Data Stream Processing (Part IV)

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 $<k, c[j]>$ implying
    - $A[k] := A[k] + c[j]$  ($c[j]$ can be $>0$, $<0$)

• **Goal**: Compute functions on $A[]$ subject to
  - Small space
  - Fast processing of updates
  - Fast function computation
  - ...
Streaming Model: Special Cases

- **Time-Series Model**

- **Cash-Register Model**
  - $c[j]$ is always $\geq 0$ (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 $<0$ (i.e., increment or decrement)

- *Problem difficulty varies depending on the model*
  - E.g., MIN/MAX in Time-Series vs. Turnstile!
Data-Stream Processing Model

- **Approximate answers often suffice**, e.g., trend analysis, anomaly detection
- **Requirements for stream synopses**
  - *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 synopses) must be low
  - *Delete-Proof*: Can handle record deletions as well as insertions
  - *Composable*: Built in a *distributed fashion* and combined later

(GigaBytes)  
Continuous Data Streams

Stream Synopses (in memory)

(KiloBytes)  
Approximate Answer with Error Guarantees “Within 2% of exact answer with high probability”

Stream Processing Engine

Query Q

R1

Rk
Probabilistic Guarantees

• Example: Actual answer is within $5 \pm 1$ with prob $\geq 0.9$

• **Randomized algorithms:** Answer returned is a specially-built random variable

• **User-tunable $(\varepsilon, \delta)$-approximations**
  - Estimate is within a relative error of $\varepsilon$ with probability $\geq 1-\delta$

• Use **Tail Inequalities** to give probabilistic bounds on returned answer
  - **Markov Inequality**
  - **Chebyshev’s Inequality**
  - **Chernoff Bound**
  - **Hoeffding Bound**
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
Linear-Projection (aka AMS) Sketch Synopses

- **Goal:** Build small-space summary for distribution vector $f(i)$ ($i=1,\ldots, N$) seen as a stream of $i$-values

Data stream: $3, 1, 2, 4, 2, 3, 5, \ldots$

- **Basic Construct:** Randomized Linear Projection of $f() = \text{project onto inner/dot product of f-vector}$

$$< f, \xi > = \sum f(i) \xi_i \quad \text{where} \quad \xi = \text{vector of random values from an appropriate distribution}$$

- Simple to compute over the stream: Add $\xi_i$ whenever the $i$-th value is seen

Data stream: $3, 1, 2, 4, 2, 3, 5, \ldots$

$$\xi_1 + 2\xi_2 + 2\xi_3 + \xi_4 + \xi_5$$

- Generate $\xi_i$ 's in small $(\log N)$ space using pseudo-random generators
- **Tunable probabilistic guarantees** on approximation error
- **Delete-Proof:** Just subtract $\xi_i$ to delete an $i$-th value occurrence
- **Composable:** Simply add independently-built projections
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 $L = O(\log N)$

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

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

$x = 5 \rightarrow h(x) = 101100 \rightarrow \text{lsb}(h(x)) = 2$

BITMAP

\[\begin{array}{cccccc}
5 & 4 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0
\end{array}\]
Hash (aka FM) Sketches for Distinct Value Estimation [FM85]

- By uniformity through \( h(x) \): \( \text{Prob}[\text{BITMAP}[k]=1] = \text{Prob}[10^k] = \frac{1}{2^{k+1}} \)
  - Assuming \( d \) distinct values: expect \( d/2 \) to map to BITMAP[0], \( d/4 \) to map to BITMAP[1], …

\[ \begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array} \]

- Let \( R = \text{position of rightmost zero in BITMAP} \)
  - Use as indicator of \( \log(d) \)

- Average several iid instances (different hash functions) to reduce estimator variance
Generalization: Distinct Values Queries

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

- SELECT COUNT( DISTINCT o_custkey )
- FROM orders
- WHERE o_orderdate >= '2002-01-01'

  - “How many distinct customers have placed orders this year?”
  - Predicate not necessarily only 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
Processing Set Expressions over Update Streams [GGRO3]

- Estimate cardinality of *general set expressions* over streams of updates
  - E.g., number of distinct (source,dest) pairs seen at both R1 and R2 but not R3? \(| (R1 \cap R2) - R3 |\)

- **2-Level Hash-Sketch (2LHS) stream synopsis**: Generalizes FM sketch
  - *First level*: \(\Theta(\log N)\) buckets with exponentially-decreasing probabilities (using \(\text{lsb}(h(x))\), as in FM)
  - *Second level*: Count-signature array (\(\log N+1\) counters)
    - One “total count” for elements in first-level bucket
    - \(\log N\) “bit-location counts” for 1-bits of incoming elements

**Diagram:**
- \(\text{insert}(17) \rightarrow \text{lsb}(h(17))\)
- \(17 = 0 0 0 0 1 0 0 0 1\)
- **-1 for deletes!!**
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. ICDE’04; Manjhi 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??
New stuff for today...

- A different sketch structure for multi-sets: *The CountMin (CM) sketch*
- The Sliding Window model and Exponential Histograms (EHs)
- Peek into distributed streaming
The CountMin (CM) Sketch

- Simple sketch idea, can be used for point queries, range queries, quantiles, join size estimation
- Model input at each node as a vector $x_i$ of dimension $N$, where $N$ is large
- Creates a small summary as an array of $w \times d$ in size
- Use $d$ hash functions to map vector entries to $[1..w]$
CM Sketch Structure

- Each entry in vector $A$ is mapped to one bucket per row
- Merge two sketches by entry-wise summation
- Estimate $A[j]$ by taking $\min_k \text{sketch}[k, h_k(j)]$

[Cormode, Muthukrishnan ’05]
**CM Sketch Summary**

- **CM sketch** guarantees approximation error on point queries less than $\varepsilon \| A \|_1$ in size $O(1/\varepsilon \log 1/\delta)$
  
  - Probability of more error is less than $1-\delta$
  
  - Similar guarantees for range queries, quantiles, join size

- **Hints**
  
  - Counts are biased! Can you limit the expected amount of extra “mass” at each bucket? (*Use Markov*)
  
  - Use Chernoff to boost the confidence for the min{} estimate

- **Food for thought:** How do the CM sketch guarantees compare to AMS??
Sliding Window Streaming Model

• Model
  - At every time $t$, a data record arrives
  - The record “expires” at time $t+N$ ($N$ is the window length)

• When is it useful?
  - Make decisions based on “recently observed” data
  - Stock data
  - Sensor networks
Time in Data Stream Models

Tuples arrive $X_1, X_2, X_3, \ldots, X_t, \ldots$

- Function $f(X,t,\text{NOW})$
  - Input at time $t$: $f(X_1,1,t)$, $f(X_2,2,t)$, $f(X_3,3,t)$, ..., $f(X_t,t,t)$
  - Input at time $t+1$: $f(X_1,1,t+1)$, $f(X_2,2,t+1)$, $f(X_3,3,t+1)$, ..., $f(X_{t+1},t+1,t+1)$

- Full history: $f = \text{identity}$

- Partial history: Decay
  - Exponential decay: $f(X,t,\text{NOW}) = 2^{-\text{NOW}-t}X$
    - Input at time $t$: $2^{-(t-1)}X_1$, $2^{-(t-2)}X_2$, ..., $\frac{1}{2} \cdot X_{t-1}, X_t$
    - Input at time $t+1$: $2^{-t}X_1$, $2^{-(t-1)}X_2$, ..., $\frac{1}{4} \cdot X_{t-1}, \frac{1}{2} \cdot X_t, X_{t+1}$
  - Sliding window (special type of decay):
    - $f(X,t,\text{NOW}) = X$ if $\text{NOW}-t < N$
    - $f(X,t,\text{NOW}) = 0$, otherwise
    - Input at time $t$: $X_1, X_2, X_3, \ldots, X_t$
    - Input at time $t+1$: $X_2, X_3, \ldots, X_t, X_{t+1}$
Simple Example: Maintain Max

- Problem: Maintain the maximum value over the last N numbers.

- Consider all non-decreasing arrangements of N numbers (Domain size R):
  - There are \((N+R)\ choose N\) distinct arrangements
  - Lower bound on memory required:
    \[ \log(N+R \ choose N) \geq N \log(R/N) \]
  - So if \(R=\text{poly}(N)\), then lower bound says that we have to store the last N elements (\(\Omega(N \log N)\) memory)
Statistics Over Sliding Windows

• Bitstream: Count the number of ones [DGIM02]
  - Exact solution: $\Theta(N)$ bits
  - Algorithm BasicCounting:
    • $1 + \varepsilon$ approximation (relative error!)
    • Space: $O(1/\varepsilon \cdot (\log^2 N))$ bits
    • Time: $O(\log N)$ worst case, $O(1)$ amortized per record
  - Lower Bound:
    • Space: $\Omega(1/\varepsilon \cdot (\log^2 N))$ bits
Approach: Temporal Histograms

Example: ... 01101010011111110110 0101 ...

Equi-width histogram:

... 0110 1010 0111 1111 0110 0101 ...

• Issues:
  
  - Error is in the last (leftmost) bucket.
  - Bucket counts (left to right): $C_m, C_{m-1}, ..., C_2, C_1$
  - Absolute error $\leq C_m/2$.
  - Answer $\geq C_{m-1}+...+C_2+C_1+1$.
  - Relative error $\leq C_m/2(C_{m-1}+...+C_2+C_1+1)$.
  - Maintain: $C_m/2(C_{m-1}+...+C_2+C_1+1) \leq \epsilon (=1/k)$.  


Naïve: Equi-Width Histograms

• Goal: Maintain $C_m/2 \leq \varepsilon (C_{m-1} + \ldots + C_2 + C_1 + 1)$

Problem case:
... 0110 1010 0111 1111 0110 1111 0000 0000 0000 0000 ...

• Note:
  - Every Bucket will be the last bucket sometime!
  - New records may be all zeros ➔
    For every bucket $i$, require $C_i/2 \leq \varepsilon (C_{i-1} + \ldots + C_2 + C_1 + 1)$
Exponential Histograms

• Data structure invariant:
  - Bucket sizes are non-decreasing powers of 2
  - For every bucket size other than that of the last bucket, there are at least \( \frac{k}{2} \) and at most \( \frac{k}{2}+1 \) buckets of that size
  - Example: \( k=4 \): \((8,4,4,4,2,2,2,1,1,\ldots)\)

• Invariant implies:
  - Assume \( C_i=2^j \), then
    • \( C_{i-1}+\ldots+C_2+C_1+1 \geq k/2*(\Sigma(1+2+4+\ldots+2^{j-1})) \geq k*2^j /2 \)
    • \( \geq k/2*C_i \)
  - Setting \( k = \frac{1}{\varepsilon} \) implies the required error guarantee!
Space Complexity

- Number of buckets $m$:
  - $m \leq \lceil \# \text{ of buckets of size } j \rceil \times \lceil \# \text{ of different bucket sizes} \rceil$
  - $\leq (k/2 + 1) \times ((\log(2N/k)+1) = O(k \times \log(N))$

- Each bucket requires $O(\log N)$ bits.

- Total memory:
  $O(k \log^2 N) = O(1/\varepsilon \times \log^2 N)$ bits

- Invariant (with $k = 1/\varepsilon$) maintains error guarantee!
EH Maintenance Algorithm

Data structures:
- For each bucket: timestamp of most recent 1, size = #1's in bucket
- LAST: size of the last bucket
- TOTAL: Total size of the buckets

New element arrives at time t
- If last bucket expired, update LAST and TOTAL
- If (element == 1)
  - Create new bucket with size 1; update TOTAL
- Merge buckets if there are more than k/2+2 buckets of the same size
- Update LAST if changed

Anytime estimate: TOTAL - (LAST/2)
Example Run

- If last bucket expired, update LAST and TOTAL
- If (element == 1)
  Create new bucket with size 1; update TOTAL
- Merge two oldest buckets if there are more than k/2+2 buckets of the same size
- Update LAST if changed

Example (k=2):

32,16,8,8,4,4,2,1,1
32,16,8,8,4,4,2,2,1
32,16,8,8,4,4,2,2,1,1
32,16,16,8,4,2,1
Lower Bound

• Argument: Count number of different arrangements that the algorithm needs to distinguish
  - $\log(N/B)$ blocks of sizes $B, 2B, 4B, ..., 2^iB$ from right to left.
  - Block $i$ is subdivided into $B$ blocks of size $2^i$ each.
  - For each block (independently) choose $k/4$ sub-blocks and fill them with 1.

• Within each block: ($B$ choose $k/4$) ways to place the 1s

• ($B$ choose $k/4$)$^{\log(N/B)}$ distinct arrangements
Lower Bound (continued)

- Example:

```
+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+
```

- Show: An algorithm has to distinguish between any such two arrangements
Assume we do not distinguish two arrangements:
- Differ at block d, sub-block b

Consider time when b expires
- We have c full sub-blocks in A1, and c+1 full sub-blocks in A2 [note: c+1≤k/4]
- A1: \(c2^d + \text{sum1} \) to \(d-1\) \(k/4 \times (1+2+4+..+2^{d-1})\)
  = \(c2^d + k/2(2^d-1)\)
- A2: \((c+1)2^d + k/4 \times (2^d-1)\)
- Absolute error: \(2^{d-1}\)
- Relative error for A2:
  \(2^{d-1}/[(c+1)2^d + k/4 \times (2^d-1)]\) \(\geq 1/k = \varepsilon\)
**Lower Bound (continued)**

**Calculation:**

- **A1:** \( c2^d + \text{sum1 to } d-1 \frac{k}{4^*}(1+2+4+..+2^{d-1}) \)
  \[= c2^d + k \frac{2^d-1}{2} \]

- **A2:** \( (c+1)2^d + k \frac{4^*}{2^d-1} \)

- **Absolute error:** \( 2^{d-1} \)

- **Relative error:**
  \[2^{d-1}/[(c+1)2^d + k/4^* \cdot 2^d]\]
  \[= 2^{d-1}/[2^d \cdot k/4^*] = 1/k = \varepsilon\]
The Power of EHs

- Counter for N items = $O(\log N)$ space
- EH = $\varepsilon$–approximate counter over sliding window of N items that requires $O(1/\varepsilon \times \log^2 N)$ space
  - $O(1/\varepsilon \log N)$ penalty for (approx) sliding-window counting

- Can plug EH-counters to counter-based streaming methods ⇒ work in sliding-window model!!
  - Examples: histograms, CM-sketches, ...
- Complication: counting is now $\varepsilon$–approximate
  - Account for that in analysis
Data-Stream Algorithmics Model

Continuous Data Streams

Stream Synopses (in memory)

Approximate Answer with Error Guarantees “Within 2% of exact answer with high probability”

- Approximate answers- e.g. trend analysis, anomaly detection
- Requirements for stream synopses
  - Single Pass: Each record is examined at most once
  - Small Space: Log or polylog in data stream size
  - Small-time: Low per-record processing time (maintain synopses)
  - Also: delete-proof, composable, ...
Distributed Streams Model

- Large-scale querying/monitoring: *Inherently distributed!*
  - Streams physically distributed across remote sites
    E.g., stream of UDP packets through subset of edge routers
- Challenge is “holistic” querying/monitoring
  - Queries over the union of distributed streams $Q(S_1 \cup S_2 \cup \ldots)$
  - Streaming data is spread throughout the network
Distributed Streams Model

- Need timely, accurate, and efficient query answers
- Additional complexity over centralized data streaming!
- Need space/time- and communication-efficient solutions
  - Minimize network overhead
  - Maximize network lifetime (e.g., sensor battery life)
  - Cannot afford to “centralize” all streaming data
Conclusions

- Querying and finding patterns in massive streams is a real problem with many real-world applications
- Fundamentally rethink data-management issues under stringent constraints
  - Single-pass algorithms with limited memory resources
- A lot of progress in the last few years
  - Algorithms, system models & architectures
    - GigaScope (AT&T), Aurora (Brandeis/Brown/MIT), Niagara (Wisconsin), STREAM (Stanford), Telegraph (Berkeley)
- Commercial acceptance still lagging, but will most probably grow in coming years
  - Specialized systems (e.g., fraud detection, network monitoring), but still far from “DSMSs”