Randomized Synopses for Query Assurance on Data Streams

Ke Yi*, Feifei Li#, Marios Hadjieleftheriou†, George Kollios‡, Divesh Srivastava†

*Department of Computer Science and Engineering, HKUST
Clear Water Bay, Kowloon, Hong Kong
yike@cse.ust.hk

#Department of Computer Science, Florida State University
Tallahassee, FL 32306, USA
lifeifei@cs.fsu.edu

†AT&T Labs–Research
Florham Park, NJ 07932, USA
marioh@research.att.com
divesh@research.att.com

‡Department of Computer Science, Boston University
Boston, MA 02215, USA
gkollios@cs.bu.edu

***Abstract*— The overwhelming flow of information in many data stream applications forces many companies to outsource to a third-party the deployment of a Data Stream Management System (DSMS) for performing desired computations. Remote computations intrinsically raise issues of trust, making query execution assurance on data streams a problem with practical implications. Consider a client observing the same data stream as a remote server (e.g., network traffic), that registers a continuous query on the server's DSMS, and receives answers upon request. The client needs to verify the integrity of the results using significantly fewer resources than evaluating the query locally. Towards that goal, we propose a probabilistic algorithm for selection and aggregate/group-by queries, that uses constant space irrespective of the result-set size, has low update cost, and arbitrarily small probability of failure. We generalize this algorithm to allow some tolerance on the number of errors permitted (irrespective of error magnitude), and also discuss the hardness of permitting arbitrary errors of small magnitude. We also perform an empirical evaluation using live network traffic.**

I. Introduction

A large number of commercial Data Stream Management Systems (DSMS) have been developed recently to handle the continuous nature of data being generated by a variety of applications, like telephony and networking [1], [2], [3], [4], [5], [6]. Companies deploy DSMSs for gathering invaluable statistics about day-to-day operations. Due to the overwhelming data flow observed, some companies are not willing to acquire the necessary resources for deploying a DSMS. In these cases outsourcing the data stream and the desired computations to a third-party is the only alternative. For example, an Internet Service Provider (e.g., Verizon) could outsource the task of performing essential network traffic analysis to another company (e.g., AT&T) that already possesses the appropriate tools for that purpose (e.g., Gigascope). Clearly, data outsourcing and remote computations intrinsically raise issues of trust. As a consequence, outsourced query assurance on data streams is a problem with important practical implications. This problem has been studied before in the context of static outsourced data [7]. To the best of our knowledge, this is the first work to address query assurance on data streams.

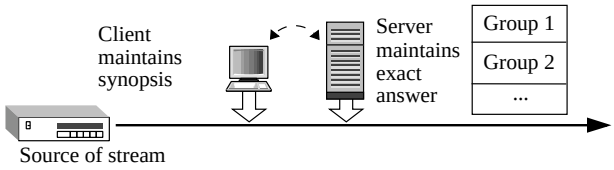


Fig. 1. System architecture.

Consider a setting where continuous queries are processed using a remote, untrusted server (that can be compromised, malicious, running faulty software, etc). A client with limited processing capabilities observing exactly the same input as the server registers queries on the server DSMS and receives results upon request (Figure 1). Assuming that charging fees are calculated based on the computation resources consumed, or the volume of traffic processed, the server has an incentive to deceive the client for increased profit. More importantly, the server might have a competing interest to provide fraudulent results. Hence, a passive malicious server can drop query results or provide random answers in order to reduce the cost of answering queries, while a compromised or active malicious server might devote additional resources to provide fraudulent results. In other cases, incorrect answers might be a result of faulty software, or due to load shedding strategies, which are essential for dealing with bursty data [8]. In critical applications real time execution assurance is essential.

Ideally, the client should be able to verify the integrity of

the computations performed in real time using significantly fewer resources than evaluating the queries locally. Here, we concentrate on verification of selection and aggregation queries. We develop solutions for verifying count, sum and other aggregates on any type of grouping imposed on the input data (e.g., as a `GROUP BY` clause in standard SQL). First, we provide a solution for verifying the absolute correctness of the results, and second, an algorithm that can tolerate a *small* number of inconsistent answers (of arbitrary magnitude). We also discuss the hardness of supporting an arbitrary number of errors with *small* absolute or relative magnitude. Clearly, if a client wants to verify the query results with absolute confidence the only solution is to compute the answers exactly, which obviates the need of outsourcing. Hence, we investigate *high-confidence probabilistic* solutions and develop verification algorithms that significantly reduce resource consumption.

The contributions of this work are: 1. A probabilistic synopsis that raises an alarm if there exists at least one error in the results. The algorithm uses $O(1)$ space (three words), O(1) update time for count queries, and $O(\log n)$ time for sum queries (n is the result-set size), for an arbitrarily small probability of failure δ; 2. A theoretical analysis of the algorithm that proves its space optimality on the bits level; 3. A strong result stating that the same synopsis can be used for verifying multiple simultaneous queries *in constant space*, on the same aggregate attribute and different selections/groupings; 4. An algorithm for tolerating a limited number of errors; 5. A proof of the hardness of tolerating an arbitrary number of small errors; 6. An empirical evaluation using live network traffic, showing that our algorithms work extremely well in practice and are very simple to implement.

II. Problem Formulation

The queries examined in this work have the following structure:

```
SELECT AGG(A_1), ..., AGG(A_N) FROM T
WHERE ... GROUP BY G_1, ..., G_M
```

This is a rather general form of SQL queries, as any "`SELECT, FROM, WHERE`" query can be written in a `GROUP BY` aggregate form by grouping every tuple by itself using a primary key. For example "`SELECT A,B FROM T WHERE B>10`" can be written as "`SELECT SUM(A),SUM(B) FROM T WHERE B>10 GROUP BY PK`". Aggregate `GROUP BY` queries have wide applications in monitoring and statistical analysis of data streams (e.g., in networking and telephony applications). Previous work has addressed exactly these types of queries numerous times (cf. [9] and related work therein). For example, a query that appears frequently in network monitoring applications is the following:

```
SELECT SUM(packet_size) FROM IP_Trace
GROUP BY source_ip, destination_ip
```

In the rest of the paper we will use this query as our main motivating example and concentrate on sum and count aggregates. Other aggregates that can be reduced to count and sum (average, standard deviation, etc.) can be easily supported.

Data Stream Model. Following the example query of the previous section, the `GROUP BY` predicate partitions the streaming tuples into a set of n groups, computing one sum per group. The data stream can be viewed as a sequence of additions (or subtractions) over a set of items in $[n] = \{1, \ldots, n\}$. Denote this data stream as $\mathcal{S}$ and its τ-th tuple as $s^\tau = (i, u^\tau)$, an update of amount u to the ith group. Formally, the query answer can be expressed as a dynamic vector of non-negative integers $\mathbf{v}^\tau = [v_1^\tau, \ldots, v_n^\tau] \in \mathbb{N}^n$. Initially, $\mathbf{v}^0$ is the zero vector. A new tuple $s^\tau = (i, u^\tau)$ updates the corresponding group i in $\mathbf{v}^\tau$ as $v_i^\tau = v_i^{\tau-1} + u^\tau$. We allow u^τ to be either positive or negative, but require $v_i^\tau \geq 0$ for all τ and i. For count queries, we have $u^\tau = 1$ for all τ. We assume that the L_1 norm of $\mathbf{v}^\tau$ is always bounded by some large m, i.e., at any τ, $\|\mathbf{v}^\tau\|_1 = \sum_{i=1}^n v_i^\tau \leq m$. Our streaming model is the same as the general Turnstile model of [10], and our algorithms are designed to work under this model. Our solution naturally supports the tumbling window semantics. Furthermore, the proposed scheme can be easily extended for sliding window semantics (cf. Section VII). Readers are referred to two excellent papers [10], [11] for detailed discussions of data stream models.

Problem Definition. The problem of *Continuous Query Verification*[1] *on data streams (CQV)* is defined as follows:

Definition 1: Given a data stream $\mathcal{S}$, a continuous query $\mathcal{Q}$ and a user defined parameter $\delta \in (0, \frac{1}{2})$, build a synopsis $\mathcal{X}$ of $\mathbf{v}$ such that for any τ, given any $\mathbf{w}^\tau$ and using $\mathcal{X}(\mathbf{v}^\tau)$, we: 1. raise an alarm with probability at least $1 - \delta$ if $\mathbf{w}^\tau \neq \mathbf{v}^\tau$; 2. shall not raise an alarm if $\mathbf{w}^\tau = \mathbf{v}^\tau$.

Here $\mathbf{w}^\tau$, for example, could be the answer provided by the server, while $\mathcal{X}(\mathbf{v}^\tau)$ is the synopsis maintained by the client for verifying vector $\mathbf{v}$.

With this definition the synopsis raises an alarm with high probability if any component (or group aggregate) v_i^τ is inconsistent. Consider a server that is using semantic load shedding, i.e., dropping tuples from certain groups, on bursty stream updates. In this scenario the aggregate of a certain, small number of components will be inconsistent without malicious intent. We would like to design a technique that allows a certain degree of tolerance in the number of erroneous answers contained in the query results, rather than raising alarms indiscriminately. The following definition captures the semantics of *Continuous Query Verification with Tolerance for a Limited Number of Errors (CQV$^\gamma$)*:

Definition 2: For any $\mathbf{w}, \mathbf{v} \in \mathbb{N}^n$, let $E(\mathbf{w}, \mathbf{v}) = \{i \mid w_i \neq v_i\}$. Then $\mathbf{w} \neq_\gamma \mathbf{v}$ iff $|E(\mathbf{w}, \mathbf{v})| \geq \gamma$ and $\mathbf{w} =_\gamma \mathbf{v}$ iff $|E(\mathbf{w}, \mathbf{v})| < \gamma$. Given user defined parameters $\gamma \in \{1, \ldots, n\}$ and $\delta \in (0, \frac{1}{2})$, build a synopsis $\mathcal{X}$ of $\mathbf{v}$ such that, for any τ, given any $\mathbf{w}^\tau$ and using $\mathcal{X}(\mathbf{v}^\tau)$, we: 1. raise an alarm with

[1] We use "verification" and "assurance" interchangeably in this paper.

probability at least $1-\delta$, if $\mathbf{w}^\tau \neq_\gamma \mathbf{v}^\tau$; 2. shall not raise an alarm if $\mathbf{w}^\tau =_\gamma \mathbf{v}^\tau$.

Note that CQV is the special case of CQV$^\gamma$ with $\gamma = 1$. Similarly, we would like to design techniques that can support random load shedding, i.e., which can tolerate small absolute or relative errors on any component irrespective of the total number of inconsistent components. The following definition captures the semantics of *Continuous Query Verification with Tolerance for Small Errors (CQV$^\eta$)*:

Definition 3: For any $\mathbf{w}, \mathbf{v} \in \mathbb{N}^n$, let $\mathbf{w} \not\approx_\eta \mathbf{v}$ iff there is some i such that $|w_i - v_i| > \eta$, and $\mathbf{w} \approx_\eta \mathbf{v}$ iff $|w_i - v_i| \leq \eta$ for all $i \in [n]$. Given user defined parameters η and $\delta \in (0, \frac{1}{2})$, build a synopsis $\mathcal{X}$ of $\mathbf{v}$ such that, for any τ, given any $\mathbf{w}^\tau$ and using $\mathcal{X}(\mathbf{v}^\tau)$, we: 1. raise an alarm with probability at least $1-\delta$, if $\mathbf{w}^\tau \not\approx_\eta \mathbf{v}^\tau$; 2. shall not raise an alarm if $\mathbf{w}^\tau \approx_\eta \mathbf{v}^\tau$.

Note that the definition above requires the *absolute* errors for each v_i^τ to be no larger than η. It is also possible to use *relative* errors, i.e., raise an alarm iff there is some i such that $|w_i^\tau - v_i^\tau|/|v_i^\tau| > \eta$. Thus CQV is also a special case of CQV$^\eta$ with $\eta = 0$.

We will work under the standard RAM model, where it is assumed that an addition, subtraction, multiplication, division, or taking mod involving two words takes one unit of time. We also assume that n/δ and m/δ fit in one word. In the rest of the paper, we drop the superscript τ when there is no confusion.

III. Possible Solutions

A naive method for solving CQV is for the client to execute exactly the same procedure as the server, in essence maintaining all the v_i's. This simple method consumes $\Theta(n)$ space, and makes outsourcing meaningless. Therefore, as with all data stream problems [10], [12], we are only interested in solutions that use space significantly smaller than $O(n)$. Next, we briefly mention some intuitive solutions and discuss why they do not solve the CQV problem. We focus on count queries only; the discussion extends to sum queries since count is a special case of sum.

Random sampling. A first attempt is random sampling. Assuming a sampling rate r, the client randomly chooses rn groups. Clearly, with probability r this method will raise an alarm if $\mathbf{w} \neq \mathbf{v}$. In order to satisfy the problem statement requirements we need to set $r = 1 - \delta$. For CQV$^\gamma$, if the server modifies exactly γ answers, then the probability of raising an alarm is only roughly r^γ, which is obviously too small for practical r's and γ's. Thus, random sampling can at most reduce the space cost by a tiny fraction.

Sketches. Recent years have witnessed a large number of sketching techniques (e.g. [12], [13], [14]) that are designed to summarize high-volume streaming data with small space. It is tempting to maintain such a sketch for the purpose of verification. However, although it is imaginable that such an approach would be likely to catch most unintentional errors such as bad communication links, the fact that they are not designed for verification leaves them vulnerable under certain attacks. More precisely, we show in [15] that there are certain $\mathbf{w} \neq \mathbf{v}$ that correspond to identical sketches with high probability. This means that using the sketch either poses a security risk, or has to incur high space and update costs as many independent copies have to be maintained. In the next section, we present our solution, which not only solves the CQV problem, but also uses much less space than all known sketches.

IV. PIRS: Polynomial Identity Random Synopsis

This section presents the *Polynomial Identity Random Synopses* (PIRS), for solving CQV (Definition 1). The synopses, as the name suggests, are based on testing the identity of polynomials at a randomly chosen point. The technique of verifying polynomial identities can be traced back to the seventies [16]. It has found applications in verifying matrix multiplications and pattern matching, among others [17].

PIRS-1. Let p be a prime s.t. $\max\{m/\delta, n\} < p \leq 2\max\{m/\delta, n\}$. According to Bertrand's Postulate such a p always exists [18]. For PIRS-1, we choose α from $\mathbb{Z}_p$ uniformly at random and compute

$$\mathcal{X}(\mathbf{v}) = (\alpha-1)^{v_1} \cdot (\alpha-2)^{v_2} \cdot \cdots \cdot (\alpha-n)^{v_n}, \qquad (1)$$

where all subtractions and multiplications are performed in the field $\mathbb{Z}_p$.

Given any input $\mathbf{w}$, PIRS-1 can verify that $\mathbf{w} = \mathbf{v}$ with high probability and without explicitly storing $\mathbf{v}$. First it checks whether $\sum_{i=1}^n w_i > m$, and if so rejects $\mathbf{w}$ immediately; otherwise it computes $\mathcal{X}(\mathbf{w})$ as:

$$\mathcal{X}(\mathbf{w}) = (\alpha-1)^{w_1} \cdot (\alpha-2)^{w_2} \cdot \cdots \cdot (\alpha-n)^{w_n},$$

again in the field $\mathbb{Z}_p$. If $\mathcal{X}(\mathbf{w}) = \mathcal{X}(\mathbf{v})$, then it declares $\mathbf{w} = \mathbf{v}$; otherwise it raises an alarm.

Example 1: Consider a simple example where we would like to verify a count/group-by query with $n = 4$ groups. Suppose we choose $p = 101$, and also $\alpha = 37$, which is unknown to the server. Assume that the stream of tuples are $(2, 3, 3, 1, 1, 4)$. Initially, we set PIRS-1 as $\mathcal{X}(v) = 1$. For each incoming tuple i, we update it as $\mathcal{X}(v) := (\mathcal{X}(v) \cdot (\alpha - i)) \bmod p$. It is easy to see that this way we are always maintaining $\mathcal{X}(v)$ as defined in (1). In the end we have $\mathcal{X}(v) = 74$. Suppose that the server returns $\mathbf{w} = (2, 1, 2, 2)$, then after computing $\mathcal{X}(\mathbf{w}) = 18$, we will declare that a wrong answer has been returned.

It is easy to see that PIRS-1 never raises a false alarm. Therefore we only need to show that it misses a true alarm with probability at most δ.

Theorem 1: Given any $\mathbf{w} \neq \mathbf{v}$, PIRS-1 raises an alarm with probability at least $1 - \delta$.

Proof: Consider polynomials $f_{\mathbf{v}}(x) = (x-1)^{v_1}(x-2)^{v_2} \cdots (x-n)^{v_n}$ and $f_{\mathbf{w}}(x) = (x-1)^{w_1}(x-2)^{w_2} \cdots (x-n)^{w_n}$. Since the leading coefficient of both polynomials is 1, if

$\mathbf{v} = \mathbf{w}$, then $f_{\mathbf{v}}(x) \equiv f_{\mathbf{w}}(x)$. If $\mathbf{v} \neq \mathbf{w}$, since both $f_{\mathbf{v}}(x)$ and $f_{\mathbf{w}}(x)$ have degree at most m, $f_{\mathbf{v}}(x) = f_{\mathbf{w}}(x)$ happens at no more than m values of x, due to the fundamental theorem of algebra. Hence, for PIRS-1, since we have $p \geq m/\delta$ choices for α, the probability that $\mathcal{X}(\mathbf{v}) = \mathcal{X}(\mathbf{w})$ happens is at most δ. ∎

Note that once we have chosen α, it is easy to maintain $\mathcal{X}(\mathbf{v})$ incrementally. For count queries, each new tuple increments a v_i by one, so the update cost is constant (one addition and one multiplication). For sum queries, a tuple $s = (i, u)$ increases v_i by u, so we need to compute $(\alpha - i)^u$, which can be done in $O(\log u)$ time (exponentiation by squaring). To perform a verification with $\mathbf{w}$, we need to compute $(x - i)^{w_i}$ for each nonzero entry w_i of $\mathbf{w}$, which takes $O(\log w_i)$ time, so the time needed for a verification is $O(\sum \log w_i) = O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$. Since both $\mathcal{X}(\mathbf{v})$ and α are smaller than p, the space complexity of the synopsis is $O(\log \frac{m}{\delta} + \log n)$ bits.

Theorem 2: PIRS-1 occupies $O(\log \frac{m}{\delta} + \log n)$ bits of space, spends $O(1)$ (resp. $O(\log u)$) time to process a tuple for count (resp. sum) queries, and $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$ time to perform a verification.

Some special care is needed for handling deletions (when u is negative), as the field $\mathbb{Z}_p$ is not equipped with division. First, we need to compute $(\alpha - i)^{-1}$, the multiplicative inverse of $(\alpha - i)$ in modulo p, in $O(\log p)$ time (using Euclid's gcd algorithm). Then we compute $(\alpha - i)^{-1|u|}$.

A nice property of PIRS is that the verification can be performed in one pass of $\mathbf{w}$ using a constant number of words of memory. This is especially useful when $|\mathbf{w}|$ is large. The client receives $\mathbf{w}$ in a streaming fashion, verifies it online, and either forwards it to a dedicated server for further processing, or a network storage device for offline use. This means that PIRS always uses very small memory throughout the entire execution and verification process.

PIRS-2. When $n \ll m$ we can improve the space bound of PIRS-1. Let prime p s.t. $\max\{m, n/\delta\} \leq p \leq 2\max\{m, n/\delta\}$. For α chosen uniformly at random from $\mathbb{Z}_p$, we compute

$$\mathcal{X}(\mathbf{v}) = v_1\alpha + v_2\alpha^2 + \cdots + v_n\alpha^n.$$

By considering the polynomial $f_{\mathbf{v}}(x) = v_1 x + v_2 x^2 + \cdots + v_n x^n$, we can use a similar proof to show that Theorem 1 holds for PIRS-2. Nevertheless, PIRS-2 has an $O(\log n)$ update cost both for count and sum queries, since given tuple $s = (i, u)$ we need to compute $u\alpha^i$.

Theorem 3: PIRS-2 occupies $O(\log m + \log \frac{n}{\delta})$ bits of space, spends $O(\log n)$ time to process a tuple, and $O(|\mathbf{w}| \log n)$ time to perform a verification.

Space optimality. Below we give a lower bound showing that PIRS is space-optimal on the bits level for almost all values of m and n.

Theorem 4: Any synopsis solving the CQV problem with failure probability at most δ has to keep $\Omega(\log \frac{\min\{m,n\}}{\delta})$ bits.

Proof: We will take an information-theoretic approach. Assume that $\mathbf{v}$ and $\mathbf{w}$ are both taken from a universe $\mathcal{U}$, and let $\mathcal{M}$ be the set of all possible memory states the synopsis might keep. Any synopsis $\mathcal{X}$ can be seen as a function $f : \mathcal{U} \to \mathcal{M}$; and if $\mathcal{X}$ is randomized, it can be seen as a function randomly chosen from a family of such functions $\mathcal{F} = \{f_1, f_2, \dots\}$, where f_i is chosen with probability $p(f_i)$. Without loss of generality, we assume that $p(f_1) \geq p(f_2) \geq \cdots$. Note that $\mathcal{X}$ needs at least $\log |\mathcal{M}|$ bits to record the output of the function and $\log |\mathcal{F}|$ bits to describe the function chosen randomly from $\mathcal{F}$.

For any $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$, let $\mathcal{F}_{\mathbf{w},\mathbf{v}} = \{f \in \mathcal{F} \mid f(\mathbf{w}) = f(\mathbf{v})\}$. For a randomized synopsis $\mathcal{X}$ to solve CQV with error probability at most δ, the following must hold for all $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$:

$$\sum_{f \in \mathcal{F}_{\mathbf{w},\mathbf{v}}} p(f) \leq \delta. \tag{2}$$

Let us focus on the first $k = \lceil \delta \cdot |\mathcal{F}| \rceil + 1$ functions $f_1, \dots, f_k$. It is easy to see that $\sum_{i=1}^{k} p(f_i) > \delta$. Since there are a total of $|\mathcal{M}|^k$ possible combinations for the outputs of these k functions, by the pigeon-hole principle, we must have

$$|\mathcal{U}| \leq |\mathcal{M}|^k \tag{3}$$

so that no two $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$ have $f_i(\mathbf{w}) = f_i(\mathbf{v})$ for all $i = 1, \dots, k$; otherwise we would find $\mathbf{w}, \mathbf{v}$ that violate (2).

Taking log on both sides of (3), we have

$$\log |\mathcal{U}| \leq (\lceil \delta \cdot |\mathcal{F}| \rceil + 1) \log |\mathcal{M}|.$$

Since $\mathbf{v}$ has n entries whose sum is at most m, by simple combinatorics, we have $|\mathcal{U}| \geq \binom{m+n}{n}$, or $\log |\mathcal{U}| \geq \min\{m, n\}$. We thus obtain the following tradeoff:

$$|\mathcal{F}| \cdot \log |\mathcal{M}| = \Omega(\min\{m, n\}/\delta).$$

If $\log |\mathcal{F}| \leq (1 - \epsilon) \log(\min\{m, n\}/\delta)$ (i.e., $|F| \leq (\min\{m, n\}/\delta)^{1-\epsilon}$) for any constant $\epsilon \in (0, 1)$, then $\mathcal{X}$ has to use super-polylogarithmic space $\log |\mathcal{M}| = \Omega((\min\{m, n\}/\delta)^{\epsilon})$; else $\mathcal{X}$ has to keep $\log |\mathcal{F}| \geq \log (\min\{m, n\}/\delta)$ random bits. ∎

Therefore, when $m \leq n$, PIRS-1 is optimal as long as $\log n = O(\log \frac{m}{\delta})$; when $m > n$, PIRS-2 is optimal as long as $\log m = O(\log \frac{n}{\delta})$. The bounds are not tight when $\log \frac{m}{\delta} = o(\log n)$ or $\log \frac{n}{\delta} = o(\log m)$.

Practical considerations. The theoretical analysis above focuses on the bit-level space complexity. In practice, however, both PIRS-1 and PIRS-2 use three words (p, α, and $\mathcal{X}(\mathbf{v})$), and thus do not seem to differ. Nevertheless, there are some technical issues to be considered. First, we shall choose p to be the maximum prime that fits in a word, so as to minimize δ. Note that $\delta = m/p$ for PIRS-1 and $\delta = n/p$ for PIRS-2. For instance if we use 64-bit words and $m < 2^{32}$, then δ is at most 2^{-32} for PIRS-1, which practically means a minuscule probability of failure. Second, since we need to extract the

group id i from each incoming tuple directly without the use of a dictionary (which would blow up the memory cost), the size of the group space n needs to be large for certain queries. For example, the SQL query in Section II has a group space of $n = 2^{64}$ (the concatenation of two IP addresses), even though the actual number of nonzero entries $|\mathbf{v}|$ may be nowhere near n. In such cases, since m will typically be much smaller than n, PIRS-1 would be the better choice.

V. Tolerance for Few Errors

This section presents a synopsis for solving the CQV$^\gamma$ problem (Definition 2). Let γ be the number of components in $\mathbf{v}$ that are allowed to be inconsistent. First, we present a construction that gives an exact solution that satisfies the requirements of CQV$^\gamma$, and requires $O(\gamma^2 \log \frac{1}{\delta} \log n)$ bits of space, which is practicable only for small γ's. Then, we provide an approximate solution which uses only $O(\gamma \log \frac{1}{\delta}(\log m + \log n))$ bits. Both solutions use PIRS as a black box, and therefore can choose either PIRS-1 or PIRS-2. We state all the results using PIRS-1 for count queries. The corresponding results for sum queries and PIRS-2 can be obtained similarly.

A. PIRS$^\gamma$: An Exact Solution

By using PIRS as a building block we can construct a synopsis that satisfies the requirements of CQV$^\gamma$. This synopsis, referred to as PIRS$^\gamma$, consists of multiple *layers*, where each layer contains $k = c_1\gamma^2$ *buckets* ($c_1 \geq 1$ is a constant to be determined shortly). Each component of $\mathbf{v}$ is assigned to one bucket per layer, and each bucket is represented using only its PIRS synopsis (see Figure 2). PIRS$^\gamma$ raises an alarm if at least γ buckets in any layer raise an alarm. The intuition is that if there are less than γ errors, no layer will raise an alarm, and if there are more than γ errors, at least one of the layers will raise an alarm with high probability (when the γ inconsistent components do not collide on any bucket for this layer). By choosing the probability of failure of the individual PIRS synopsis carefully, we can guarantee that PIRS$^\gamma$ achieves the requirements of Definition 2.

$$\begin{bmatrix} \underset{\{v_3\}}{\mathcal{X}_{11}} & \underset{\{v_2,v_5\}}{\mathcal{X}_{12}} & \underset{\{v_1,v_4,v_6\}}{\mathcal{X}_{13}} \\ \underset{\{v_1,v_3\}}{\mathcal{X}_{21}} & \underset{\{v_2,v_5\}}{\mathcal{X}_{22}} & \underset{\{v_4,v_6\}}{\mathcal{X}_{23}} \end{bmatrix}$$

Fig. 2. The PIRS$^\gamma$ synopsis.

Concentrating on one layer only, let $b_1, \ldots, b_n$ be n γ-wise independent random numbers, uniformly distributed over $\{1, \ldots, k\}$. PIRS$^\gamma$ assigns v_i to the b_i-th bucket, and for each bucket computes the PIRS synopsis of the assigned subset of v_i's with probability of failure $\delta' = 1/(c_2\gamma)$ ($c_2 \geq 1$ is a constant to be determined shortly). According to Theorem 2 each of these k synopses occupies $O(\log \frac{m}{\delta'} + \log n) = O(\log m + \log n)$ bits. Given some $\mathbf{w} =_\gamma \mathbf{v}$, since there are less than γ errors, the algorithm will not raise an alarm. We can choose constants c_1 and c_2 such that if $\mathbf{w} \neq_\gamma \mathbf{v}$, then the algorithm will raise an alarm with probability at least $1/2$ for this layer. In this case there are two cases when the algorithm will fail to raise an alarm: 1. There are less than γ buckets that contain erroneous components of $\mathbf{w}$; 2. There are at least γ buckets containing erroneous components but at least one of them fails due to the failure probability of PIRS. We show that by setting constants $c_1, c_2 = 4.819$ either case occurs with probability at most $1/4$ (details in [15]).

Algorithm 1: PIRS$^\gamma$-INIT(Prime p, Threshold γ)

```
c = 4.819, k = ⌈cγ²⌉
Generate β_{ℓ,j} uniformly at random from Z_p, for
  1 ≤ ℓ ≤ log 1/δ, 1 ≤ j ≤ k
for ℓ = 1, ..., ⌈log 1/δ⌉ do
    Layer L_ℓ = [X_1(v) := 0, ··· , X_k(v) := 0]
    // X_j(v) is a PIRS synopsis with
         δ' = 1/cγ
```

Algorithm 2: PIRS$^\gamma$-UPDATE(Tuple $s = (i, u)$)

```
for ℓ = 1, ..., ⌈log 1/δ⌉ do
    b_{ℓ,i} = (β_{ℓ,γ} i^{γ-1} + ... + β_{ℓ,2} i + β_{ℓ,1}) mod k + 1
    Update L_ℓ.X_{b_{ℓ,i}}(v) using s
```

Therefore, using one layer PIRS$^\gamma$ will raise an alarm with probability at least $1/2$ on some $\mathbf{w} \neq_\gamma \mathbf{v}$, and will not raise an alarm if $\mathbf{w} =_\gamma \mathbf{v}$. By using $\log \frac{1}{\delta}$ layers and reporting an alarm if at least one of these layers raises an alarm, the probability is boosted to $1 - \delta$.

Theorem 5: For any $\mathbf{w} \neq_\gamma \mathbf{v}$, PIRS$^\gamma$ raises an alarm with probability at least $1 - \delta$. For any $\mathbf{w} =_\gamma \mathbf{v}$, PIRS$^\gamma$ will not raise an alarm.

In addition to the $k \log \frac{1}{\delta}$ PIRS synopses, we also need to generate the γ-wise independent random numbers. Using standard techniques we can generate them on-the-fly using $O(\gamma \log n)$ truly random bits. Specifically, the technique of [19] for constructing *k-universal hash families* can be used. Let p be some prime between n and $2n$, and $\alpha_0, \ldots, \alpha_{\gamma-1}$ be γ random numbers chosen uniformly and independently from $\mathbb{Z}_p$. Then we set

$$b_i = ((\alpha_{\gamma-1} i^{\gamma-1} + \alpha_{\gamma-2} i^{\gamma-2} + \cdots + \alpha_0) \mod p) \mod k+1,$$

for $i = 1, \ldots, n$. For an incoming tuple $s = (i, u)$, we compute b_i using the α_j's in $O(\gamma)$ time, and then perform the update to the corresponding PIRS. To perform a verification, we can compute in parallel for all the layers while making one pass over $\mathbf{w}$. The detailed initialization, update and verification algorithms for PIRS$^\gamma$ appear in Algorithms 1, 2, and 3. The next theorem bounds both the space and time complexity of PIRS$^\gamma$.

Theorem 6: PIRS$^\gamma$ requires $O(\gamma^2 \log \frac{1}{\delta}(\log m + \log n))$ bits, spends $O(\gamma \log \frac{1}{\delta})$ time to process a tuple, and $O(|\mathbf{w}|(\gamma + \log \frac{m}{|\mathbf{w}|}) \log \frac{1}{\delta})$ time to perform a verification.

Algorithm 3: PIRS$^{\gamma}$-VERIFY(Vector $\mathbf{w}$)

1 **for** $\ell = 1, \ldots, \lceil \log 1/\delta \rceil$ **do**
2 Layer $M_\ell = [\mathcal{X}_1(\mathbf{w}) := 0, \cdots, \mathcal{X}_k(\mathbf{w}) := 0]$
   // $\mathcal{X}_j(\mathbf{w})$ is a PIRS synopsis with $\delta' = 1/c\gamma$
3 **for** $i = 1, \ldots, n$ **do**
4 Generate $b_{\ell,i}$ as line 2, Algorithm 2
5 Update $M_\ell.\mathcal{X}_{b_{\ell,i}}(\mathbf{w})$ by $s = (i, w_i)$
6 **if** $|\{j \mid L_i.\mathcal{X}_j(\mathbf{v}) \neq M_i.\mathcal{X}_j(\mathbf{w}), 1 \leq j \leq k\}| \geq \gamma$ **then** Raise an alarm

B. PIRS$^{\pm\gamma}$: An Approximate Solution

The exact solution works when only a small number of errors can be tolerated. In applications where γ is large, the quadratic space requirement is prohibitive. If we relax the definition of CQV$^{\gamma}$ to allow raising alarms when *approximately* γ errors have been observed, we can design more space-efficient algorithms. Notice that for large γ, small deviations are often acceptable in practice. This section presents such an approximate solution, denoted with PIRS$^{\pm\gamma}$, that guarantees the following:

Theorem 7: PIRS$^{\pm\gamma}$: 1. raises no alarm with probability at least $1-\delta$ on any $\mathbf{w} =_{\gamma^-} \mathbf{v}$ where $\gamma^- = (1 - \frac{c}{\ln\gamma})\gamma$; and 2. raises an alarm with probability at least $1-\delta$ on any $\mathbf{w} \neq_{\gamma^+} \mathbf{v}$ where $\gamma^+ = (1 + \frac{c}{\ln\gamma})\gamma$, for any constant $c > -\ln\ln 2 \approx 0.367$.

Note that this is a very sharp approximation; the multiplicative approximation ratio $1 \pm \frac{c}{\ln\gamma}$ is close to 1 for large γ.

PIRS$^{\pm\gamma}$ also contains multiple *layers* of *buckets*, where each bucket is assigned a subset of the components of $\mathbf{v}$ and summarized using PIRS (Figure 2). Focusing on one layer only, our desiderata is on any $\mathbf{w} =_{\gamma^-} \mathbf{v}$ not to raise an alarm with probability at least $1/2+\epsilon$ for some constant $\epsilon \in (0, 1/2)$, and on any $\mathbf{w} \neq_{\gamma^+} \mathbf{v}$ to raise an alarm with probability at least $1/2 + \epsilon$. By using $O(\log \frac{1}{\delta})$ independent layers and reporting the *majority* of the results, the probabilistic guarantee will be boosted to $1 - \delta$ using Chernoff bounds [17].

Let k be the number of buckets per layer. The components of $\mathbf{v}$ are distributed into the k buckets in a γ^+-wise independent fashion, and for each bucket the PIRS sum of those components is computed using $\delta' = 1/\gamma^2$. Given some $\mathbf{w}$, let this layer raise an alarm only if all the k buckets report alarms. The intuition is that if $\mathbf{w}$ contains more than γ^+ erroneous members, then the probability that every bucket gets at least one such component is high; and if $\mathbf{w}$ contains less than γ^- erroneous members, then the probability that there exists some bucket that is not assigned any erroneous members is also high.

The crucial factor that determines whether a layer could possibly raise an alarm is the distribution of erroneous components into buckets. The event that all buckets raise alarms is only possible if each bucket contains at least one inconsistent component. Let us consider all the inconsistent components in $\mathbf{w}$ in some order, say $w_1, w_2, \ldots$, and think of each of them as a collector that randomly picks a bucket to "collect". Assume for now that we have enough inconsistent elements, and let the random variable Y denote the number of inconsistent components required to collect all the buckets, i.e., Y is the smallest i such that $w_1, \ldots, w_i$ have collected all the buckets. Then the problem becomes an instantiation of the *coupon collector's problem* [17] (viewing buckets as coupons and erroneous components as trials). With k buckets, it is known that $E[Y] = k \ln k + O(k)$, therefore we set k such that $\gamma = \lceil k \ln k \rceil$. It is easy to see that $k = O(\gamma / \ln \gamma)$, hence the desired storage requirement. A detailed analysis of PIRS$^{\pm\gamma}$ can be found in [15].

Finally, we use the technique discussed in Section V-A to generate γ^+-wise independent random numbers, by storing $O(\gamma^+) = O(\gamma)$ truly random numbers per layer. We have thus obtained the desired results:

Theorem 8: PIRS$^{\pm\gamma}$ uses $O(\gamma \log \frac{1}{\delta}(\log m + \log n))$ bits of space, spends $O(\gamma \log \frac{1}{\delta})$ time to process an update and $O(|\mathbf{w}|(\gamma + \log \frac{m}{|\mathbf{w}|}) \log \frac{1}{\delta})$ time to perform a verification.

VI. Tolerance for Small Errors

In this section we prove the hardness of solving CQV$^{\eta}$ (Definition 3) using sub-linear space, even if approximations are allowed.

Theorem 9: Let η and $\delta \in (0, 1/2)$ be user specified parameters. Let $\mathcal{X}$ be any synopsis built on $\mathbf{v}$ that given $\mathbf{w}$: 1. Raises an alarm with probability at most δ if $\mathbf{w} \approx_\eta \mathbf{v}$; and 2. Raises an alarm with probability at least $1-\delta$ if $\mathbf{w} \not\approx_{(2-\epsilon)\eta} \mathbf{v}$ for any $\epsilon > 0$. Then $\mathcal{X}$ has to use $\Omega(n)$ bits.

Proof: We will reduce from the problem of approximating the infinity frequency moment, defined as follows. Let $A = (a_1, a_2, \ldots)$ be a sequence of elements from set $\{1, \ldots n\}$. The *infinity frequency moment*, denoted by F_∞, is the number of occurrences of the most frequent element. Alon et al. [12] showed that any randomized algorithm that makes one pass over A and computes F_∞ with a relative error of at most $1/3$ and a success probability greater than $1 - \delta$ for any $\delta < 1/2$, has to use $\Omega(n)$ memory bits. In particular, they proved that even if each element appears at most twice, it requires $\Omega(n)$ bits in order to decide if F_∞ is 1 or 2 with probability at least $1 - \delta$. Let $\mathcal{X}$ by a synopsis solving the problem in Theorem 9. We will show that $\mathcal{X}$ can compute the infinity frequency moment for any A in which each element appears at most twice, in one pass. For any element i, we update $\mathcal{X}$ with $s = (i, \eta)$. In the end, we verify $\mathbf{w} = 0$ using $\mathcal{X}(\mathbf{v})$. If $\mathcal{X}$ asserts that $\mathbf{w} \approx_\eta \mathbf{v}$, we return $F_\infty = 1$; if $\mathcal{X}$ asserts that $\mathbf{w} \not\approx_{(2-\epsilon)\eta} \mathbf{v}$, we return $F_\infty = 2$. It is not difficult to see that we have thus computed the correct F_∞ with probability at least $1 - \delta$. ∎

The problem remains difficult for relative instead of absolute errors, as can be shown by setting $s = (i, n)$ for element i, and doing the verification with $\mathbf{w} = (n/(1+\eta), \cdots, n/(1+\eta))$ in the proof above.

VII. Extensions

In this section we discuss several extensions of PIRS. We will focus on PIRS-1 for count queries only; the same arguments apply to sum queries, as well as to PIRS-2, PIRS$^{\gamma}$, and PIRS$^{\pm\gamma}$.

Handling Multiple Queries. The discussion so far focused on handling a single query per PIRS synopsis. Our techniques can be used for handling multiple queries simultaneously. Consider a number of aggregate queries on a single attribute (e.g., packet size) but with different partitioning on the input tuples (e.g., source/destination IP and source/destination port). Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ be k such queries. A simple solution for this problem would be to apply the PIRS algorithm once per query, using space linear in k. Interestingly, by treating all the queries as one unified query of n groups we can use one PIRS synopsis to verify the combined vector $\mathbf{v}$. The time cost for processing one update increases linearly in k, since each incoming tuple is updating k components of $\mathbf{v}$ at once (one group for every query in the worst case):

Corollary 1: PIRS-1 for k queries occupies $O(\log \frac{m}{\delta} + \log n)$ bits of space, spends $O(k)$ time to process an update, and $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$ time to perform a verification.

This is a very strong result, since we can effectively verify multiple queries with a few words of memory.

Handling sliding windows. Another nice property of PIRS is that it is decomposable, i.e., for any $\mathbf{v}_1, \mathbf{v}_2$, $\mathcal{X}(\mathbf{v}_1 + \mathbf{v}_2) = \mathcal{X}(\mathbf{v}_1) \cdot \mathcal{X}(\mathbf{v}_2)$ (and for PIRS-2 $\mathcal{X}(\mathbf{v}_1 + \mathbf{v}_2) = \mathcal{X}(\mathbf{v}_1) + \mathcal{X}(\mathbf{v}_2)$). This property allows us to extend PIRS for periodically sliding windows using standard techniques [20]. Take for example the following query:

```
SELECT SUM(packet_size) FROM IP_Trace
GROUP BY source_ip, destination_ip
WITHIN LAST 1 hour SLIDE EVERY 5 minutes
```

In this case, we build one PIRS-1 for every 5-minute period, and keep it in memory until it expires from the sliding window. Assume that there are k such periods in the window, and let $\mathcal{X}(\mathbf{v}_1), \ldots, \mathcal{X}(\mathbf{v}_k)$ be the PIRS for these periods. When the server returns a result $\mathbf{w}$, the client computes the overall $\mathcal{X}(\mathbf{v}) = \prod_{i=1}^{k} \mathcal{X}(\mathbf{v}_i)$, and verifies the result.

Corollary 2: For a periodically sliding window query with k periods, our synopsis uses $O(k(\log \frac{m}{\delta} + \log n))$ bits of space, spends $O(1)$ time to process an update, and $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$ time to perform a verification.

Synchronization. In our discussions we omitted superscript τ for simplicity. Hence, an implicit assumption was made that the result $\mathbf{w}^{\tau_s}$ returned by the server was synchronized with $\mathcal{X}(\mathbf{v}^{\tau_c})$ maintained by the client, i.e., $\tau_s = \tau_c$. Correct verification can be performed only if the server and the client are synchronized. Obviously, such perfect synchronization is hard to obtain in practice. Also, if n is large, transmitting the result itself takes non-negligible time. The solution to this problem is as follows. Suppose that the client sends out a request to the server asking for the query result at time τ, ($\tau \geq t_{now}$). When the client receives s^{τ} and computes synopsis for $\mathcal{X}(\mathbf{v}^{\tau})$, it makes a copy and continues updating PIRS. When the server returns answer $\mathbf{w}^{\tau}$, the client can do the verification using the copy. The synchronization problem once again illustrates the importance of using small space, as keeping a copy (or potentially many copies if there are significant delays in the server's response) is expensive. Similar ideas can be used on the server side for dealing with queries referring to the past.

Exploiting locality. In many practical situations data streams tend to exhibit a large degree of locality. Simply put, updates to $\mathbf{v}$ tend to cluster to the same components. In this setting, it is possible to explore space/time trade-offs. We can allocate a small buffer used for storing exact aggregate results for a small number of groups. With data locality, a large portion of updates will hit the buffer. Whenever the buffer is full and a new group needs to be inserted, a victim is selected from the buffer using the simple *least recently used (LRU)* policy. Only then does the evicted group update PIRS, using the overall aggregate value computed within the buffer. We flush the buffer to update PIRS whenever a verification is required. Since we are aggregating the incoming updates in the buffer and update the synopsis in bulk, we incur a smaller, amortized update processing cost per tuple.

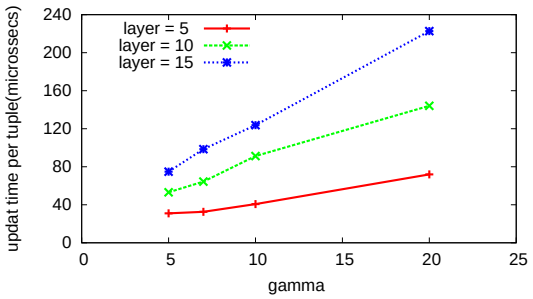
VIII. Empirical Evaluation

In this section we evaluate the performance of the proposed synopses over two real data streams [21], [22]. The experimental study demonstrates that our synopses: 1. use very small space; 2. support fast updates; 3. have very high accuracy; 4. support multiple queries; and 5. are easy to implement.

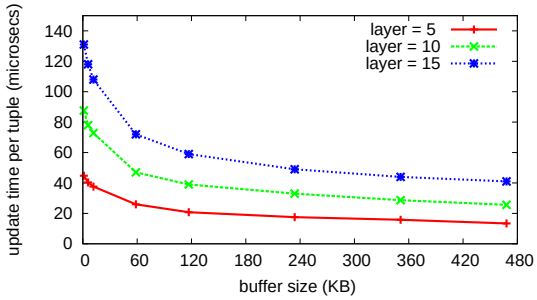
A. Setup

Our synopses are implemented using GNU C++ and the GNU GMP extension which provides arbitrary precision arithmetic, useful for operating on numbers longer than 32 bits. The experiments were run on an Intel Pentium 2.8GHz CPU with 512KB L2 cache and 512MB of main memory. Our results show that using our techniques, even a low-end client machine can efficiently verify online queries with millions of groups on real data streams.

The *World Cup (WC)* data stream [21] consists of web server logs for the 1998 Soccer World Cup. Each record in the log contains several attributes such as a timestamp, a client id, a requested object id, a response size, etc. We used the request streams of days 46 and 47 that have about 100 millions records. The *IP traces (IPs)* data stream [22] is collected over the AT&T backbone network; each tuple is a TCP/IP packet header. Here, we are interested in analyzing the source IP/port, destination IP/port, and packet size header fields. The data set consists of a segment of one day traffic and has 100 million packets. Without loss of generality, unless otherwise stated, we perform the following default query: 1. Count or Sum (on response size) query group-by client id/object id for the WC data set; 2. Count or Sum (on packet size) query

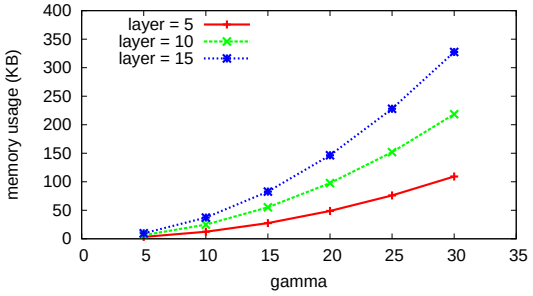


(a) Update cost per tuple.

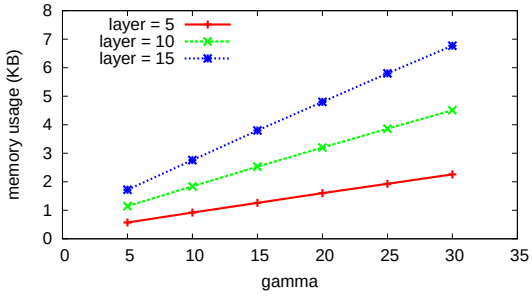


(b) Space time tradeoff.

Fig. 3. PIRS$^{\gamma}$,PIRS$^{\pm\gamma}$: running time.



(a) PIRS$^{\gamma}$.



(b) PIRS$^{\pm\gamma}$.

Fig. 4. PIRS$^{\gamma}$, PIRS$^{\pm\gamma}$: memory usage.

TABLE I

AVERAGE UPDATE TIME PER TUPLE.

	WC	IPs
Count	0.98 μs	0.98 μs
Sum	8.01 μs	6.69 μs

group-by source IP/destination IP for the IPs data set. Each client id, object id, IP address, the response size, or the packet size is a 32-bit integer. Thus, the group id is 64-bit long (by concatenating the two grouping attributes), meaning a potential group space of $n = 2^{64}$. The number of nonzero groups is of course far lower than n: WC has a total of 50 million nonzero groups and IPs has 7 million nonzero groups.

B. PIRS

A very conservative upper bound for the total response size and packet size is $m = 10^{10} \ll n \approx 2 \times 10^{19}$ for all cases in our experiments. So from our analysis in Section IV, PIRS-1 is clearly the better choice, and is thus used in our experiments. We precomputed p as the smallest prime above 2^{64} and used the same p throughout this section. Thus, each word (storing p, α, and $\mathcal{X}(\mathbf{v})$) occupies 9 bytes.

Space usage. As our analysis has pointed out, PIRS uses only 3 words, or 27 bytes for our queries. This is in contrast to the naive solution of keeping the exact value for each nonzero group, which would require 600MB and 84MB of memory, respectively.

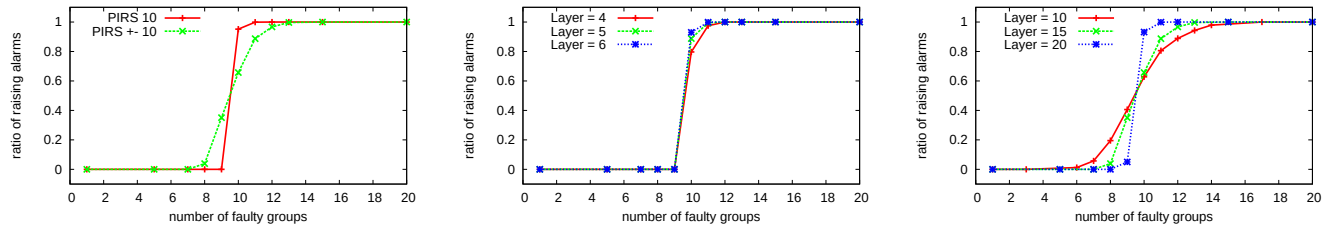
Update cost. PIRS has excellent update cost which is crucial to the streaming algorithm. The average per-tuple update cost is shown in Table I for Count and Sum queries on both WC and IPs. The update time for the two count queries stays the same regardless of the data set, since an update always incurs one addition, one multiplication, and one modulo. The update cost for sum queries is higher, since we need $O(\log u)$ time for exponentiation. The cost on WC is slightly larger as its average u is larger than that of IPs. Nevertheless, PIRS is still extremely fast in all cases, and is able to process more than 10^5 tuples (10^6 tuples for count queries) per second.

Detection accuracy. As guaranteed by the theoretical analysis, the probability of failure of PIRS-1 is $\delta \leq m/p$, which is at most 0.5×10^{-9}. Note that our estimate of m is very conservative; the actual δ is much smaller. We generated $100,000$ random attacks and, not surprisingly, PIRS identified all of them.

C. PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$

For the rest of the experiments, we focus on the Count query on the WC data set. Similar patterns have been observed on the IPs data set.

Update cost. In this set of experiments we study the performance of PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$. Clearly, PIRS$^{\gamma}$ has linear update cost w.r.t the number of layers and γ (the number of inconsistent groups to detect), as confirmed in Figure 3(a). It is not hard to see that PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$ have almost the same update cost if they are configured with the same number of layers. Essentially, each one has to generate the γ-wise (or γ^{+}-wise) independent random numbers on-the-fly and update



(a) $\gamma = 10, \ell = 15$. (b) PIRS$^{\gamma}$, $\gamma = 10$. (c) PIRS$^{\pm\gamma}$, $\gamma = 10$.

Fig. 5. Detection with tolerance for limited number of errors.

one PIRS synopsis at each layer. Hence, we only show the cost for PIRS$^{\gamma}$. However, the space cost of the two synopses is different. PIRS$^{\gamma}$, as an exact solution for CQV$^{\gamma}$, is expected to use much larger space than its counterpart PIRS$^{\pm\gamma}$, which gives approximate solutions. This is demonstrated in Figure 4. By construction, at each layer PIRS$^{\gamma}$ has $O(\gamma^2)$ and PIRS$^{\pm\gamma}$ $O(\frac{\gamma}{\ln \gamma})$ buckets, which is easily observed in Figure 4(a) and 4(b) respectively.

Space/Time Trade-offs. If the client can afford to allocate some extra space, but still cannot store the entire vector $\mathbf{v}$, as discussed in Section VII, it is possible to exploit the locality in the input data streams to reduce the amortized update cost. A simple LRU buffer has been added to PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$ and its effect on update cost is reported in Figure 3(b) with $\gamma = 10$. Again, both synopses exhibit very similar behavior. As the figure indicates, a very small buffer (up to 500 KB) that fits into the cache is able to reduce the update cost by an order of magnitude. The improvement on the update cost of this buffering technique depends on the degree of locality in the data stream.

Detection accuracy. We observed that both of our synopses can achieve excellent detection accuracy as the theoretical analysis suggests. All results reported here are the ratios obtained from $100,000$ rounds. Since the detection mechanism of the synopses does not depend on the data characteristics, both data sets give similar results. Figure 5(a) shows the ratios of raising alarms versus the number of actual inconsistent groups, with $\gamma = 10$ and 10 layers. As expected, PIRS$^{\gamma}$ has no false positives and almost no false negatives; only very few false negatives are observed with 10 and 11 actual inconsistent groups. On the other hand, PIRS$^{\pm\gamma}$ has a transition region around γ and it does have false positives. Nevertheless, the transition region is sharp and once the actual number of inconsistent groups is slightly off γ, both false positives and negatives reduce to zero. We have also studied the impact of the number of layers on the detection accuracy. Our theoretical analysis gives provable bounds. For example with PIRS$^{\gamma}$ the probability of missing an alarm is at most $1/2^{\ell}$ (for ℓ layers). In practice, the probability is expected to be even smaller. We repeated the same experiments using different layers, and Figure 5(b) reports the result for PIRS$^{\gamma}$. With less layers (4–6) it still achieves excellent detection accuracy. Only when the number of actual inconsistent groups is close to γ, a small drop in the detection ratio is observed. Figure 5(c) reports the same experiment for PIRS$^{\pm\gamma}$ with layers from 10 to 20. Smaller number of layers enlarges the transition region and larger number of layers sharpens it. Outside this region, 100% detection ratio is always guaranteed. Finally, experiments have been performed over different values of γ's and similar behavior has been observed.

TABLE II
UPDATE TIME AND MEMORY USAGE OF PIRS FOR MULTIPLE QUERIES.

# queries	5	10	15	20
update time (μs)	5.0	9.9	14.9	19.8
memory usage (bytes)	27	27	27	27

D. Multiple Queries

Our final set of experiments investigates the effect of multiple, simultaneous queries. Without loss of generality, we simply execute the same query a number of times. Note that the same grouping attributes with different query ids are considered as different groups. We tested with 5, 10, 15, and 20 queries in the experiments. Note that on the WC data set, the exact solution would use 600MB for each query, hence 12GB if there are 20 queries. Following the analysis in Section VII, our synopses naturally support multiple queries and still have the same memory usage as if there was only one query. Nevertheless, the update costs of all synopses increase linearly with the number of queries. In Table II we report the update time and memory usage for PIRS; similar results were observed for PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$.

IX. Related Work

As discussed in Section III, sketching techniques cannot solve the verification problem [12], [13], [14], [23]. Various other authentication techniques though are related. By viewing $\mathbf{v}$ as a message, the client can compute an authenticated signature $\sigma(\mathbf{v})$ and any alteration to $\mathbf{v}$ will be detected. However, a fundamental problem is performing incremental updates at the client side, without storing $\mathbf{v}$. Considerable effort has been devoted in *incremental cryptography* [24] for that purpose. In particular, incremental signatures and incremental MAC [24] are closely related, but they support updates only for *block edit operations*, and not arithmetic updates. There is also considerable work on authenticated queries in outsourced databases [7], [25], but they do not apply

for online, one-pass streaming scenarios. Work in [26], [27] has studied secure in-network aggregation in sensor networks using cryptographic tools. Nevertheless, the problem setting therein is fundamentally different from the one discussed here. Authentication of sliding window queries and continuous queries over data streams have been studied in [28], [29]. However, in that model the client does not have access to the data stream and the data publisher is responsible for injecting "proofs" into the stream.

Verifying the identity of polynomials is a fingerprinting technique, i.e., a method for efficient probabilistic checking of equality between two elements from a large universe U. Fingerprint algorithms employ algebraic techniques combined with randomization, like PIRS. Examples include verifying univariate polynomial multiplication, multivariate polynomial identities, and verifying equality of strings [16], [17], [30]. An excellent related discussion appears in [17].

X. Conclusion

Verifying query results in an outsourced data stream setting is a problem that has not been addressed before. We proposed various space/time efficient probabilistic algorithms for selection and aggregation queries. First, we proposed algorithms for detecting errors with high confidence. Then, we extended the algorithms to permit a pre-defined number of arbitrary errors. Finally, we proved that permitting arbitrary errors of small magnitude is hard. The experimental evaluation showed that the proposed techniques work very well in practice. In the future, we plan to investigate the problem of query assurance on more complex queries, such as joins.

Acknowledgments

This work was the outcome of extended visits of the first and second authors at AT&T Labs. Ke Yi is supported in part by Hong Kong Direct Allocation Grant (DAG07/08). Feifei Li and George Kollios were supported in part by the NSF grant IIS-0133825. The authors would like to thank Graham Cormode for many insightful discussions on this problem.

References

[1] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, D. Madden, V. Raman, F. Reiss, and M. A. Shah, "TelegraphCQ: continuous dataflow processing for an uncertain world," in *Proc. of Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.

[2] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. M. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong, "Nile: A query processing engine for data streams," in *Proc. of International Conference on Data Engineering (ICDE)*, 2004, p. 851.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom, "STREAM: The Stanford stream data manager," *IEEE Data Engineering Bulletin*, vol. 26, no. 1, pp. 19–26, 2003.

[4] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk, "Gigascope: A stream database for internet databases," in *Proc. of ACM Management of Data (SIGMOD)*, 2003, pp. 647–651.

[5] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring streams—a new class of data management applications," in *Proc. of Very Large Data Bases (VLDB)*, 2003, pp. 215–226.

[6] D. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *CIDR*, 2005.

[7] R. Sion, "Query execution assurance for outsourced databases," in *Proc. of Very Large Data Bases (VLDB)*, 2005, pp. 601–612.

[8] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in *Proc. of Very Large Data Bases (VLDB)*, 2003, pp. 309–320.

[9] R. Zhang, N. Koudas, B. C. Ooi, and D. Srivastava, "Multiple aggregations over data streams," in *Proc. of ACM Management of Data (SIGMOD)*, 2005, pp. 299–310.

[10] S. Muthukrishnan, "Data streams: algorithms and applications," 2003.

[11] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, 2002.

[12] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. of ACM Symposium on Theory of Computing (STOC)*, 1996, pp. 20–29.

[13] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[14] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[15] K. Yi, F. Li, M. Hadjieleftheriou, D. Srivastava, and G. Kollios, "Randomized synopses for query verification on data streams," AT&T Labs Inc., Tech. Rep., 2007, http://cs-people.bu.edu/lifeifei/papers/pirs.pdf.

[16] R. Freivalds, "Fast probabilistic algorithms." in *Proc. of International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 1979, pp. 57–69.

[17] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[18] T. Nagell, *Introduction to Number Theory*, 2nd ed. Chelsea Publishing Company, 1981.

[19] M. N. Wegman and J. L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, vol. 22, no. 3, 1981.

[20] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," in *Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pp. 635–644.

[21] M. Arlitt and T. Jin, http:// www.acm.org/ sigcomm/ITA/, iTA, 1998 World Cup Web Site Access Logs.

[22] "AT&T network traffic streams," AT&T Labs.

[23] G. S. Manku and R. Motwani, "Approximate Frequency Counts over Data Streams," in *Proc. of Very Large Data Bases (VLDB)*, 2002, pp. 346–357.

[24] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *Proc. of Advances in Cryptology (CRYPTO)*, 1994, pp. 216–233.

[25] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan, "Verifying completeness of relational query results in data publishing," in *Proc. of ACM Management of Data (SIGMOD)*, 2005, pp. 407–418.

[26] H. Chan, A. Perrig, and D. Song, "Secure hierarchical in-network aggregation in sensor networks," in *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2006, pp. 278–287.

[27] M. Garofalakis, J. M. Hellerstein, and P. Maniatis, "Proof sketches: Verifiable in-network aggregation," in *Proc. of International Conference on Data Engineering (ICDE)*, 2007.

[28] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios, "Proof-infused streams: Enabling authentication of sliding window queries on streams," in *Proc. of Very Large Data Bases (VLDB)*, 2007.

[29] S. Papadopoulos, Y. Yang, and D. Papadias, "CADS: Continuous authentication on data streams," in *Proc. of Very Large Data Bases (VLDB)*, 2007.

[30] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.