On Computing Correlated Aggregates Over Continual Data Streams*

Johannes Gehrke
Cornell University
johannes@cs.cornell.edu

Flip Korn
AT&T Labs–Research
flip@research.att.com

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

ABSTRACT

In many applications from telephone fraud detection to network management, data arrives in a stream, and there is a need to maintain a variety of statistical summary information about a large number of customers in an online fashion. At present, such applications maintain basic aggregates such as running extrema values (`MIN`, `MAX`), averages, standard deviations, etc., that can be computed over data streams with limited space in a straightforward way. However, many applications require knowledge of more complex aggregates relating different attributes, so-called *correlated aggregates*. As an example, one might be interested in computing the percentage of international phone calls that are longer than the average duration of a domestic phone call. Exact computation of this aggregate requires multiple passes over the data stream, which is infeasible.

We propose single-pass techniques for approximate computation of correlated aggregates over both landmark and sliding window views of a data stream of tuples, using a very limited amount of space. We consider both the case where the independent aggregate (average duration in the example above) is an extrema value and the case where it is an average value, with any standard aggregate as the dependent aggregate; these can be used as building blocks for more sophisticated aggregates. We present an extensive experimental study based on some real and a wide variety of synthetic data sets to demonstrate the accuracy of our techniques. We show that this effectiveness is explained by the fact that our techniques exploit monotonicity and convergence properties of aggregates over data streams.

1. INTRODUCTION

In many applications from telephone fraud detection to network management, data arrives in a *stream*, and online decisions are made based on a "recently observed" portion of the stream. For example, telephone call records are generated for each call, at the end of the call. The stream of generated call records may be used by applications such as telephone fraud detection, which examines this stream for various patterns of potentially fraudulent calling behavior. As another example, router interfaces are periodically polled by network operators using SNMP to get a variety of performance data. The stream of polled SNMP data is then used by network monitoring and management applications to determine if an interface is down, router is inaccessible, *etc.*

The large volume of stream data (hundreds of millions of phone call records from tens of millions of customers per day; tens of millions of SNMP records from tens of thousands of router interfaces per day, *etc.*), and the online nature of the various applications that operate on such data, makes it imperative for the applications to compute and maintain a variety of statistical summary information in an online fashion. At present, such applications maintain basic aggregates such as running averages, standard deviations, *etc.*, that can be computed over data streams with limited storage in a straightforward way. However, more complex aggregates are often desired, especially when exploring correlations between attributes of tuples in the data stream; this application scenario allows users to specify *ad hoc* complex aggregates as the data stream flows by, and to request that results be computed and reported periodically. For example, for each telephone customer, what percentage of calls longer than the average duration are international calls? Or, for each router interface, how often is the total outbound traffic within, say, 50% of the maximum outbound traffic?

Correlated aggregates [6, 5, 4] provide a natural mechanism for the flexible composition of standard aggregates, that are useful for such applications. For example, $\texttt{COUNT}\{x : x > 0.5{*}\texttt{MAX}(x)\}$ operates on a multiset of x values, and computes the number of x values that are within 50% of the maximum x value in the multiset. Similarly, $\texttt{MAX}\{y : x < \texttt{AVG}(x)\}$ operates on a multiset of (x, y) tuples, and computes the maximum y value obtained from tuples where the x value is less than the average x value in the set; this value may not be the maximum y value in the entire multiset. Prior work [6, 5, 4] has considered only the exact computation of correlated aggregates over finite data sets. In general, this exact computation requires multiple passes over the data set. The first pass is needed to determine the independent aggregate (average duration of calls, or the maximum outbound traffic, in our applications). The second pass is needed to determine the dependent aggregate (percentage of calls longer than the

*Work of Johannes Gehrke supported in part by a gift from Microsoft Corporation and an IBM Faculty Development Award.

ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

(previously computed) value, or the number of times the total traffic is within 50% of the (previously computed) value. For data streams of large volume, such exact computation is not feasible, and providing a quick approximate answer will have to suffice. The problem of *efficient approximate computation of correlated aggregates over data streams* is the focus of this paper, and we make the following main contributions:

- We classify correlated aggregates over data streams, based on their *scope* and the nature of the *independent aggregate*.

  We illustrate via examples that two natural alternatives for the scope are *landmark windows*, and *sliding windows*. Landmark windows identify certain landmarks in the data stream, and the aggregate value at a point is defined with respect to the tuples from the immediately-preceding landmark until the current point; for example, correlated aggregates on a daily basis, or from the point where a user requested the computation of an *ad hoc* aggregate. Sliding windows are typically of a fixed width, and the aggregate value at a point is defined with respect to the tuples that precede this point and are within this width; for example, correlated aggregates over the previous 60 minutes.

  We consider both the case where the independent aggregate is an extrema value (`MIN`, `MAX`) and the case where it is an average value; any standard aggregate can be used as a dependent aggregate.

- Since correlated aggregates typically cannot be computed exactly in a single pass (equivalently, the exact computation of correlated aggregates in a single pass requires an unbounded buffer size), it is natural to expect that the accuracy of a single-pass, approximate computation of correlated aggregates directly depends on the additional "buffer" space available for maintaining an estimate of the aggregate value.

  Histograms have been used widely both in the research literature and in commercial DBMSs for quickly estimating approximate statistics about underlying data domains, in limited space. Hence, the use of traditional histograms, with few buckets (depending on the buffer space available per aggregate), is a viable strawman for our problem. We observed two limitations of the use of standard histograms with respect to our problem:

  - First, at a point during the stream computation, we may be interested only in a focused small, subinterval of the entire range of values from the underlying domain. Since traditional histogram techniques (equiwidth, equidepth, *etc.*) allocate buckets over the *entire range of values in the domain*, they often waste buckets allocating them to regions that cannot (or are unlikely to) affect subsequent results.
  - Second, the region of interest (*e.g.*, the tuples whose x values are less than $\texttt{AVG}(x)$) may shift while the data stream is passing by. Traditional work on histograms has not addressed this issue, to the best of our knowledge.

  These two issues motivate us to seek novel solutions to the problem of accurately maintaining histogram information over a stream of data records, which can effectively deal with a dynamically changing (shifting, contracting, expanding) region of interest.

- We propose single-pass techniques for the approximate computation of correlated aggregates that adapt the histogram bucket boundaries to a dynamically changing region of interest in one of two ways, over both landmark and sliding window data streams of tuples: (i) a *wholesale* approach that can change each and every bucket boundary in response to a change in the region of interest; and (ii) a *piecemeal* approach that is more conservative and changes bucket boundaries only when absolutely necessary.

  We show how properties of the independent aggregate (monotonicity of extrema, and convergence of average) can be used effectively for the approximate computation of correlated aggregates, over landmark window data streams.

  Over sliding windows, our techniques need to deal with the simultaneous inclusion of a data tuple and exclusion of another data tuple, in efficiently computing the correlated aggregate. We achieve this by combining the approach we use for cumulative data streams with novel incremental mechanisms for approximately maintaining extrema aggregates in a sliding window.

- We complement our algorithmic analysis with an extensive experimental study based on some real and some synthetic data sets to demonstrate the accuracy of our techniques for approximate computation of correlated aggregates. The main conclusions of our study are as follows:

  - Use of focused histograms to estimate correlated aggregates over data streams is demonstrably superior, in a large variety of cases, to the use of traditional histograms.
  - A simple "piecemeal" strategy, which maintains uniformly-spaced bucket boundaries in its region of interest, and changes bucket boundaries only at the extremities of the region, when the region of interest changes, is the strategy of choice across a wide variation in window scopes, nature of aggregate, and type of data stream.

The principal consequence of our study is that we have a robust approach for the approximate computation of the important class of correlated aggregates over streaming data. To the best of our knowledge, ours is the first work on evaluating this important class of aggregates over streaming data.

2. PROBLEM DEFINITION

2.1 Data Streams

Consider a relational schema R with attributes $X_1, \ldots, X_k$ where attribute X_i has $\texttt{dom}(X_i)$. We call $\texttt{att}(R) \stackrel{\text{def}}{=} \texttt{dom}(R) \stackrel{\text{def}}{=} \texttt{dom}(X_1) \times \cdots \times \texttt{dom}(X_k)$ the *attribute space of* R. Let R be a relational schema with attribute space $\texttt{att}(R)$. We call

a function $O : \mathbf{N} \to \mathtt{att}(R)$ an *ordering of* R. A *sequence* is a tuple $S(R, O)$ where R is a relational schema and O is an ordering of R. Given a sequence $S(R_S, O_S)$, we also refer to the natural numbers as *positions*, and we use $S[i]$ for $O_S(i)$, the record at the ith position.

Our model of computation is similar to the model introduced by Henzinger, Raghavan, and Rajagopalan [19]. It contains a single input sequence S_{in} and a single output sequence S_{out}, and the model has as single parameter m, the amount of space available. Computation in our model proceeds in steps, and each computation step consists of three substeps. Consider the ith computation step. In the first substep, we read $S_{in}[i]$ from the input sequence S_{in} into a memory location; in the second substep, we perform an unlimited amount of computation in memory; and in the third substep we write into the ith position of the output sequence $S_{out}[i]$. We call an algorithm for our model of computation a *stream algorithm*. Thus, our algorithms map input streams into output streams, which could be used for further processing.

We consider stream algorithms for aggregate computations. A *stream aggregate* has three components: A *scalar aggregate function* $AGG : 2^{\mathbf{R}} \to \mathbf{R}$, a *scope function* $scope : \mathbf{N} \to 2^{\mathbf{N}}$, and a *selection predicate* P. Given an input sequence S_{in}, a stream aggregate operator $Agg(AGG, scope, P)$ returns the sequence S_{out} such that

$$\begin{aligned} S_{out}[i] \quad = \quad & AGG\{S_{in}[j].X_l \mid j \in scope(i) \wedge \\ & P(S_{in}[j], S_{in}[scope(i)])\} \end{aligned}$$

where $S_{in}[scope(i)] \stackrel{\text{def}}{=} \{S_{in}[j] \mid j \in scope(i)\}$ and X_l is an attribute of R.

Three particular types of scope functions are especially interesting. We call the scope function $fScope : \mathbf{N} \to 2^{\mathbf{N}}$ such that $fScope(i) = \{1, \ldots, i\}$ for $i \in \mathbf{N}$ a *full window scope*, and we call the scope function $swScope_w$ such that $swScope_w(i) = \{\max(1, i-w+1), \max(1, i-w+2), \ldots, i\}$ a *sliding window scope of size* w. A full window scope is just a special case of a *landmark window scope*. A landmark window scope takes as input a *landmark set* $S = \{s_1, s_2, \ldots\}$. Given such a set S and a position i, $lmScope(S, i) = \{s_j, s_j + 1, \ldots, i-1, i\}$, where s_j is the largest position in S that is $\leq i$ where S is understood from the context. We omitted and simply use $lmScope(i)$. In this paper, we concentrate on sliding window scopes and on landmark window scopes.

Consider the stream aggregate operator $Agg(AGG, scope, P)$. If the selection predicate P does not contain any aggregate function, then we call Agg a *level* 0 stream aggregate operator. Recursively, let Agg be a level i stream aggregate operator. Then $Agg'(AGG', scope', P')$ is a level $i+1$ stream aggregate operator if

$$\begin{aligned} S_{out}[i] \quad = \quad & AGG'\{S_{in}[j].X_l \mid j \in scope'(i) \wedge \\ & P'(S_{in}[j], Agg(S_{in})[i])\} \end{aligned}$$

Our notion of the level of a stream aggregate lets us relate stream aggregates to regular queries over a static relation. A level i stream aggregate can be evaluated over a static relation in at most $i+1$ scans. Note that our notation of a level i stream aggregate is purely syntactic. There are level i stream aggregates that have equivalent level 0 aggregates for any i. Establishing such equivalences is outside the scope of this paper.

Consider the following application scenario from the telecommunications industry. Our data stream contains information about phone calls, captured in the following schema:

`CallDetail (origin, dialed, time, duration, isIntl)`

The attributes `origin` and `dialed` contain the originating and destination phone number of the call, respectively; the attributes `time` and `duration` denote the start time and duration of the phone call, respectively; the attribute `isIntl` indicates whether the call was an international phone call. We are interested in computing the following stream aggregates.

Example 1: At any point in time, we would like to compute the number of international calls over the last two months that took longer than 10 minutes. This is an example of a level 0 stream aggregate with a window scope of two months. With respect to our notation, this stream aggregate can be expressed as follows:

$$\begin{aligned} S_{out}[i] \quad = \quad & \mathtt{COUNT}\{S_{in}[j].\mathtt{origin} \mid j \in swScope(i) \wedge \\ & S_{in}[j].\mathtt{isIntl} = 1 \wedge S_{in}[j].\mathtt{duration} > 10\}, \end{aligned}$$

where $swScope$ is the appropriate sliding window scope that restricts relevant records from the input sequence S_{in} to the last two months. □

Example 2: At any point in time, we would like to find the number of international calls this year that were longer than the average call duration. Note that if `CallDetail` were a regular relation, then evaluation of this query would require two passes over the (materialized) relation. In the first pass, we would compute the average call duration d, and then in the second pass we would compute the number of international calls that have a duration longer than d. This is an example of a level 1 stream aggregate with a landmark window scope; the landmark set consists of the beginnings of each year. With respect to our notation, this stream aggregate can be expressed as follows:

$$\begin{aligned} S_{out}[i] \quad = \quad & \mathtt{COUNT}\{S_{in}[j].\mathtt{origin} \mid j \in lmScope(i) \wedge \\ & S_{in}[j].\mathtt{isIntl} = 1 \wedge S_{in}[j].\mathtt{duration} \\ & \geq \mathtt{AVG}\{S_{in}[k].\mathtt{duration} \mid k \in lmScope(i)\}\}, \end{aligned}$$

where $lmScope$ is the appropriate landmark window scope that restricts relevant records from the input sequence S_{in} to the current year. □

Example 3: At any point in time, we would like to find the number of international calls whose duration was within 10% of the call with the longest duration with respect to the last two weeks. This is an example of a level 1 stream aggregate with a window scope of two weeks. With respect to our notation, this stream aggregate can be expressed as follows:

$$\begin{aligned} S_{out}[i] \quad = \quad & \mathtt{COUNT}\{S_{in}[j].\mathtt{origin} \mid j \in swScope(i) \wedge \\ & S_{in}[j].\mathtt{isIntl} = 1 \wedge S_{in}[j].\mathtt{duration} \\ & \geq 0.9 \cdot \mathtt{MAX}\{S_{in}[k].\mathtt{duration} \mid k \in swScope(i)\}\}, \end{aligned}$$

where $swScope$ is the appropriate sliding window scope that restricts relevant records from the input sequence S_{in} to the last two weeks. □

If the amount of available space m is infinite, then we can compute S_{out} exactly for any stream aggregate through the

following simple algorithm: At step i, we read $S_{in}[i]$ into memory location $M[i]$. We then compute the exact value of the stream aggregate $S_{out}[i]$ using the copy of the input stream being stored and store the output in $S_{out}[i]$. The focus of this paper is on algorithms for computing stream aggregates with a given *constant* amount of space.[1]

In this paper, we focus on level 1 stream aggregates for a relational schema $R(X, Y)$ with two numerical attributes. Specifically, we consider stream aggregates of the form:

$$\begin{aligned} S_{out}[i] &= AGG\text{-}D\{S_{in}[j].Y \mid j \in scope(i) \wedge \\ &\quad P(S_{in}[j].X, AGG\text{-}I\{S_{in}[k].X \mid k \in scope(i)\})\}, \end{aligned}$$

where $AGG\text{-}D$ and $AGG\text{-}I$ are aggregate functions, *scope* is a scope function, and P is a simple boolean selection predicate. We call $AGG\text{-}D$ the *dependent aggregate*, and we call $AGG\text{-}I$ the *independent aggregate*.

As concrete instantiations, we concentrate in this paper on the following three types of stream aggregates:

- The independent aggregate is either `MIN` or `MAX`. An example instantiation of a stream aggregate operator produces records of the output sequence S_{out} that are computed as follows:

$$\begin{aligned} S_{out}[i] &= AGG\text{-}D\{S_{in}[j].Y \mid j \in scope(i) \wedge \\ &\quad \texttt{MIN}\{S_{in}[k].X \mid k \in scope(i)\} \leq \\ &\quad S_{in}[j].X \leq \\ &\quad (1+\epsilon) \cdot \texttt{MIN}\{S_{in}[k].X \mid k \in scope(i)\}\}. \end{aligned}$$

- The independent aggregate is `AVG`. An example instantiation of a stream aggregate operator produces records of the output sequence S_{out} that are computed as follows:

$$\begin{aligned} S_{out}[i] &= AGG\text{-}D\{S_{in}[j].Y \mid j \in scope(i) \wedge \\ &\quad S_{in}[j].X \geq \texttt{AVG}\{S_{in}[k].X \mid k \in scope(i)\}\}. \end{aligned}$$

2.2 Properties of Aggregate Functions

Let us introduce some properties of stream aggregate operators. These properties motivate our algorithms in Section 3. We call a stream aggregate $Agg(AGG, scope, P)$ *monotonic* if

$$\forall i \in \mathbf{N} : S_{out}[i+1] \leq S_{out}[i] \ \vee \ \forall i \in \mathbf{N} : S_{out}[i+1] \geq S_{out}[i]$$

We will see later that some stream aggregates under landmark scope with `MIN` or `MAX` as the independent aggregate are monotonic. However, this is not the case when `AVG` is the independent aggregate. In this case, we can give tight confidence bounds on the difference between the currently computed value of the aggregate and the (theoretical) expected value μ of the underlying distribution, depending on the step i. Define

$$\widehat{\mu}_i \stackrel{\text{def}}{=} \frac{1}{i}\sum_{j=1}^{i} S_{in}[j], \text{ and } \widehat{\sigma}_i^2 = \frac{1}{i}\sum_{j=1}^{i}(S_{in}[j] - \widehat{\mu}_i)^2$$

[1] If we assume that our algorithms use only a constant amount of space, we neglect here the logarithmic growth of the number of bits that results from computations on very long input sequences. Such sequences require our algorithms to run for many steps and potentially require the storage of very large numbers that grow beyond the precision possible by 32 or 64 bit architecture. We will disregard this logarithmic growth in our space considerations since we do not believe that it is important in practice.

After the ith step, the running value of the average is given by $\widehat{\mu}_i$. Then, by the standard central limit theorem for i.i.d. random variables,

$$\frac{\sqrt{i}(\widehat{\mu}_i - \mu)}{\widehat{\sigma}_i} \Rightarrow N(0,1),$$

as $i \to \infty$, where $\Rightarrow$ denotes convergence in distribution, and $N(0,1)$ denotes a random variable with normal distribution, mean 0 and variance 1. Thus for large values of i, we know that for $\epsilon > 0$

$$P(|\widehat{\mu}_i - \mu| \leq \epsilon) \approx 2 \cdot \Phi(\frac{\epsilon\sqrt{i}}{\widehat{\sigma}_i}) - 1,$$

where Φ is the standardized normal probability distribution.

2.3 Quality Measures

Since our focus is on complex stream aggregates that cannot be computed exactly with a constant amount of space, we need to quantitatively differentiate between our approximation strategies proposed in Sections 3 and 4. Both for landmark scope and for sliding window scope, we would like our algorithms to approximate the exact value as well as possible at every position i of the stream. Let $Agg(AGG, scope, P)$ be a stream aggregate. Define the value of the stream of exact answers S_{exact} at step i as follows:

$$S_{exact}[i] \stackrel{\text{def}}{=} AGG\{S_{in}[j].X_l \mid j \in scope(i) \wedge P(S_{in}, j, scope)\}.$$

Given the exact answer of the aggregate computation, we can define the root mean-squared error $RMSE_n$ at step n as follows:

$$RMSE_n \stackrel{\text{def}}{=} \sqrt{\frac{1}{|scope(i)|}\sum_{j \in scope(i)}(S_{out}[j] - S_{exact}[j])^2}.$$

3. LANDMARK WINDOW ALGORITHMS

In this section we describe algorithms that maintain correlated aggregates for data streams over a landmark window. Our algorithms exploit the monotonicity and convergence properties discussed in Section 2. We then present some experimental results on real and synthetic data streams to validate the effectiveness of our algorithms.

3.1 Description

Here we propose algorithms for maintaining correlated aggregates over landmark-based windows, where the independent aggregate is either an extrema or `AVG`. Our focus is on (one-sided) correlations such as $\texttt{COUNT}\{y : x < (1+\epsilon) * \texttt{MIN}(x)\}$ and $\texttt{COUNT}\{y : x > \texttt{AVG}(x)\}$, although it is straightforward how to extend our techniques to deal with two-sided correlations such as $\texttt{COUNT}\{y : (\texttt{AVG}(x) - \epsilon) < x < (\texttt{AVG}(x) + \epsilon)\}$. For exposition, we describe the case where `COUNT` is used as the dependent aggregate; our techniques easily extend to the case where `SUM` is the dependent aggregate.

Our solution involves the use of histograms as a summary data structure for the incoming data tuples $S_{in}[i]$. The histograms contain an ordered set of m adjacent buckets ordered by abscissae over the x-axis, and are of the form $\langle (v_1, f_1), (v_2, f_2), \ldots, (v_m, f_m)\rangle$. Aggregates are approximated from the histograms in a straightforward way, by estimating the overlap with the existing buckets. The

approximation error in correlated aggregates comes from the truncation of a single bucket (the bucket containing $(1+\epsilon) * \texttt{MIN}(x)$; the bucket containing the mean for `AVG`). Some assumption (*e.g.*, local uniformity) is required to estimate the number of data tuples in a subrange of a single bucket; however, upper- or lower-bounds can be reported based on counting or discarding the entire bucket, respectively. It is beneficial to allocate most of the space to buckets in the region where approximation error is expected. We exploit the properties described in the previous section to design appropriate strategies to do this. In particular, we exploit the monotonicity of extremas when `MIN` or `MAX` is the independent aggregate, and the Central Limit Theorem when `AVG` is the independent aggregate.

At any given moment, our histogram buckets are tuned for up-to-the-moment data tuples from the stream. However, an existing bucket allocation can degrade from the given partitioning policy (uniform, quantiled) with the arrival of new tuples. We describe two bucket reallocation strategies to compensate for this: *wholesale* and *piecemeal*. The wholesale approach completely revises the set of buckets from scratch at each step based on a new partitioning; the reallocation requires the use of some form of interpolation (*e.g.*, based on uniformity). The piecemeal approach tries to preserve the existing bucket infrastructure while staying consistent with the given bucketing policy, to reduce the approximation error resulting from repeated application of interpolation based on the uniformity assumption. For each reallocation approach, we discuss two bucket partitioning policies: the first partitions buckets uniformly, whereas the second tries to maintain quantiles.

3.1.1 Sketch of Our Algorithms

Our algorithms follow the outline given in Figure 1. The subroutines InitializeHistogram and ReallocateHistogram differ based on which independent aggregate is supplied. Also, the conditions for which they are applied are different. First, we describe what these subroutines are for extrema; then we describe what they are for `AVG`.

Landmark Window Algorithm:
Input: data stream and aggregate query
Output: result stream
  $H \leftarrow$ InitializeHistogram(m);
  for $i = 1$ to w do
    read tuple $S_{in}[i]$;
    if ($condition_1$) then $H \leftarrow$ InitializeHistogram(m);
    if ($condition_2$) then ReallocateHistogram(H);
    add tuple to appropriate histogram bucket;

Figure 1: Landmark Window Algorithm

3.1.2 Extrema as the Independent Aggregate

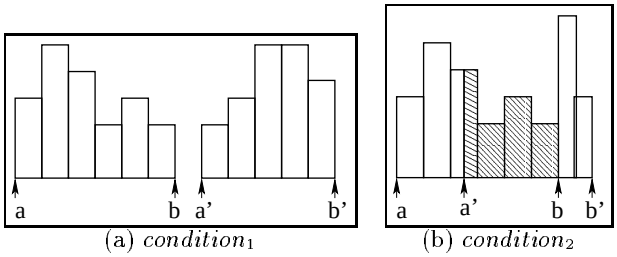
Let $[a, b]$ represent the running range predicate, *e.g.*, $[a, b] = [\min, (1+\epsilon) * \min]$. Given a new minima (or maxima), the range shifts to $[a', b']$. InitializeHistogram is invoked when $condition_1$ is satisfied, which occurs when $(b' \leq a)$ for the case of `MIN`, and $(a' \geq b)$ for the case of `MAX`. When this occurs, we can reinitialize the histogram and toss out all the tuples seen up to that point without incurring any approximation error. ReallocateHistogram is invoked when $condition_2$ is satisfied, which occurs when $(a' \neq a)$ or $(b' \neq b)$. When this occurs, we can truncate the histogram; the resulting approximation error is not cumulative. Figure 2 illustrates the conditions for `MAX`.

InitializeHistogram: Given the first m tuples $S_{in}[i]$ that arrive, we determine the range $[a, b]$ from the $S_{in}[i].X$-values. However, some of these tuples may be outside this range, so we can purge them from memory. We continue to read tuples, monotonically shifting the range $[a, b]$ based on the arriving x-values, until exactly m tuples remain after the purges.

PartitionHistogram: Uniform partitioning is straightforward: let $v_j = a + j * \frac{b-a}{m}$.

ReallocateHistogram: There are two general approaches for reallocating buckets. In the wholesale approach, we distribute frequences from the old histogram partitioning to the new one; each new frequency is determined by a weighted linear combination of old frequencies. Figure 3(a) gives the pseudocode. In the piecemeal approach, when a new range $[a', b']$ causes some buckets to be truncated, we reallocate the space for new buckets in a manner that best preserves the partitioning policy. Figure 3(b) gives the pseudocode.

Quantiles: In the InitializeHistogram step, we simply sort the first m incoming tuples by x-value. In PartitionHistogram, to get quantiles we start with (v_j, f_j) and determine $(v'_j, \overline{f})$, where $\overline{f} = \frac{1}{m}\sum_{j=1}^{m} f_j$ based on local uniformity assumptions. In the piecemeal approach, buckets can quickly become unbalanced. Our strategy for preserving the quantiles involves occasional merging and splitting of buckets, and is similar to the techniques used in [14]. We periodically check to see when two adjacent buckets can be merged at the same time as a single bucket split. If there is a net gain in improvement, then we perform this merge-split "swap". A merge operation takes two adjacent buckets, (v_j, f_j) and (v_{j+1}, f_{j+1}), and creates a new bucket $(v_j, f_j + f_{j+1})$; a split operation takes a bucket (v_j, f_j) and creates two buckets $(v_j, \frac{f_j}{2})$ and $(v_{j+1}, \frac{f_j}{2})$. We use the variance of the frequencies, which is the standard measure of "goodness" for a quantiled partitioning [14]: $Var(H) = \frac{1}{m}\sum_j (f_j - \overline{f})^2$ where $\overline{f} = \frac{1}{m}\sum f_j$.



(a) $condition_1$ (b) $condition_2$

Figure 2: Conditions when (a) InitializeHistogram and (b) ReallocateHistogram are called.

3.1.3 `AVG` as the Independent Aggregate

Assume that currently the query range is $[\widehat{\mu}_n, \max]$, where

WholesaleReallocate:
Input: old histogram $H_{old} = \{(v_j, f_j)\}$ in $[a, b]$
Output: new histogram $H_{new} = \{(v'_k, f'_k)\}$ in $[a', b']$
$H_{new} \leftarrow$ PartitionHistogram(H_{old});
if $(a' < a)$ then
  while $(v'_k < v_j)$ $k++$;
else
  while $(v_j < v'_k)$ $j++$;
while $(j < m)$ and $(k < m)$ do
  $f'_k += f_j * \frac{\min\{v_{j+1}, v_{k+1}\} - \max\{v_j, v_k\}}{v_{j+1} - v_j}$
  if $(v'_{k+1} > v_{j+1})$ $j++$;
  else $k++$;

PiecemealReallocate:
Input: old histogram $H_{old} = (v_j, f_j)$ in $[a, b]$
Output: new histogram $H_{new} = (v'_k, f'_k)$ in $[a', b']$
$k \leftarrow j$ such that $b' \in [v_j, v_{j+1}]$;
$v'_k = b'$;
$f_k* = \frac{b - v_k}{v_{k+1} - v_k}$;
keep buckets $\langle (v_1, f_1), ..., (v_k, f_k) \rangle$;
split remaining buckets according to max widths;

Figure 3: Wholesale and Piecemeal Algorithms

$\widehat{\mu}_n = \frac{1}{n}\sum_i^n S_{in}[i].X$ is the running mean from tuples $S_{in}[i]$ that have arrived up to step n. The arrival of a new tuple may cause $\widehat{\mu}_{n+1}$ to shift from $\widehat{\mu}_n$. However, the Central Limit Theorem gives us some indication of the behavior of $\widehat{\mu}_{n+1}$. In particular, it tells us to expect that $\widehat{\mu}_{n+1}$ will remain within the range $[\widehat{\mu}_n - \frac{\widehat{\sigma}_n}{\sqrt{n}}, \widehat{\mu}_n + \frac{\widehat{\sigma}_n}{\sqrt{n}}]$ with 68% probability.[2] Thus, we keep histogram buckets at $(\min, (\widehat{\mu}_n - \frac{\widehat{\sigma}_n}{\sqrt{n}}), \ldots, (\widehat{\mu}_n + \frac{\widehat{\sigma}_n}{\sqrt{n}}), \max)$ where the bucket locations in between are determined by the partitioning policy. InitializeHistogram is invoked once initially, but $condition_1$ is null. ReallocateHistogram is invoked when $condition_2$ is satisfied, which occurs whenever the mean shifts.

InitializeHistogram: Read the first m tuples $S_{in}[i]$ and compute $\widehat{\mu}_n = \frac{1}{m}\sum_i^m S_{in}[i].X$.

PartitionHistogram: We consider two strategies: one that partitions the subinterval $[\widehat{\mu}_n - \frac{\widehat{\sigma}_n}{\sqrt{n}}, \widehat{\mu}_n + \frac{\widehat{\sigma}_n}{\sqrt{n}}]$ uniformly, and one that partitions the interval according to the quantiles of the normal distribution with mean $\widehat{\mu}_n$ and standard deviation $\frac{\widehat{\sigma}_n}{\sqrt{n}}$.

ReallocateHistogram: The subroutines for the wholesale and piecemeal approaches are very similar. The only difference is that the reallocation of bucket frequences occurs within the subinterval $[\widehat{\mu}_n - \frac{\widehat{\sigma}_n}{\sqrt{n}}, \widehat{\mu}_n + \frac{\widehat{\sigma}_n}{\sqrt{n}}]$.

Quantiles: The details are the same as for extrema.

3.2 Experiments

We ran an extensive set of experiments to understand the following questions:

- How useful is it to use a summary data structure (in this case histograms) to maintain correlated aggregates as opposed to employing a "memoryless" algorithm? We ascertain the answer to this question by investigating the behavior a simple heuristic compared to histogram-based methods.
- How do our proposed methods compare with known histogram techniques, in particular, equiwidth and equidepth histograms? We demonstrate the impact of designing methods that specifically deal with the focused subranges involved in answering correlated aggregates, rather than for the *a priori* fixed ranges that existing histogram methods consider.
- Which approaches work well for our techniques? We have considered two general strategies, wholesale and piecemeal, and within each we have considered uniform and quantiled bucketing policies.

First we explain the experimental setup. Then we consider the answers to these questions in the context of the subsections that report on the accuracy for extrema and `AVG` as the independent aggregate.

3.2.1 Experimental Setup

Data: We used two real data sets: USAGE, usage data of 20K customers from AT&T; and MGCTY, lat/long of 65K road crossings in Montgomery County, MD.[3] We also used two synthetic data sets: ZIPF, a Zipfian distribution of points with $\lambda = 7$; and MULTIFRAC, a binomial multifractal obeying the "80-20 law".[4] The real data sets are ordered the way they were originally obtained; the synthetic data sets were generated in random order.
Queries: We tested correlated queries with both monotonic and non-monotonic independent aggregates. In particular, for monotonic aggregates we used `MIN` as the independent aggregate with the predicate $x < (1 + \epsilon) * \texttt{MIN}(x)$. For non-monotonic queries we used `AVG` as the independent aggregate with the predicate $x > \texttt{AVG}(x)$. We used `COUNT` and `SUM` as the dependent aggregates in both cases.
Quality Measure: To measure the accuracy, we used the formula given in Section 2:

$$RMSE_n = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(S_{out}[i] - S_{exact}[i])^2}$$

Competing Methods: As a frame of reference, we used two simple heuristics to maintain a running independent aggregate value and either (i) reset the count or (ii) continue to add to the existing one, when a new extrema value is encountered; this gives a lower- and upper-bound on the exact count, respectively. Among the existing methods, we computed "true" equiwidth and equidepth histograms, which required a single pass and multiple passes, respectively, at each time step. Clearly, this is not feasible in practice – we have given them an unfair advantage.[5] For our techniques, we consider both the wholesale and piecemeal strategies, each

[2] While our discussion uses a confidence interval of one standard deviation, this is a tunable parameter.

[3] Available at `http://www.esri.com/data/online/tiger`.

[4] Recent studies of network traffic data have shown that they are modeled well using multifractals [11].

[5] Note that the recent single-pass approximate quantile algorithms of [2, 20, 21] are designed for *offline* computation and, in any case, would likely give less accurate results than an exact equidepth histogram.

with uniform and quantiling partitioning policies. We call our methods `wholesale-uniform`, `wholesale-quantile`, `piecemeal-uniform`, and `piecemeal-quantile`.

3.2.2 Accuracy for Independent Extrema

We computed correlated aggregates of the form $\texttt{COUNT}\{y : x < (1+\epsilon) * \texttt{MIN}(x)\}$ and $\texttt{SUM}\{y : x < (1+\epsilon) * \texttt{MIN}(x)\}$ over real and synthetic data sets. Figure 4 plots the correlated `COUNT` at different time steps of a landmark window along with the streaming approximations determined by the competing methods, for the data sets USAGE and ZIPF. Similar plots were obtained for the other data sets but are omitted due to space constraints. The graphs in Figure 4(a) show the query answers for all of the methods. There is a clear separation of the methods. As expected, the simple heuristics give lower- and upper-bounds of the exact query; both were worse than all other methods. Also as expected, the equidepth histogram outperforms equiwidth. This was the case in all the experiments, so we drop equiwidth and report only the results from the equidepth method in the remaining plots. Note that the equidepth histogram appears to be diverging from the exact value over time, whereas the proposed methods track it closely. The graph in Figure 4(b) confirms this, plotting the error for the proposed methods. Figure 4(c) plots $RMSE_i$ for the proposed methods and for equidepth histograms using ZIPF. In both data sets, all of our methods give very small $RMSE_i$ (less than 5) and appear to stabilize at a constant approximation error. Figure 5 gives analogous graphs for the case where `SUM` is the dependent aggregate. Here we see an even greater divergence between equidepth histograms and the proposed methods.

3.2.3 Sensitivity Analysis for Extrema

To test the robustness of the methods, we performed the following experiments. First, we ran queries over the same data sets with different arrival orders. We tried several random permutations of the real data and found the results to be very similar to those in Figure 4; hence, we omit the plots for brevity. Then we artificially permuted the data so that initially only large values occur and there is a sudden large drop in the running minima. We present the plots from the USAGE data set with this partially-sorted reverse ordering in Figure 6. The error for the equidepth method appears to be increasing whereas it is decreasing for the other methods. Thus, our methods appear to be the most robust.

We varied the number of histogram buckets and ran the same experiments. Figure 7 plots the results with 5 buckets for USAGE; equidepth was chosen as a representative of the competing methods. Again, our methods gave the best accuracy. Figure 7(b) plots the RMSE for only our methods. Using 5 buckets rather than 10, we see a separation of the curves, with `piecemeal-uniform` performing the best, the wholesale approaches in the middle, and `piecemeal-quantile` performing the worst.

3.2.4 Accuracy for Independent Average

We computed correlated aggregates of the form $\texttt{COUNT}\{y : x > \texttt{AVG}(x)\}$ and $\texttt{SUM}\{y : x > \texttt{AVG}(x)\}$ over real and synthetic data sets. Figure 8 plots the exact tracking value at different time steps of a landmark window along with the streaming approximations determined by the competing methods, for USAGE and MULTIFRAC, which were chosen as representatives of real and synthetic data. The plots in Figure 8(a) tracks the correlated `COUNT` for all the methods. Unlike when `MIN` was the independent aggregate, here we see that the simple heuristic performs well. This is because the mean converges early on for these data sets, so the running average is a good estimate of the true average. Of the competitors, the equidepth histogram again performed the best (although not as well as in the `MIN` case), so we chose it as a representative in the $RMSE_i$ plots. Our methods again performed better than equidepth, especially for the MULTIFRAC data. As Figure 8(c) shows, the $RMSE_i$ for equidepth grows to 180, whereas it remains below 30 for our methods with sublinear growth. Figure 9 gives analogous graphs for the case where `SUM` is the dependent aggregate; there is an even greater divergence between equidepth histograms and the proposed methods.

3.2.5 Sensitivity Analysis for Average

We tested the robustness of the methods using the same experiments as we did for queries involving `MIN` as the independent aggregate. First, we tried several random permutations of the data and found that the accuracy for all the methods was slightly better (We omit the plot since the curves looked similar.) This is not surprising – the random arrival order of tuples helps the mean converge faster, and many of the methods are based on convergence properties of the mean. Then we artificially permuted the USAGE data so that initially only large values occur and there is a sudden large drop in values. Thus, the running mean will drop sharply at that point. Figure 10(a) presents the tracking value for all the methods; (b) presents the RMSE for equidepth and our methods. It is apparent that all methods gave worse accuracy overall. Our methods did not perform as well as the true equidepth method, as shown in Figure 10(b), due to the mean converge assumption built into them; however, they were clearly superior to equiwidth histograms. We also tried running the experiments with varying numbers of buckets, but the plots did not reveal any new observations.

4. SLIDING WINDOW ALGORITHMS

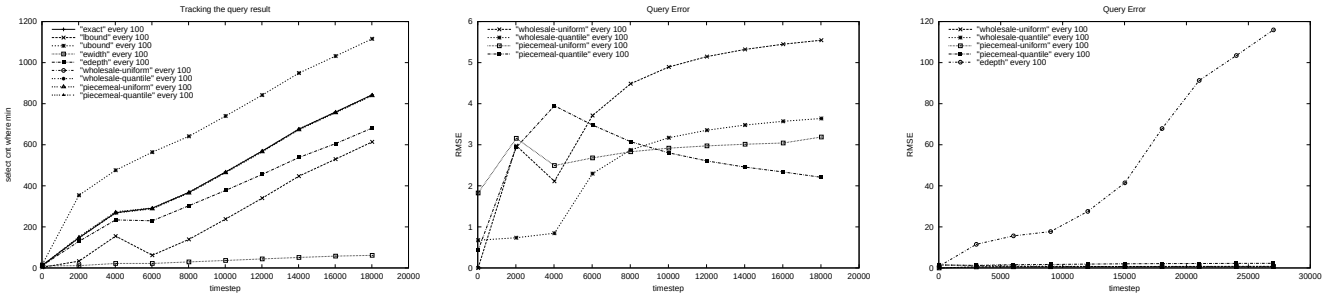
For sliding window aggregates, many of the properties discussed in Section 2 do not hold. In particular, extremas are not monotonic and confidence intervals about the mean do not converge like they do in the landmark window case. Thus, we use some reasonable heuristics in our algorithms. We describe these and present some experimental results to validate our techniques.

4.1 Algorithms

First, we give the outline of our algorithms, and then we describe the details of each method.

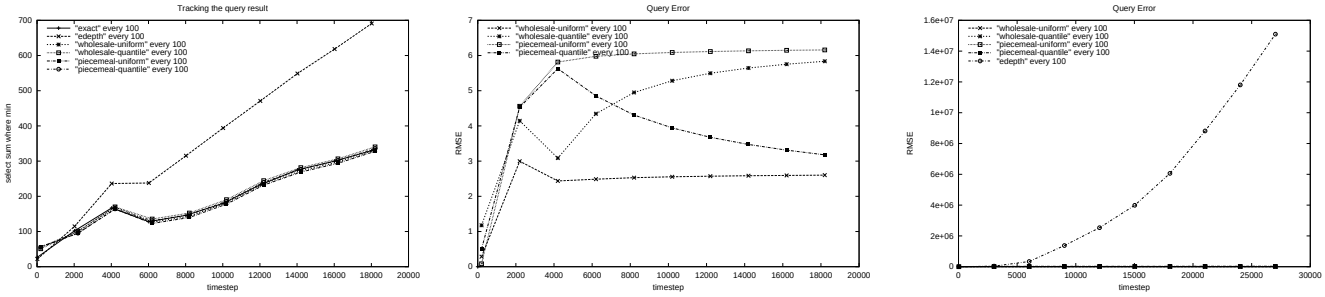
4.1.1 Sketch of Our Algorithms

Our algorithms all follow the general outline given in Figure 11. Again, the histograms contain m bins and are of the form $\langle (v_1, f_1), (v_2, f_2), ..., (v_m, f_m) \rangle$. The UpdateExtrema subroutine requires explanation. Unlike the landmark window case, where updating an extrema value is trivial, here the extrema values are non-monotonic because changes can occur due to both incoming and outgoing tuples. Thus, we must keep track of some auxiliary information to give good estimates for the extrema. We propose the following strategy for maintaining extrema values. We partition the sliding window into fixed-length intervals and keep track of the lo-



(a) tracking value for USAGE (b) $RMSE_i$ for USAGE (c) $RMSE_i$ for ZIPF

Figure 4: Correlated `COUNT` with independent `MIN` over landmark window: (a) tracking the query answer for USAGE with 10 buckets and $\epsilon = 99$; (b) $RMSE_i$; (c) $RMSE_i$ for ZIPF with 10 buckets and $\epsilon = 1000$.



(a) tracking value for USAGE (b) $RMSE_i$ for USAGE (c) $RMSE_i$ for ZIPF

Figure 5: Correlated `SUM` with independent `MIN` over landmark window: (a) tracking the query answer for USAGE with 10 buckets and $\epsilon = 99$; (b) $RMSE_i$; (c) $RMSE_i$ for ZIPF with 10 buckets and $\epsilon = 1000$.

cal extrema within each interval. When an outgoing (global) extrema value departs from the sliding window, we update the extrema using the remaining local extrema.

Sliding Window Algorithm:
Input: data stream and aggregate query
Output: result stream
$H \leftarrow$ InitializeHistogram(m);
while stream not empty do
  read tuple $S_{in}[i]$;
  UpdateExtrema($S_{in}[i]$);
  if (*condition*) then ReallocateHistogram(H);
  add incoming tuple to appropriate bucket;
  delete outgoing tuple from appropriate bucket;

Figure 11: Sliding Window Algorithm

4.1.2 Extrema as the Independent Aggregate

Here we must keep track of an interval wider than the interval $[a, b] = [\min, (1+\epsilon) * \min]$ for the landmark window algorithm, since the extrema values are non-monotonic over a sliding window. We propose to place bins at $(\min, \ldots, (1+\epsilon) * maxmin, \max)$, where $maxmin$ is the maximum of the local minimums. The remaining bucket locations are placed according to the given partitioning policy.

InitializeHistogram: This subroutine is basically the same as the one for landmark windows.

PartitionHistogram: This subroutine partitions the histogram uniformly in the interval $[\min, (1+\epsilon) * maxmin]$, leaving one extra bucket from the end of this interval to `MAX`(x).

ReallocateHistogram: This subroutine is similar to the landmark window version, except that all but the last bucket are redistributed. In the case of the piecemeal approach, we approximately maintain a uniform partitioning of the buckets in $[\min, (1+\epsilon) * maxmin]$.

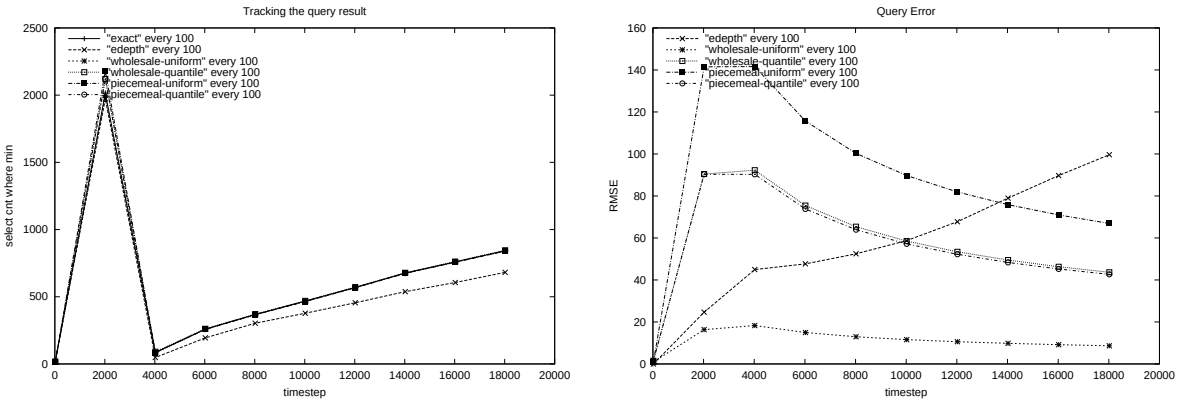
Quantiles: This subroutine is very similar to the landmark window version, except that all but the last bucket are re-quantiled.

4.1.3 Average as the Independent Aggregate

The algorithms are basically the same as the landmark window versions, except that the confidence interval does not shrink. Instead, it stays constant at $[\widehat{\mu}_n - \frac{\widehat{\sigma}_n}{\sqrt{w}}, \widehat{\mu}_n + \frac{\widehat{\sigma}_n}{\sqrt{w}}]$, where w is the size of the sliding window.

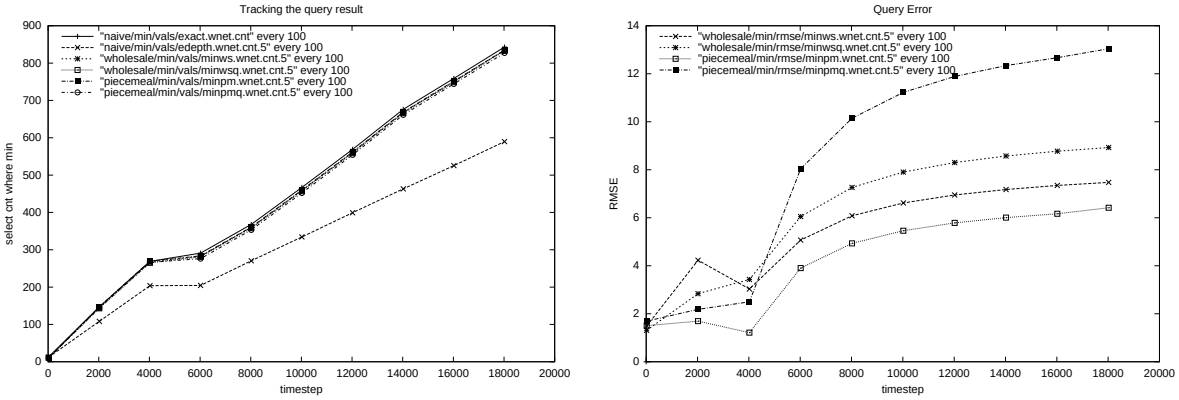
4.2 Experiments

The experimental setup is very similar to the one used in the landmark window case. We used the same data sets and queries. The competing methods are the sliding window versions of the landmark methods. We used the following



(a) query results (b) $RMSE_i$

Figure 6: Correlated `COUNT` with independent `MIN` over landmark window with data in partially-sorted reverse order: (a) tracking the query answer on USAGE with 10 buckets and $\epsilon = 99$; (b) $RMSE_i$.



(a) tracking value with 5 bins (b) $RMSE_i$ with 5 bins

Figure 7: Using fewer buckets: (a) query answer and (b) $RMSE_i$ for USAGE data with 5 buckets and $\epsilon = 99$.

quality measure for sliding window aggregates:

$$RMSE_n = \sqrt{\frac{1}{w} \sum_{i=n-w}^{n} (S_{out}[i] - S_{exact}[i])^2}$$

where w is the size of the window.

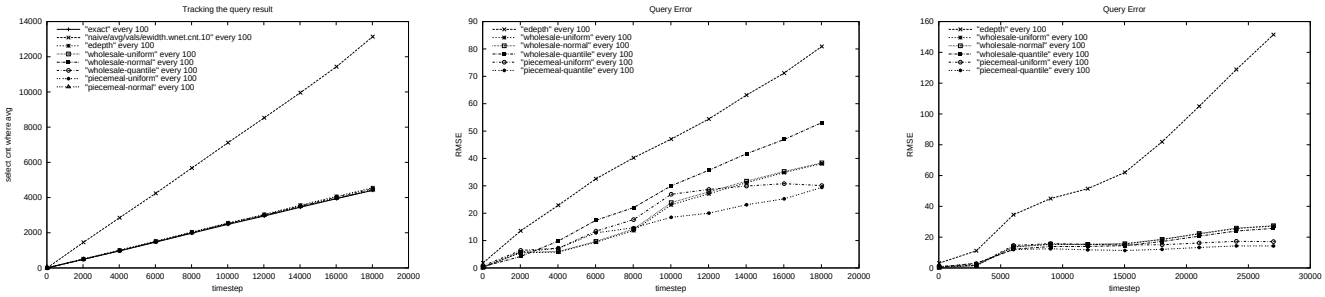
4.2.1 Accuracy for Independent Extrema

We computed correlated aggregates of the form $\texttt{COUNT}\{y : x < (1+\epsilon) * \texttt{MIN}(x)\}$ over real and synthetic data sets, over a sliding window of size 500. Figure 12 plots the exact tracking value at different time steps of a cumulative window along with the streaming approximations determined by the competing methods, for two representative data sets USAGE and MULTIFRAC. The plots in Figure 12(a) and (c) show the query result values for all the competing methods. It is somewhat clear that the best methods are equidepth and `piecemeal-uniform`. The plots in Figure 12(b) and (d) confirm this, showing the error for the proposed methods along with the equidepth competitor. In both plots, the RMSE accuracy of `piecemeal-uniform` is comparable to that of equidepth histograms. Note that, as we mentioned in Section 3, we have implemented a "true" (offline) equidepth histogram, requiring multiple passes over the data at each step; which is certainly no more feasible for sliding window streams than it is for landmark window streams.

The experiments we ran clearly separated out our methods. The versions of our methods with quantiled partitionings performed the worse. Since extrema values are not monotonic over sliding windows, the frequent and abrupt changes in bucket frequencies causes the quantiles to become stale quickly, thus rendering the policy useless. On the other hand, the methods which use uniform partitionings performed much better, with the piecemeal approach being superior to the wholesale approach in our experiments.

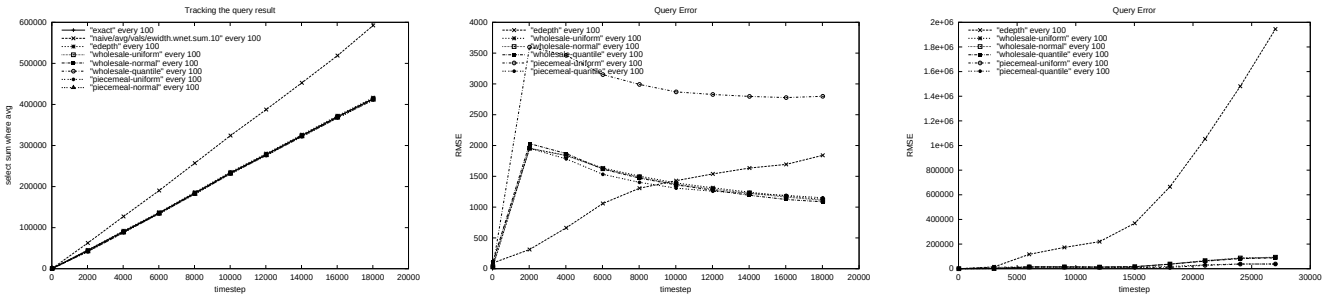
4.2.2 Accuracy for Independent Average

We computed correlated aggregates of the form $\texttt{COUNT}\{y : x > \texttt{AVG}(x)\}$ over real and synthetic data sets, over a sliding window of size 500. Figure 13 plots the exact tracking value at different time steps of a cumulative window along with the streaming approximations determined by the competing methods, for two representative data sets ZIPF and



(a) tracking value for USAGE (b) $RMSE_i$ for USAGE (c) $RMSE_i$ for MULTIFRAC

Figure 8: Correlated `COUNT` with independent `AVG` over landmark window: (a) tracking the query answer for the USAGE data with 10 buckets; (b) $RMSE_i$; (c) $RMSE_i$ for MULTIFRAC with 10 buckets.



(a) tracking value for USAGE (b) $RMSE_i$ for USAGE (c) $RMSE_i$ for MULTIFRAC

Figure 9: Correlated `SUM` with independent `AVG` over landmark window: (a) tracking the query answer for the USAGE data with 10 buckets; (b) $RMSE_i$; (c) $RMSE_i$ for MULTIFRAC with 10 buckets.

MGCTY. The plots in Figure 13(a) and (c) show the query result values for all the competing methods. The equidepth was slightly better in both experiments; the plots in Figure 13(b) and (d) show this. However, the proposed methods were competitive in both cases. For the ZIPF data, both wholesale methods were able to correct themselves after initially starting off with high $RMSE_i$.

The experiment on the real data set shows that the methods which employ uniform bucket partitioning are somewhat superior to the quantiled case, as we observed with sliding window queries with `MIN` as the independent aggregate. Here the mean is non-monotonic but, unlike with landmark windows, does not converge over time because it is computed over a fixed-length interval. Once again, the methods based on uniform partitioning appear to be more robust, making them more suitable to handle non-monotonicity.

5. RELATED WORK

Data streams have been of much recent interest [19]. In particular, algorithms and systems have been proposed for the computation of approximate frequency moments [1], L^1 and L^p distance functions [9, 12], and property testing [10].

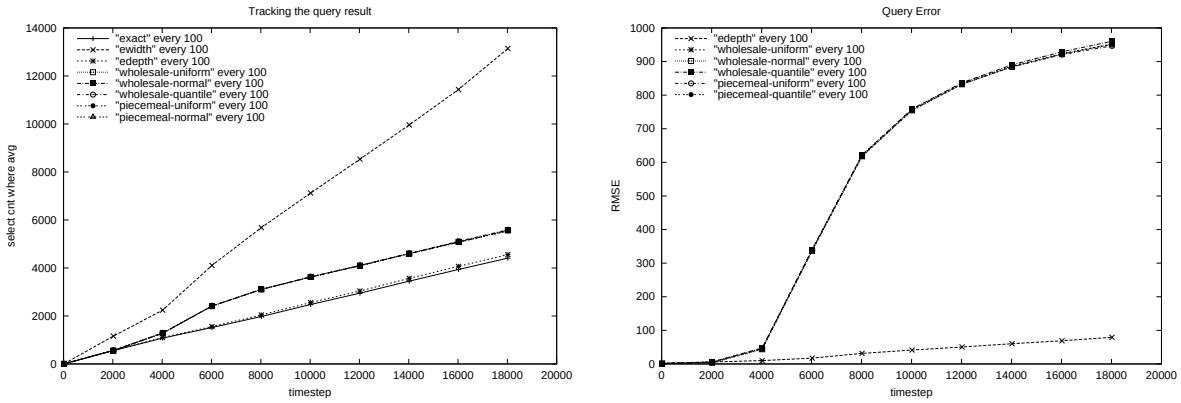
The maintenance of aggregate queries is a special case of the problem of incremental view maintenance; in particular, the maintenance of basic statistical aggregates in the presence of database updates was considered in [22]. The synopsis data structures of Matias et al. [15] consider the approximate maintenance of more fancy aggregates in the presence of updates. In online aggregation, Hellerstein et al. study the convergence of basic aggregates over finite data sets [18] and they describe access methods that retrieve records in random order in order to use statistical estimators based on independence assumptions. This work has been extended to online computation of joins [17], online reordering [23] and to adaptive query processing [3].

There has also been recent work on mining data streams, such as the construction of decision trees over data streams [13, 8] and clustering data streams [16]. Recent work by Alsabti et al. [2] and Manku et al. [20, 21] considers how to compute the approximate median and other quantiles in a single pass over a data set.

Correlated aggregates were originally considered in [6, 5, 4]; there the focus is on exact computation over finite data sets in multiple passes. To the best of our knowledge, there has not been any prior work on the approximate computation of this important class of aggregates.

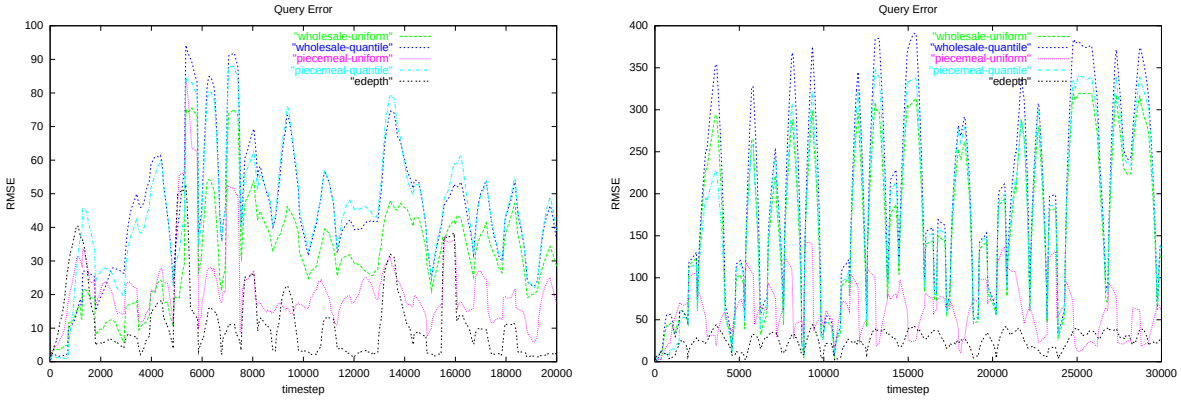
6. CONCLUSIONS AND FUTURE WORK

Effectively dealing with large volumes of streaming data, generated by applications as diverse as network management and telephone fraud detection, is a major challenge for the database community. A fundamental problem in this area is to compute and maintain a variety of complex aggregates, in an online fashion. We show that, while computing complex correlated aggregates exactly is not possible on streaming data, it is certainly feasible to do so *approximately*.



(a) query results (b) $RMSE_i$

Figure 10: Correlated `COUNT` with independent `AVG` over landmark window with data in partially-sorted reverse order: (a) tracking the query answer on USAGE with 10 buckets; (b) $RMSE_i$.



(a) $RMSE_i$ for USAGE (b) $RMSE_i$ for MULTIFRAC

Figure 12: Correlated `COUNT` with `MIN` as independent aggregate over sliding window of size $w = 500$: (a) $RMSE_i$ for USAGE with 10 buckets and $\epsilon = 99$; (b) $RMSE_i$ for MULTIFRAC with 10 buckets and $\epsilon = 99$.

Our solution to this problem is based on the use of focused histograms, which require accurate maintenance of summary information only in small data intervals; what makes the problem challenging is that these intervals are not known *a priori*: they can "move around", "expand" or "shrink", depending on the data in the stream. This renders current histogramming techniques ineffective. We presented two families of adaptive techniques, which we called wholesale and piecemeal, for efficiently "tracking" the true values of the desired correlated aggregates. Experimental results on a variety of real and synthetic data sets confirm the versatility of these techniques. Piecemeal, in particular, is a simple, elegant strategy that is extremely effective for a wide range of window scopes, and different types of aggregates, making it the strategy of choice.

7. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *JCSS: Journal of Computer and System Sciences*, 58, 1999.

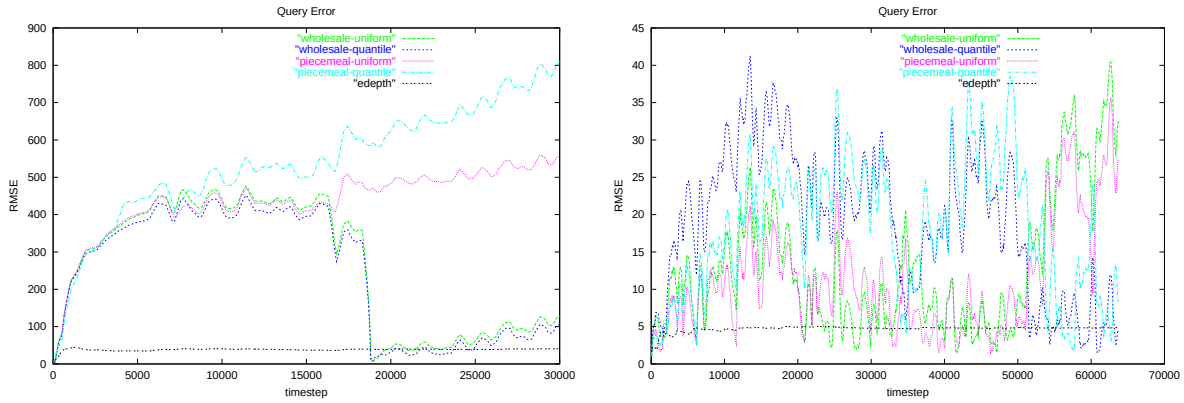
[2] K. Alsabti, S. Ranka, and V. Singh. A one-pass algorithm for accurately estimating quantiles for disk-resident data. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 346–355, 1997.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 261–272, 2000.

[4] D. Chatziantoniou. Ad hoc OLAP: Expression and evaluation. In *Proceedings of the IEEE International Conference on Data Engineering*, 1999.

[5] D. Chatziantoniou, M. Akinde, T. Johnson, and S. Kim. The MD-join: An operator for complex OLAP. In *Proceedings of the IEEE International Conference on Data Engineering*, 2001.

[6] D. Chatziantoniou and K. A. Ross. Querying multiple features of groups in relational databases. In *Proceedings of the International Conference on Very*



(a) $RMSE_i$ for ZIPF (b) $RMSE_i$ for MGCTY

Figure 13: Correlated `COUNT` with `AVG` as independent aggregate over sliding window of size $w = 500$: (a) $RMSE_i$ for ZIPF with 10 buckets; (b) $RMSE_i$ for MGCTY with 10 buckets.

Large Databases, pages 295–306, 1996.

[7] A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors. *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*. ACM Press, 1999.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, August 2000. ACM.

[9] J. Feigenbaum, R. Kannan, M. Strauss, and M. Viswanathan. An approximate L1-difference algorithm for massive data streams. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[10] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[11] A. Feldmann, A. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of internet wan traffic. In *ACM SIGCOMM*, pages 42–55, 1998.

[12] J. Fong and M. Strauss. An approximate L^p-difference algorithm for massive data streams. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science*, 2000.

[13] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh. Boat-optimistic decision tree construction. In Delis et al. [7], pages 169–180.

[14] P. Gibbons, Y. Mattias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. *Proceedings of VLDB, Athens Greece*, pages 466–475, Aug. 1997.

[15] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 909–910, N.Y., Jan. 17–19 1999. ACM-SIAM.

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

[17] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 287–298, 1999.

[18] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In J. Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 171–182. ACM Press, 1997.

[19] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-011, Digital Eqipment Corporation, Systems Research Center*, May, 1998.

[20] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In L. M. Haas and A. Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 426–435. ACM Press, 1998.

[21] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In Delis et al. [7], pages 251–262.

[22] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 144–155. Morgan Kaufmann, 2000.

[23] V. Raman, B. Raman, and J. M. Hellerstein. Online dynamic reordering for interactive data processing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 709–720, 1999.