Efficient Approximation of Correlated Sums on Data Streams

Rohit Ananthakrishna
Cornell University
rohit@cs.cornell.edu

Abhinandan Das
Cornell University
asdas@cs.cornell.edu

Johannes Gehrke
Cornell University
johannes@cs.cornell.edu

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

S. Muthukrishnan
AT&T Labs–Research
muthu@research.att.com

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

Abstract

In many applications such as IP network management, data arrives in streams, and queries over those streams need to be processed online using limited storage. Correlated-sum (CS) aggregates are a natural class of queries formed by composing basic aggregates on (x, y) pairs, and are of the form $\texttt{SUM}\{g(y) : x \leq f(\texttt{AGG}(x))\}$, where $\texttt{AGG}(x)$ can be any basic aggregate and $f()$, $g()$ are user-specified functions. CS-aggregates cannot be computed exactly in one pass through a data stream using limited storage; hence, we study the problem of computing approximate CS-aggregates.

We guarantee *a priori* error bounds when $\texttt{AGG}(x)$ can be computed in limited space (e.g., `MIN`, `MAX`, `AVG`), using two variants of Greenwald and Khanna's summary structure for the approximate computation of quantiles. Using real data sets, we experimentally demonstrate that an adaptation of the quantile summary structure uses much lesser space, and is significantly faster, than a more direct use of the quantile summary structure, for the same *a posteriori* error bounds. Finally, we prove that, when $\texttt{AGG}(x)$ is a quantile (which cannot be computed over a data stream in limited space), the error of a CS-aggregate can be arbitrarily large.

Index: Correlated aggregates, data streams, approximation, summary structures, *a priori* error bounds, IP network management.

1 Introduction

In many applications from IP network management to telephone fraud detection, data arrives in *streams*, and queries over those streams need to be processed in an online fashion to enable real-time responses. For example, IP router interfaces are periodically polled by network operators using Simple Network Management Protocol (SNMP) to get a variety of performance measurements such as incoming and outgoing traffic volumes. This stream of polled SNMP data is then queried to monitor network traffic behavior.

The large volume of stream data, and the online nature of the various applications that operate on such data, make it imperative for the applications to compute a variety of summary information in an online fashion using a bounded amount of space (see, e.g., [1, 2, 4, 5, 6, 7, 8], and the references therein). Correlated aggregates (see, e.g., [3, 6]), which provide a natural mechanism for the flexible composition of basic aggregates, are desirable since they are more descriptive than the basic aggregates for understanding relationships between variables in the stream data.

An example correlated aggregate from a network management application is $\texttt{AVG}\{pmit : pmot \leq \texttt{AVG}(pmot)\}$, which operates on a multiset of $(pmot, pmit)$ pairs, and correlates them by computing the average of *pmit* (per-minute incoming traffic) values over pairs whose *pmot* (per-minute outgoing traffic) value does not exceed the average *pmot* value in the multiset. Queries of this form have been reported in [9] for the Tribeca network monitoring system. Yet another example of a correlated aggregate is the robust *trimmed mean* statistic, which is computed after eliminating low and high quantile values from the input; it is less sensitive to outliers than `AVG`. Numerous applications of correlated aggregates abound for other domains (e.g., see `http://www.traderbot.com` for examples in a real-time financial trading system).

Correlated-sum (CS) aggregates are a natural class of correlated aggregates on (x, y) pairs, and are of the form $\texttt{SUM}\{g(y) : x \leq f(\texttt{AGG}(x))\}$, where the independent aggregate $\texttt{AGG}(x)$ can be any basic aggregate. The above examples are simple variations of CS-aggregates. In general, CS-aggregates *cannot* be computed exactly over data streams with limited storage; the $\texttt{AGG}(x)$ value needs to be known exactly before one can identify the relevant input tuples for the computation of $\texttt{SUM}(g(y))$. Recently, Gehrke et al. [6] proposed practical online techniques for the approximate computation of a variety of correlated aggregates, using adaptive histograms. While they experimentally showed the utility of their approach, no *a priori* quality guarantees are provided by their techniques.

In this paper, we formally study the problem of *space-efficient approximation of CS-aggregates of very large data sets in a single pass*, and make the following contributions:

- We obtain an ϵ-approximation of CS-aggregates using $O(\frac{1}{\epsilon}\log(\epsilon Y_{sum}))$ space where Y_{sum} is the sum of the $g(y)$ values in the input stream, when $\texttt{AGG}(x)$ can be computed exactly using limited space (e.g., `MIN`, `MAX`, `AVG`, `STDEV`).

  Our construction uses two variants of the summary structure of [8] for the approximate computation of quantiles. Using real data sets, we experimentally demonstrate the space and time superiority of an adaptation of the quantile summary structure over its more direct use, for the same *a posteriori* error bounds.

- We prove that, when the independent aggregate $\texttt{AGG}(x)$ is a quantile (which cannot be computed exactly over a data stream in limited space), the error of a CS-aggregate may be arbitrarily large. This establishes a formal *separation* result.

2 Preliminaries

We are interested in computing correlated-sum (CS) aggregates over data streams of (x, y) pairs; useful complex aggregates are expressible by combining CS-aggregates. Formally,

CS-aggregates are of the form $\texttt{SUM}\{g(y) : x \leq f(\texttt{AGG}(x))\}$ where the *independent aggregate* $\texttt{AGG}(x)$ can be any basic aggregate such as $\texttt{MIN}(x), \texttt{MAX}(x), \texttt{AVG}(x), \texttt{STDEV}(x)$, or $\texttt{QTL}_\phi(x)$, $f()$ and $g()$ are user-specified functions, and both x and $g(y)$ are positive integers. Permitting $f()$ and $g()$ allows users to conduct exploratory analysis. For example, $\texttt{SUM}\{pmot : pmit \leq 0.5 * \texttt{AVG}(pmit)\}$ and $\texttt{SUM}\{pmot : pmit \leq 1.5 * \texttt{AVG}(pmit)\}$ differ only in $f()$. As we shall see later in the paper, both of these can be answered quite precisely using the *same* summary structure. In contrast, the work of [6] would need a *separate* summary structure for each of these two queries. By setting $g() = 1$, for example, we can compute $\texttt{COUNT}\{y : x \leq f(\texttt{AGG}(x))\}$. For the remainder of this paper, we do not mention $g()$, since it is straightforward how to extend our results to deal with $g()$.

The error metric we use is the *margin* L_1-error $\frac{|S-\hat{S}|}{S_\infty}$, where S and $\hat{S}$ are the exact and approximate answers, respectively, and S_∞ is the maximum value of the aggregate (over all possible inputs). Recent work on computing approximate quantiles has employed an analogous error metric where ϕ-quantiles are determined to within a rank precision of ϵn, rather than a "relative" rank precision of $\epsilon\phi n$ (see, e.g., [8]).[1] For the remainder of this paper, we shall consider the worst-case error, which provides a deterministic guarantee of the degree of imprecision for any arbitrary query, and show that CS-aggregates can often be ϵ-approximated under this error metric.

3 Computing CS-Aggregates Over Exact Aggregates

We present two online algorithms for computing ϵ-approximations of CS-aggregates over data streams, using variants of the summary structure of [8] for the approximate computation of quantiles, when the independent aggregate $\texttt{AGG}(x)$ is any basic aggregate that can be computed exactly over a data stream in limited space, such as `MIN`, `MAX`, and `AVG`. Finally, we experimentally compare their space and time requirements using real data sets.

3.1 A Direct Use of Quantile Summaries

Greenwald and Khanna [8] recently proposed the *quantile summary* data structure, that effectively maintains the minimum and maximum possible rank $(r_{min}(v), r_{max}(v))$ for each input sample v it stores. At any point in time n, the data structure $\texttt{S}(n)$ consists of an ordered sequence of tuples $t_0, t_1, \ldots, t_{s-1}$, where each tuple $t_i = (v_i, g_i, \Delta_i)$ consists of three components: (i) v_i, a value from the first n elements of the data stream, (ii) $g_i = r_{min}(v_i) - r_{min}(v_{i-1})$, and (iii) $\Delta_i = r_{max}(v_i) - r_{min}(v_i)$. t_0 and t_{s-1} correspond to the minimum and maximum elements seen so far. Note that $r_{min}(v_i) = \sum_{j \leq i} g_j$, and $r_{max}(v_i) = r_{min}(v_i) + \Delta_i$.

By ensuring that $\texttt{S}(n)$ satisfies the property that $\max_i(g_i + \Delta_i) \leq 2\epsilon n$, $\texttt{S}(n)$ can be used to answer any *prefix range count* (PRC) query of the form $\texttt{COUNT}\{x : x \leq x'\}$, to within an ϵn precision. Intuitively, this property can be satisfied in an online fashion, for the $(n+1)$-th observation v (which lies between v_{i-1} and v_i) as follows: (i) the tuple $(v, 1, g_i + \Delta_i - 1)$ is inserted into `S`, (ii) the tuple t_j in the (modified) `S` that has the smallest value of $e_j = g_j + g_{j+1} + \Delta_{j+1}$ is identified, and (iii) if $e_j \leq 2\epsilon(n+1)$, tuple t_j is dropped from `S` and g_j is

[1] n denotes both the time step and the number of observations seen so far in the data stream.

```
Algorithm PRC(x')
let j be the largest index s.t. x' ≥ v_j
if (j == s − 1)
   return (r_min(v_{s−1}) + r_max(v_{s−1}))/2
else
   return (r_min(v_j) + r_max(v_{j+1}) − 1)/2
```

```
Algorithm PRS(x')
let j be the largest index s.t. x' ≥ x_j
if (j == s − 1)
   return (ys_min(v_{s−1}) + ys_max(v_{s−1}))/2
else
   return (ys_min(v_j) + ys_max(v_{j+1}) − y_{j+1})/2
```

Figure 1: Algorithms PRC and PRS

added to g_{j+1}. The resulting summary structure is $\mathtt{S}(n+1)$. Algorithm $\mathtt{PRC}(x')$, in Figure 1, gives the pseudocode for answering the prefix range count query.

When $\mathtt{AGG}(x)$ is any basic aggregate that can be computed exactly over a data stream in limited space (such as MIN, MAX, and AVG), one can use the quantile summary structure in a straightforward manner for computing an ϵ-approximation of the CS-aggregate $\mathtt{SUM}\{y : x \leq f(\mathtt{AGG}(x))\}$ over data streams of (x, y) pairs. Essentially, we maintain a quantile summary over the data stream of x-values obtained from the original stream of (x, y) pairs, by (dynamically) replacing each data item (x_i, y_i) by y_i copies of x_i, and inserting them sequentially. The CS-aggregate can be estimated by invoking Algorithm $\mathtt{PRC}(x')$, where $x' = f(\mathtt{AGG}(x))$ (which is known exactly), on this quantile summary structure.

If the original data stream was of length n, the transformed data stream is of length $Y_{sum}(n)$, where $Y_{sum}(n) = \sum_{i=1}^{n} y_i$. From the results of [8], it follows that $\max_i(g_i + \Delta_i) \leq 2\epsilon Y_{sum}(n)$, and the space used is $O(\frac{1}{\epsilon}\log(\epsilon Y_{sum}(n)))$. This establishes the following result:

Proposition 3.1 *For arbitrary ϵ, and an $\mathtt{AGG}(x)$ that can be computed exactly over a data stream of (x, y) pairs in limited space, the summary structure $\mathtt{S}(n)$ can be used to compute the CS-aggregate $\mathtt{SUM}\{y : x \leq f(\mathtt{AGG}(x))\}$ to within a precision of $\epsilon Y_{sum}(n)$, using at most $O(\frac{1}{\epsilon}\log(\epsilon Y_{sum}(n)))$ samples.* ∎

3.2 PRS(n): Using an Adaptation of Quantile Summaries

When individual y-values in the data stream of (x, y) pairs are large, creating y_i copies of x_i to input to the quantile summary $\mathtt{S}(n)$ has two adverse consequences. First, it results in a large number of updates (i.e., tuple insertions and merges) to the summary structure. In a high speed data stream, as arises in many IP network management applications, the amount of time spent updating the summary structure may render this approach infeasible. Second, creating y_i copies of x_i may also result in multiple samples being retained in $\mathtt{S}(n)$ for the same x-value, resulting in a larger summary structure than needed for answering CS-aggregates. In this section, we present an *adaptation* of the quantile summary data structure that addresses these concerns, and computes an ϵ-approximation of the CS-aggregate $\mathtt{SUM}\{y : x \leq f(\mathtt{AGG}(x))\}$ using bounded space.

Our summary structure, $\mathtt{PRS}(n)$, effectively maintains the minimum and maximum possible y-sum $(ys_{min}(v'), ys_{max}(v'))$ for each input sample $v' = (x', y')$ it stores. That is,

$ys_{min}(v') \leq \texttt{SUM}\{y : x \leq x'\} \leq ys_{max}(v')$. At any point in time n, the data structure $\texttt{PRS}(n)$ consists of an ordered sequence of tuples $t_0, t_1, \ldots, t_{s-1}$, where each tuple $t_i = (v_i, h_i, \Gamma_i)$ consists of three components: (i) $v_i = (x_i, y_i)$, an element from the first n elements of the data stream; (ii) $h_i = ys_{min}(v_i) - ys_{min}(v_{i-1})$, for $i \geq 1$, and $h_0 = ys_{min}(x_0)$; (iii) $\Gamma_i = ys_{max}(v_i) - ys_{min}(v_i)$. t_0 and t_{s-1} correspond to the minimum and maximum x values seen so far. Note that $ys_{min}(v_i) = \sum_{j \leq i} h_j$, and $ys_{max}(v_i) = ys_{min}(v_i) + \Gamma_i$.

By ensuring that $\texttt{PRS}(n)$ satisfies the property that $\max_i(h_i + \Gamma_i - y_i) \leq 2\epsilon Y_{sum}(n)$, $\texttt{PRS}(n)$ can be used to answer any *prefix range sum* (PRS) query of the form $\texttt{SUM}\{y : x \leq x'\}$, to within an $\epsilon Y_{sum}(n)$ precision. Intuitively, this property can be satisfied in an online fashion, for the $(n+1)$-th observation $v = (x, y)$ as follows: (i) given an index i such that $x_{i-1} < x \leq x_i$, the tuple $(v, y, h_i + \Gamma_i - y_i)$ is inserted into PRS, (ii) the tuple t_j in the (modified) PRS that has the smallest value of $e_j = h_j + h_{j+1} + \Gamma_{j+1} - y_{j+1}$ is identified, and (iii) if $e_j \leq 2\epsilon Y_{sum}(n+1)$, tuple t_j is dropped from PRS and h_j is added to h_{j+1}. The resulting summary structure is $\texttt{PRS}(n+1)$. The computation of the prefix range sum is based on Algorithm PRS, in Figure 1.

When $\texttt{AGG}(x)$ is any basic aggregate that can be computed exactly over a data stream in limited space (such as MIN, MAX, and AVG), one can use our summary structure $\texttt{PRS}(n)$ for computing an ϵ-approximation of the CS-aggregate $\texttt{SUM}\{y : x \leq f(\texttt{AGG}(x))\}$ over data streams of (x, y) pairs. Essentially, we maintain $\texttt{PRS}(n)$ over the data stream of (x, y) pairs, and the CS-aggregate is estimated by invoking Algorithm $\texttt{PRS}(x')$, where $x' = f(\texttt{AGG}(x))$. Using an argument similar in spirit to that in [8], we have the following non-trivial result.

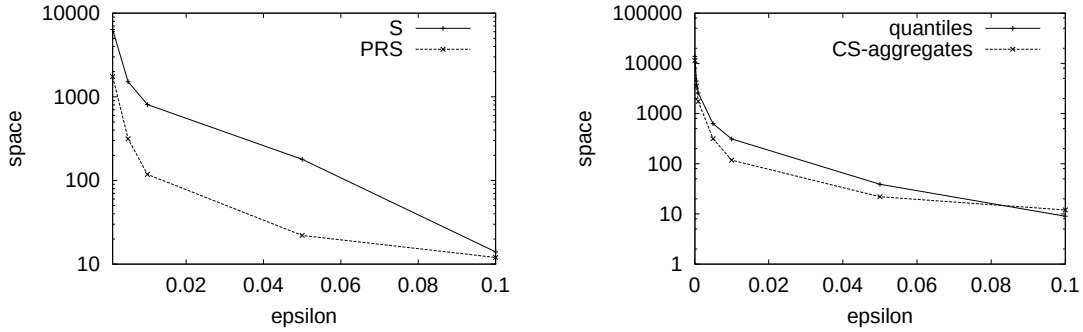
Theorem 3.1 *For arbitrary ϵ, and an $\texttt{AGG}(x)$ that can be computed exactly over a data stream of (x, y) pairs in limited space, the summary structure $\texttt{PRS}(n)$ can be used to compute the CS-aggregate $\texttt{SUM}\{y : x \leq f(\texttt{AGG}(x))\}$ to within a precision of $\epsilon Y_{sum}(n)$, using at most $O(\frac{1}{\epsilon}\log(\epsilon Y_{sum}(n)))$ samples.* ∎

3.3 Experimental Evaluation

The analytical results above (Proposition 3.1 and Theorem 3.1) indicate that the worst-case space (in terms of number of samples) used by $\texttt{S}(n)$ and $\texttt{PRS}(n)$, for computing CS-aggregates, are the same. Here, we seek to better understand their relative space-usage in terms of number of samples, and relative performance in terms of the number of merges required at runtime, on real data sets.

Figure 2(a) plots the space usage of $\texttt{PRS}(n)$ and $\texttt{S}(n)$, for computing CS-aggregates, against different values of ϵ, after 100K tuples of a real AT&T network data stream have been processed. As shown, $\texttt{PRS}(n)$ uses roughly a factor of 10 less space than $\texttt{S}(n)$ (the y-axis is drawn in logscale). We also considered the time required for both methods and found that $\texttt{PRS}(n)$ requires roughly a factor of 50 fewer merges across all ϵ and timesteps. Similar results were obtained for another real AT&T data set, but are omitted for brevity.

Figure 2(b) plots the space usage of $\texttt{S}(n)$, for computing quantiles, and $\texttt{PRS}(n)$, for computing CS-aggregates, against different values of ϵ, using the same data as above. That is, we compared the number of samples required to guarantee at most $\epsilon Y_{sum}(n)$ error for CS-aggregates on (x, y) tuples using $\texttt{PRS}(n)$ (where the worst-case space is $O(\frac{1}{\epsilon}\log(\epsilon Y_{sum}(n)))$) with that of guaranteeing at most ϵn error for quantiles on just the x-values (where the



(a) for CS-aggregates (b) quantiles vs CS-aggregates

Figure 2: Space usage of $\texttt{S}(n)$ and $\texttt{PRS}(n)$ on real AT&T data

worst-case space is $O(\frac{1}{\epsilon}\log(\epsilon n)))$. Despite the difference in their *a priori* space bounds, their space requirements in practice were comparable.

4 Inapproximability of CS-Aggregates Over Quantiles

In this section, we prove that if $\texttt{AGG}(x)$ is the quantile aggregate then the CS-aggregate $\texttt{SUM}\{y : x \leq \texttt{AGG}(x)\}$ cannot be approximated to any desired fraction ϵ by algorithms of the type we have studied in this paper.

Theorem 4.1 *There does not exist an algorithm that maintains a sample on x values over an input data stream of (x, y) pairs, along with bounds on ranks and y-sums for each sample x_i, that uses sublinear space and outputs a c-approximation ($c < 2$) to the query $\texttt{SUM}\{y : x \leq \texttt{QTL}_\phi(x)\}$ for any fraction ϕ.*

Proof: Consider the stream of length n given by $(x_1, 1), (x_2, 1), \ldots, (x_n, 1)$ with arbitrary, distinct x_i's, well separated from each other. So, $Y_{sum} = n$ so far. Say the algorithm maintains $X_0, \ldots, X_{s-1}$ in order as its samples, for some $s = o(n)$. We give this algorithm the flexibility to choose samples, choose the method for estimating the query answers, etc. By the pigeonhole principle, then there exists X_i and X_{i+1} that contain at least one of the original values x_j between them, i.e., $X_i < x_j < X_{i+1}$. This implies $r_{min}(X_{i+1}) - r_{min}(X_i) > 1$.

Now we choose α between X_i and X_{i+1}, i.e., $X_i < \alpha < X_{i+1}$, and let the next tuple in the data stream be (α, β). The algorithm would set $r_{min}(\alpha) = r_{min}(X_{i+1})$ and $\Delta(\alpha) = r_{max}(\alpha) - r_{min}(\alpha) > 1$. We also have $Y_{sum} = \beta + n$. The details of what samples, if any, are evicted is not relevant to our discussion of the lower bound.

Say our query is now $\texttt{SUM}\{y : x \leq \texttt{QTL}_\phi(x)\}$ for fraction $\phi = \frac{r_{min}(\alpha)}{n+1}$. Let us give the algorithm the prescience to get $\texttt{SUM}\{y : x \leq X_i\}$ precisely correct since it cannot make our lower bound argument any easier. Now, there are at most $\Delta(\alpha) + 1$ potential answers to the query for the algorithm of which at least two are given by the data (the others are due to inaccurate estimates of the rank of α by the algorithm). One of the two is

$\texttt{SUM}\{y : x \leq X_i\} + y_j$, and the other is $\texttt{SUM}\{y : x \leq X_i\} + \beta$, depending on the relative order of x_j and α. The algorithm cannot distinguish between these two potential answers because there is no additional information to make this distinction. By choosing α to be slightly less than x_j and to be slightly larger than x_j gives two choices for α that induces either of the two answers above. Hence, the algorithm will make at least $\frac{\beta-1}{2}$ error on average in absolute terms. This error is at least fraction $\frac{\beta-1}{2(\texttt{SUM}\{y:x\leq X_i\}+\beta)}$ which is at least $\frac{\beta-1}{2(n+\beta)}$. By choosing β to be sufficiently large (say n^2 or larger), we can make this ratio arbitrarily close to 1/2 (or 2 from over estimate). That completes the proof. ∎

5 Conclusions

Data streams have been of much recent interest, especially in applications like IP network management, financial trading, etc. CS-aggregates are an important class of queries for understanding relationships between variables in the stream data. Since they cannot be computed exactly on data streams with sublinear storage, we have studied the problem of identifying which CS-aggregates can be approximated on data streams and which ones cannot. Our work is the first to address this question.

Further study of this issue is clearly of great importance. Promising directions include studying aggregates other than SUM, extending our algorithms to sliding windows (where we are interested in queries over the last N records), and space-efficient sharing of summary structures for workloads of correlated aggregates over a data stream.

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] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of PODS*, 2002.

[3] D. Chatziantoniou and K. A. Ross. Querying multiple features of groups in relational databases. In *Proceedings of VLDB*, 1996.

[4] A. Dobra, J. Gehrke, M. Garofalakis, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proceedings of SIGMOD*, 2002.

[5] J. Feigenbaum, S. Kannan, M. Strauss, and Mahesh Viswanathan. An approximate L1-difference algorithm for massive data streams. In *Proceedings of FOCS*, 1999.

[6] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proceedings of SIGMOD*, 2001.

[7] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *Proceedings of SODA*, 1999.

[8] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of SIGMOD*, 2001.

[9] M. Sullivan and A. Heybey. Tribeca: A system for managing large databases of network traffic. In *Proceedings of USENIX*, 1998.