Data Stream Processing
(Part III)

SURVEY 1: S. Muthukrishnan. "Data Streams: Algorithms and Applications".

Streaming Model: Special Cases

- Time-Series Model
  - Only j-th update updates A[j] (i.e., A[j]:= c[j])
- Cash-Register Model
  - c[j] is always := 0 (i.e., increment-only)
  - Typically, c[j]>1, so we see a multi-set of items in one pass
- Turnstile Model
  - Most general streaming model
  - c[j] can be ±0 or ±1 (i.e., increment or decrement)
  - Problem difficulty varies depending on the model
  - E.g., MIN/MAX in Time-Series vs. Turnstile!

The Streaming Model

- Underlying signal: One-dimensional array A[1...N] with values A[i] all initially zero
  - Multi-dimensional arrays as well (e.g., row-major)
- Signal is implicitly represented via a stream of updates
  - j-th update is <k, c[j]>
  - c[j] can be ±0, ±1
- Goal: Compute functions on A[] subject to
  - Small space
  - Fast processing of updates
  - Fast function computation

Probabilistic Guarantees

- Example: Actual answer is within 5 ± 1 with prob ≥ 0.9
- Randomized algorithms: Answer returned is a specially-built random variable
- User-tunable (ε,δ)-approximations
  - Estimate is within a relative error of ε with probability ≥ 1-δ
- Use Tail Inequalities to give probabilistic bounds on returned answer
  - Markov Inequality
  - Chebyshev’s Inequality
  - Chernoff Bound
  - Hoeffding Bound

Data-Stream Processing Model

- Approximate answers often suffice, e.g., trend analysis, anomaly detection
- Requirements for stream synopsis
  - Single Pass: Each record is examined at most once, in (fixed) arrival order
  - Small Space: Log or polylog in data stream size
  - Real-time: Per-record processing time (to maintain synopsis) must be low
  - Delete-Proof: Can handle record deletions as well as insertions
  - Composable: Built in a distributed fashion and combined later

Linear-Projection (aka AMS) Sketch Synopses

- Goal: Build small-space summary for distribution vector f(i)(t) in (stream of i-values)
- Basic Construct: Randomized Linear Projection of f = project onto inner-dot product of f-vector
  - Simple to compute over the stream: Add f[i] whenever the i-th value is seen
  - Generate f[i] is small (logN) space using pseudo-random generators
  - Turnstile probabilistic guarantees on approximation error
  - Delete-Proof: Just subtract f[i] to delete an i-th value occurrence
  - Composable: Simply add independently-built projections
Overview

• Introduction & Motivation
• Data Streaming Models & Basic Mathematical Tools
• Summarization/Sketching Tools for Streams
  - Sampling
  - Linear-Projection (aka AMS) Sketches
    • Applications: Join/Multi-Join Queries, Wavelets
  - Hash (aka FM) Sketches
    • Applications: Distinct Values, Distinct Sampling, Set Expressions

Distinct Value Estimation

• Problem: Find the number of distinct values in a stream of values with
  domain \{0, \ldots, N-1\}
  - Zerth frequency moment \( F_0 \), L0 (Hamming) stream norm
  - Statistics: number of species or classes in a population
  - Important for query optimizers
  - Network monitoring: distinct destination IP addresses, source/destination pairs, requested URLs, etc.

• Example (N=64)
  • Data stream: 5 0 5 3 0 1 7 5 1 0 3 7
  • Number of distinct values: 5

  • Hard problem for random sampling (CC-MNS00)
    - Must sample almost the entire table to guarantee the estimate is
      within a factor of 10 with probability 1/2, regardless of the
      estimator used

Hash (aka FM) Sketches for Distinct Value Estimation [FM85]

• Assume a hash function \( h(x) \) that maps incoming values \( x \) in \([0, \ldots, N-1]\)
  uniformly across \([0, \ldots, 2^L-1]\), where \( L = \Theta(\log N) \)

• Let \( \text{lab}(y) \) denote the position of the least-significant 1 bit in the binary
  representation of \( y \)
  - A value \( x \) is mapped to \( \text{lab}(h(x)) \)

• Maintain \( \text{Hash Sketch} = \text{BITMAP} \) array of \( L \) bits, initialized to 0
  - For each incoming value \( x \), set \( \text{BITMAP} [\text{lab}(h(x))] = 1 \)

\( x = 5 \Rightarrow h(x) = 101100 \Rightarrow \text{lab}(h(x)) = 2 \)

<table>
<thead>
<tr>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Hash Sketches for Distinct Value Estimation

(FM85) assume "ideal" hash functions \( h(x) \) (N-wise independence)

• [AMS96]: pairwise independence is sufficient
  - \( h(x) = (a \cdot x + b) \mod N \), where \( a, b \) are random binary vectors
    in \([0, \ldots, 2^L-1]\)
  - Small-space \((c,d)\) estimates for distinct values proposed based on
    FM ideas

• Delete-Proof: Just use counters instead of bits in the sketch locations
  - +1 for inserts, -1 for deletes

• Composable: Component-wise OR/ADD distributed sketches together
  - Estimate \( |S_1 \cup S_2 \cup \ldots \cup S_k| \) = set-union cardinality

Generalization: Distinct Values Queries

• SELECT COUNT(DISTINCT target-attr)
  • FROM relation
  • WHERE predicate

• SELECT COUNT(DISTINCT a, c, t, ikey)
  • FROM orders
  • WHERE a_orderdate > '2002-01-01'
  - "How many distinct customers have placed orders this year?"
  - Predicate not necessarily on the DISTINCT target attribute

• Approximate answers with error guarantees over a stream of tuples
Distinct Sampling [Gib01]

Key Ideas
- Use FM-like technique to collect a specially-tailored sample over the distinct values in the stream
  - Use hash function mapping to sample values from the data domain
  - Uniform random sample of the distinct values
  - Very different from traditional random sample: each distinct value is chosen uniformly regardless of its frequency
  - DISTINCT query answers: simply scale up sample answer by sampling rate
- To handle additional predicates
  - Reservoir sampling of tuples for each distinct value in the sample
  - Use reservoir sample to evaluate predicates

Using the Distinct Sample [Gib01]
- If level \( l \) for our sample, then we have selected all distinct values \( x \) such that \( h(x) = l \)
  - \( \text{Prob}(h(x) = l) = \frac{1}{l} \)
  - By \( h(x) \)'s randomizing properties, we have uniformly sampled a fraction of the distinct values in our stream
  - Our sampling rate!
- Query Answering: Run distinct-values query on the distinct sample and scale the result up by \( l \)
- Distinct-value estimation: Guarantees a relative error with probability \( 1 - \delta \) using \( O(\log (1/\delta)/\epsilon^2) \) space
  - For \( \epsilon \times \delta \) selectivity predicates the space goes up inversely with \( \delta \)
- Experimental results: 0.01% error vs. 50-250% error for previous best approaches, using 0.2% to 10% synopses

Building a Distinct Sample [Gib01]
- Use FM-like hash function \( h() \) for each streaming value \( x \)
  - \( \text{Prob}(h(x) = k) = \frac{1}{2^k} \)
- Key Invariant: "All values with \( h(x) \geq k \) (and only these) are in the distinct sample"

Distinct Sampling Example
- \( B = 3, N = 8 \) (\( r = 0 \) to simplify example)
  \[ 0 1 3 5 7 \]

Distinct Sampling Example
- \( B = 3, N = 8 \) (\( r = 0 \) to simplify example)
  \[ 0 1 3 5 7 \]

Processing Set Expressions over Update Streams [G6R03]
- Estimate cardinality of general set expressions over streams of updates
  - E.g., number of distinct \((\text{source,dest})\) pairs seen at both \( R_1 \) and \( R_2 \) but not \( R_3 \)
  - \( R_1 \subseteq R_2 \subseteq R_3 \)
- 2-Level Hash-Sketch (2LHS) stream synopsis: Generalizes FM sketch
  - First level: \( \log(N) \) buckets with exponentially-decreasing probabilities (using \( \log(h(x)) \), as in FM)
  - Second level: Count signature array (\( \log(W) \) counters)
    - One "total count" for elements in first-level bucket
    - \( \log(h(x)) \) bit-location counts for \( b \)-bits of incoming elements
    - \( \text{insert}(17) \to \text{ld}(h(17)) \)

Processing Set Expressions over Update Streams: Key Ideas
- Build several independent 2LHS, fix a level \( l \), and look for singleton first-level buckets at that level
  - \( \text{lookup}(l) \to \text{count}(l) \to \text{buckets}(l) \to \text{signature}(l) \to \text{singleton}(l) \)
  - Singleton buckets and singleton element (in the bucket) are easily identified using the count signature
  - Singletons discovered form a "distinct value sample" from the union of the streams
  - Frequency-independent, each value sampled with probability \( \frac{1}{N} \)
  - Determine the fraction of "witnesses" for the set expression \( E \) in the sample, and scale-up to find the estimate for \( |E| \)
**Example: Set Difference, |A-B|**

- Parallel (same hash function), independent 2LHS synopses for input streams A, B
- Assume robust estimate $\hat{\mu}$ for $|A\cup B|$ (using known FM techniques)
- Look for buckets that are singletons for $A\cup B$ at level $1-\log\hat{\mu}$
  - Prob[singleton at level $1$] $\leq$ constant (e.g., $1/4$)
  - Number of singletons (i.e., size of distinct sample) is at least a constant fraction (e.g., $1/6$) of the number of 2LHS (w.h.p.)
- "Witness" for set difference A-B: Bucket is singleton for stream A and empty for stream B
  - Prob[witness | singleton] $= |A-B| / |A\cup B|$
- Estimate for $|A-B| = \#$ witnesses for $A-B$ / $\#$ singleton buckets

**Estimation Guarantees**

- Our set-difference cardinality estimate is within a relative error of $\epsilon$ with probability $\geq 1-\delta$ when the number of 2LHS is $\Omega\left(\frac{|A\cup B|}{\epsilon^2} \log(1/\delta)\right)$
- Lower bound of $\Omega\left(\frac{|A\cup B|}{\epsilon^2} \log(1/\delta)\right)$ space, using communication-complexity arguments
- Natural generalization to arbitrary set expressions $E = f(S_1, \ldots, S_n)$
  - Build parallel, independent 2LHS for each $S_1, \ldots, S_n$
  - Generalize "witness" condition (inductively) based on E's structure
    - $(\epsilon, \delta)$ estimate for $|E|$ using $O\left(\frac{|S_1| \ldots |S_n|}{\epsilon^2} \log(1/\delta)\right)$
      2LHS synopses
- Worst-case bounds! Performance in practice is much better [GGR03]

**Extensions**

- Key property of FM-based sketch structures: Duplicate-insensitive!
  - Multiple insertions of the same value don't affect the sketch or the final estimate
  - Makes them ideal for use in broadcast-based environments
    - E.g., wireless sensor networks (broadcast to many neighbors is critical for robust data transfer)
    - Considine et al. ICU'04; Manki et al. SIGMOD'05
- Main deficiency of traditional random sampling: Does not work in a Turnstile Model (inserts-deletes)
  - "Adversarial" deletion stream can deplete the sample
- Exercise: Can you make use of the ideas discussed today to build a "delete-proof" method of maintaining a random sample over a stream??