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 $= 0$ (i.e., increment-only)
  - Typically, $c[j] \geq 1$, 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!

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 $(c, \delta)$-approximations
  - Estimate is within a relative error of $c$ with probability $\geq 1-\delta$
- Use Tail Inequality 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 $(f(i)), i=1...N$ seen as a stream of $i$-values.
- **Data stream:** $3, 1, 2, 4, 2, 3, 5, ...$
- **Basic construct:** Randomized Linear Projection of $f()$ onto inner/dot product of T-vector
  
  \[
  \langle f, \hat{c} \rangle := \sum_i f(i) \hat{c}_i,
  \]
  
  where $\hat{c}_i$ = vector of random values from an appropriate distribution.

  - Simple to compute over the stream: Add $\hat{c}_i$ whenever the $i$-th value is seen.
  - Data stream: $3, 1, 2, 4, 2, 3, 5, ...$
  - Generate $\hat{c}_i$ in small ($\log$N) space using pseudo-random generators.
  - Tunable probabilistic guarantee on approximation error.
  - Delete-Proof: Just subtract $\hat{c}_i$ 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 $x \in [0, N-1]$ uniformly across $[0, 2L-1]$, 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 $\text{Hash Sketch}$: BITMAP array of $L$ bits, initialized to 0.

  - For each incoming value $x$, set $\text{BITMAP}[\text{lsb}(h(x))] = 1$.

### Generalization: Distinct Values Queries

- **SELECT COUNT(DISTINCT target-attr)**
  
  **Template**

- **FROM** relation

  **WHERE** predicate

  - **SELECT COUNT(DISTINCT e_custkey)**
  
  **TPC-H example**

  - **FROM** orders
  
  **WHERE** e_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 [GGRO03]

- **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 $\setminus (R1 \cap R2)$ $\setminus R3$.

- **2-Level Hash-Sketch (2LHS) stream synopsis:** Generalizes FM sketch.

  - **First level:** $O(\log N)$ 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.

- **Template**

  - **SELECT COUNT(DISTINCT e_custkey)**
  
  **WHERE** e_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.

### Processing Set Expressions over Update Streams [GGRO03]

- **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 $\setminus (R1 \cap R2)$ $\setminus R3$.

- **2-Level Hash-Sketch (2LHS) stream synopsis:** Generalizes FM sketch.

  - **First level:** $O(\log N)$ 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.

- **Insert(17)**

  - $\text{lsb}(h(17))$**
  
  - **Insert(17)**
  
  - **Insert(17)**

  - **-1 for deletes**

  - **Template**

  - **SELECT COUNT(DISTINCT e_custkey)**
  
  **WHERE** e_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.
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?

---

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 \( x_j \), 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 \([1, w]\).

\[ W \quad \text{d} \]

\[ h_i(j) \]

\[ A[j] \]

\[ h_1(j) \]

\[ h_d(j) \]

- Each entry in vector \( A \) is mapped to one bucket per row.
- Merge two sketches by entry-wise summation.
- Estimate \( A[j] \) by taking \( \min_i \text{sketch}[h_i(j)] \).

[Comodo, Muthukrishnan ’05]

---

CM Sketch Summary

- CM sketch guarantees approximation error on point queries less than \( \sqrt{|A|} \), in size \( O(\sqrt{N \log 1/\delta}) \).
  - Probability of more error is less than \( 1-\delta \).
  - Similar guarantees for range queries, quantiles, join size estimation.

- 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 \( \min_i \) estimate.

- Food for thought: How do the CM sketch guarantees compare to AMS??

---

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.

---

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**

- Function $f$: $f(x,y,z)$
  - Input at time $t$: $f(x_1, y_1, z_1, t_1)$
  - Input at time $t=1$: $f(x_1, y_1, z_1, t+1)$
  - Input at time $t=1, 2$: $f(x_1, y_1, z_1, t+1)$
  - Input at time $t=1, 2, 3$: $f(x_1, y_1, z_1, t+1)$
- Full history: $f$ = identity
- Partial history: Decay
  - Exponential decay: $f(x, y, z) = 2 \cdot \text{now} \cdot x$
    - Input at time $t$: $2 \cdot \text{now} \cdot x_1, 2 \cdot \text{now} \cdot x_2, \ldots$, $x_i, X_i$
    - Input at time $t=1$: $2 \cdot \text{now} \cdot x_1, 2 \cdot \text{now} \cdot x_2, \ldots, 1/4 \cdot x_i, X_i$
    - Sliding window (special case of decay):
      - $f(x, y, z) = X$ if $\text{now} < t$
      - $f(x, y, z) = 0$, otherwise
      - Input at time $t$: $x_1, x_2, x_3, \ldots x_t$
      - Input at time $t=1$: $x_1, x_2, x_3, \ldots x_t$

**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 $N$ > $N^2 \log R / N$
  - So if $R$ is 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: $O(N)$ bits
  - Algorithm BasicCounting:
    - $1 + \epsilon$ approximation (relative error)
    - Space: $O(1/\epsilon \log N)$ bits
    - Time: $O(\log N)$ worst case, $O(1)$ amortized per record
  - Lower Bound:
    - Space: $\Omega(1/\epsilon \log N)$ bits

**Approach: Temporal Histograms**

Example: ... 0110101001111110110 0101 ...

Equi-width histogram:
... 0110 1110 1010 1011 1111 1111 1010 0101 ...

- Issues:
  - Error is in the last (leftmost) bucket.
  - Bucket counts (left to right): $C_{n+1}, \ldots, C_2, C_1$
  - Absolute error <= $C_n / 2$
  - Answer <= $C_{n+1} + \ldots + C_2 + C_1$
  - Relative error <= $C_n / 2(C_{n+1} + \ldots + C_2 + C_1)$
  - Maintain $C_n / 2(C_{n+1} + \ldots + C_2 + C_1) <= \epsilon (1/\epsilon)$

**Naive: Equi-Width Histograms**

- Goal: Maintain $C_n / 2 < \epsilon (C_{n+1} + \ldots + C_2 + C_1)$
- Problem case:
  ... 0110 1010 0111 1111 0110 1111 0000 0000 0000 ...

- Note:
  - Every bucket will be the last bucket sometime!
  - New records may be all zeros → For every bucket $i$, require $C_i / 2 <= \epsilon (C_{i+1} + \ldots + C_2 + C_1)$

**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 $k/2$ and at most $k/2+1$ buckets of that size
  - Example: $k=4$: (8, 4, 4, 4, 2, 2, 2, 1, 1)

- Invariant implies:
  - Assume $C_i = 2^i$, then
    - $C_i + \ldots + C_2 + C_1$ = $k/2(2^1 + 2^2 + \ldots + 2^i)$ = $k2^i / 2$
  - Setting $k = 1/\epsilon$ implies the required error guarantee!
**Space Complexity**

- Number of buckets $m$:
  - $m \leq [\# \text{ of buckets of size } j]*[\# \text{ of different bucket sizes}]
  - $(k/2+1)*((\log(2N/k)-1) = O(k^* \log(N))$
- Each bucket requires $O(\log N)$ bits.
- Total memory:
  - $O(k \log^2 N) = O(1/e \times \log^3 N)$ bits
- Invariant (with $k = 1/e$) maintains error guaranteed

**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: $\text{TOTAL} - (\text{LAST}/2)$

**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 $B,2B,4B,...,2^B$ 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: $c\leq k/4$]
- $A1: c2^d=\text{sum of d-1,} k/4^i[1+2+4+...+2^{d-1}]$
- $A2: (\leq c+1)\frac{k}{4}^d2^{d-i}$
- Absolute error: $2^d$
- Relative error for A2: $2^d/((\leq c+1)\frac{k}{4}^d2^{d-i}) = 1/k + \varepsilon$
Lower Bound (continued)

Calculation:
- A1: \(c2k^{h} \sum_{k=1}^{d} \frac{k}{4^k} / ((1+2+4+...+2^d-1))\) = \(c2^k k / (2^k-1)\)
- A2: \((c+1)2^{k/4} / (2^k-1)\)
- Absolute error: \(2^{k/2} (1/4^k) \leq \varepsilon\)

The Power of EHs

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

Data-Stream Algorithmics Model

- Approximate answers: e.g., trend analysis, anomaly detection
- Requirements for stream synopsis
  - 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 synopsis)
  - Also: deletions proof, composable...

Distributed Streams Model

- Large-scale querying/monitoring: Inherently distributed
- Streams physically distributed across remote sites
  - E.g., streams of UDP packets through subset of edge routers
- Challenge is "holistic" querying/monitoring
  - Queries over union of distributed streams \(Q(S_1, S_2, ..., S_k)\)
- Streaming data is spread throughout the network

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 (Brandani/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 "DSMSSs"