 1 day. Assuming a sketch length of 144, we need additional 5% storage space for sketches. The total compression ratio is between 3× to 8×.

		Spike Storage		
N_F	LoF Storage	$T_s = 0.1$	$T_s = 0.08$	$T_s = 0.05$
24	1.67%	5.3%	10.3%	25%
48	3.3%	4.7%	9.6%	24%
96	6.6%	4.4%	9.4%	23.5%
192	13.2%	3.8%	6.6%	21.7%

Since spikes cause the variation in storage space requirement, we plot the distribution of the number of spikes detected in each stream (Figure 8) over 7 days. We use a loose threshold of $N_F = 24$, i.e. storing 48 LoF samples per day, and $T_s = 0.1$ (10% CPU utilization) in these results. We observe that in the CPU utilization data spikes are common. There are on average about 50 spikes per server per day.

6.2 Histogram Queries

We evaluate the accuracy of histogram queries using the data set. As discussed in section 4, two factors affect the accuracy of histograms: the upsampling length from LoF and the residual distribution.

Selecting bin size 10, we first vary the upsampling length when compress with 48, 96, and 192 LoF per day, as shown in Figure 9. We make three observations. First, on average, LoF + spikes give almost accurate histogram approximations, with less than 10% histogram error ratio (HER) error. However, in the worst case the error can be 20% to 30%. Second, for this data set, increasing the size of LoF trickles does not have significant impact on histogram accuracy. Finally, on average, increasing the number of samples used in computing histograms by upsampling improves histogram accuracy almost linearly.

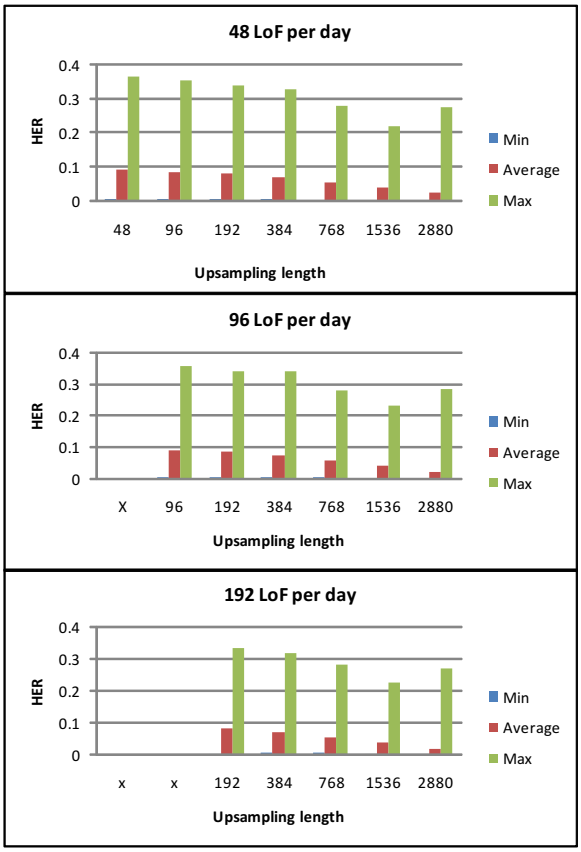

Figure 9: The effects of upsampling length on Histogram Error Rate (HER) when approximating histograms using LoF+Spikes.

Next, we evaluate the effect of varying the spike threshold T_s from 10, 8, to 5. Intuitively, the tighter the bound is, the smaller the error introduced to histogram approximation should be. Figure 10 is computed using 96 LoF. We observe that with low upsampling length, smaller T_s can significantly improve the worst case HER. However, the trend becomes counterintuitive when upsampling length is large. We believe that a small number of outliers cause the change in distribution.

6.3 Correlation Queries

Figure 11 shows the empirical probability that the sample correlation estimates sketches deviate more than $\epsilon = 0.1$ from the true sample correlations computed on the raw signals. A comparison is made between using Gaussian sketches and uniform downsampling on the HiF signals, a non-stationary subset of the HiF signals, and the uncompressed spike trickle. The results show that in general the HiF trickles in our data correspond to the "idealized" noise

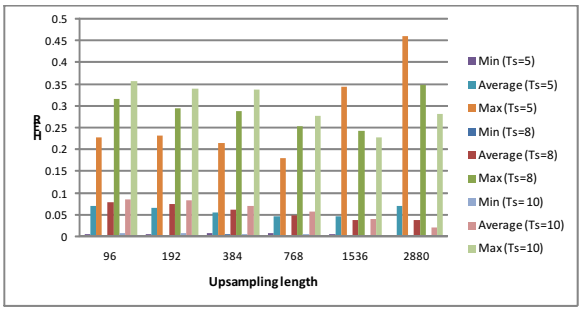

Figure 10: The effects of spike threshold on Histogram Error Ratio (HER) when approximating histograms using LoF+Spikes.

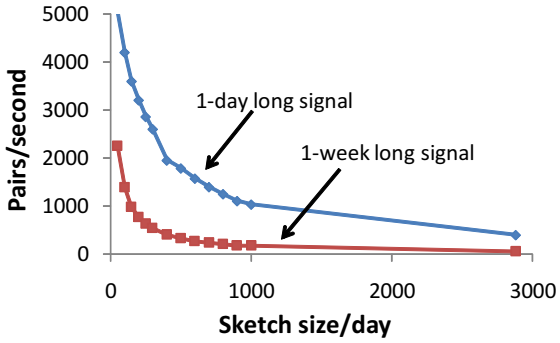

Figure 12: Throughput of pairwise correlation output as a function of different sketch size. A sketch size of 2880 represent the original signal.

assumptions discussed in Section 5, but that for a subset of non-stationary HiF signals the risk of uniform sampling can be large. Additionally, we include results for the "spiked noise" which corresponds to sketching the (uncompressed) spike trickle and also serves to illustrates the risk uniform downsampling. For comparison, we include the approximate upper bound (7) which follows from the proof of Theorem 5.2.

Note that both downsampling and random sketches have the same space requirement. These results show that for an additional sketching cost, random sketches have the property that their performance does not depend on the noise type, and is consistently below the bound (7). For example, by using a random sketch of size 500, the probability of estimation error exceeding 0.1 can be very small (close to 0.01), which is acceptable in our target applications. This shows the effectiveness of using random sketch, which is smaller than original signal, for computing correlation.

The small correlation estimation error from random sketch comes with the advantage of smaller latency of correlation estimation. To understand the benefit, we use a query that requires computing correlation of all pairs of signals in the system and measure the throughput in terms of the number of signal pairs whose correlation coefficients have been computed in a second. We consider different sketch lengths, including a sketch of length 2880 samples per day that represents the original uncompressed signal. We consider two cases where correlations are computed over 1-day long signal and over 1-week signal. Figure 12 shows the results. As shown, if the system computes correlation from original signals, it can compute correlation of around 60 pairs of 1-week long signals(or, 400 pairs of 1-day signals). In contrast, if Cypress is configured to compute correlations based on random sketches of length 300 per day, it can correlate around 550 pairs of 1-week long signals (or, 2600 pairs of 1-day long signals), giving more than a factor of 9 higher throughput than a system that uses raw signals.

The higher throughput of sketch-based correlation compared to raw-signal-based correlation comes from two sources. First, since correlation is computed on smaller signals (sketches), this involves a smaller computational overhead. Second, since in the sketch-based approach, less data needs to be read from disk, it has a smaller I/O overhead as well. Table 2 shows CPU and I/O overheads of computing correlation of a single pair of 1-day long signal. With a sketch of length 100, around 81% of total processing time is due to I/O latency. As the sketch size increases, both CPU and I/O latencies increase; however, I/O cost increases at a higher rate than CPU cost (e.g., I/O cost is 94% when correlation is computed from raw signals).

Table 2: CPU and I/O latency of computing pairwise correlation of two signals of length 2880 (1 day).

Sketch Length	CPU time (ms)	I/O time (ms)
100	0.04 (19%)	0.19 (81%)
500	0.06 (10%)	0.5 (90%)
1000	0.07 (8%)	0.89 (92%)
2880	0.14 (6%)	2.37 (94%)

7. RELATED WORK

Time series data mining is an active field of research. Several prior work has shown how to index large time series data [23, 26]. Cypress is significantly different from these systems in terms of its approach of separately compressing and indexing different frequency component of signals and in terms of the queries it support (e.g., correlation, histogram, etc.). Data structures and algorithms for supporting fast correlation computation have been studied in the context of streaming databases. In StatStream [22], correlations are approximated by transforming the signals to the frequency domain using DFT. The approach is similar to our correlation queries on LoF signals. In BRAID [21], geometric probing and curve fitting are used to compute correlations with time shifts, although each correlation coefficient is computed using the full signal.

Correlation analysis is a common statistical and data mining method for analyzing software system utilization. However, instead of solving specific applications, as in capacity planning [9] or performance debugging [3], we point out the distinct information contents in different frequency bands. So it is necessary to archive the information for long term analysis. Combining this notion with data reduction techniques, we can answer a broad spectrum of useful queries without significant increase in storage space.

The theoretical inspiration of Cypress comes from random projection and compressive sensing [17, 2, 8, 14]. Random projection (or sketches) has been used as an effective di-

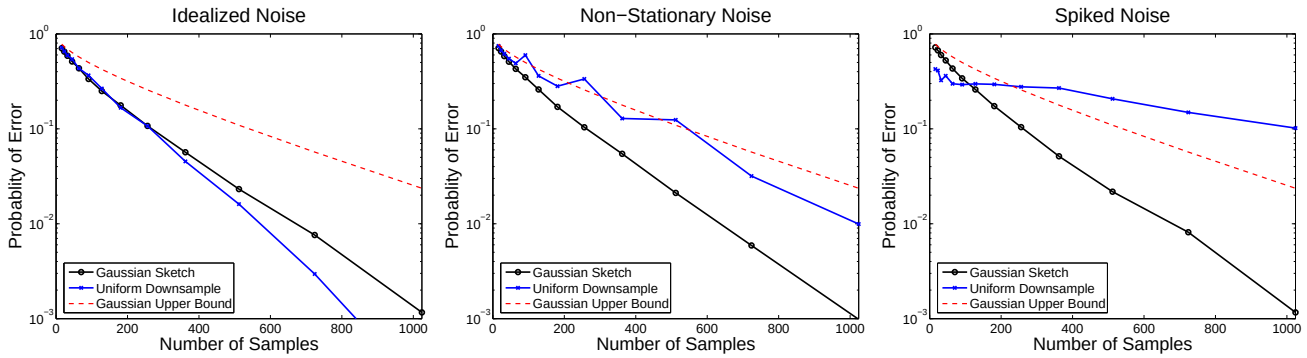

Figure 11: Empirical probability that the correlation estimate error exceeds $\epsilon = 0.1$. Comparison of Gaussian projections, uniform downsampling, and the upper bound (7) for three different types of HiF trickles.

mension reduction technique in data mining. For example, in [10], Fern et. al. apply random projection for point data clustering. In [4], the authors conducted empirical study to show that random projection out performs principle component analysis and discrete Cosine transform in image and text compression, in terms of introducing least distortion. In [24], sketches are used to maintain dynamic histograms over data streams. Compressive sensing techniques has recently been applied to collecting and processing computer system data streams. In [18], compressive sensing principle is used to derive an approximation framework for fine-grained network measurements. The focus of this paper is to provide a generic correlation analysis framework based on the random projection principle.

8. CONCLUSION

We have presented Cypress, a novel framework to archive and query massive time series streams. Cypress applies multi-scale analysis to decompose time series and to obtain sparse representations in various domains (e.g. frequency domain and time domain). Relying on the sparsity, the time series streams are archived with reduced storage space. We also showed that many statistical queries such as trend, histogram and correlations can be answered directly from compressed data rather than from reconstructed raw data. Our evaluation with server utilization data collected from real data centers showed significant benefit of our framework.

9. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, 2005.

[2] D. Achlioptas. Database-friendly random projections. In *ACM PODS*, 2001.

[3] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *ACM SOSP*, 2003.

[4] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *in Knowledge Discovery and Data Mining*, pages 245–250. ACM Press, 2001.

[5] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *USENIX NSDI*, 2008.

[6] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In *KDD*, 2005.

[7] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishan. Fast mining of massive tabular data via approximate distance computations. In *ICDE*, 2002.

[8] D. Donoho. Compressed sensing. *IEEE Trans. on Information Theory*, 52(4):1289 – 1306, 2006.

[9] X. Fan, W. dietrich Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Intl. Symp. on Computer Architecture*, pages 13–23, 2007.

[10] X. Z. Fern and C. E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *Intl. Conf. on Machine Learning*, 2003.

[11] S. Gandhi, S. Nath, S. Suri, , and J. Liu. GAMPS: Compressing Multi Sensor Data by Grouping and Amplitude Scaling . In *ACM SIGMOD*, 2009.

[12] M. Garofalakis and P. B. Gibbons. Wavelet Synopses with Error Guarantees. In *ACM SIGMOD*, 2002.

[13] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. One-pass wavelet decompositions of data streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):541–554, 2003.

[14] J. Haupt and R. Nowak. Signal reconstruction from noisy random projections. *IEEE Trans. on Information Theory*, 52(9):4036–4048, 2006.

[15] P. Indyk, N. Koudas, and S. Muthukrishan. Identifying representative trends in massive time series data sets using sketches. In *VLDB*, 2000.

[16] P. Indyk and R. Motwani. Approximate nearest neighbots: Towards removing the curse of dimensionality. In *ACM STOC*, 1998.

[17] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilber space. *Contemporary Mathematics*, 26:189–206, 1984.

[18] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: a novel counter architecture for per-flow measurement. In *ACM SIGMETRICS*, 2008.

[19] A. K. Menon, G. V. A. Pham, S. Chawla, and A. Viglas. An incremental data-stream sketch using

sparse random projections. In *SIAM International Conference on Data Mining*, 2007.

[20] S. Papdimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, 2005.

[21] Y. Sakurai, S. Papdimitriou, and C. Faloutsos. Braid: Stream mining through group lag correslations. In *SIGMOD*, June 2005.

[22] Y. Z. D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *In VLDB*, 2002.

[23] J. Shieh and E. Keogh. iSAX: indexing and mining terabyte sized time series. In *ACM SIGKDD*, 2008.

[24] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *ACM SIGMOD*, 2002.

[25] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *ACM SIGMOD*, 2008.

[26] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series. *The VLDB Journal*, 15(1):1–20, 2006.