One-Pass Streaming Algorithms

Complaints and Grievances about theory in practice
Disclaimer

- Experiences with Gigascope.
- A practitioner’s perspective.
- Will be using my own implementations, rather than Gigascope.
Outline

- What is a data stream?
- Is sampling good enough?
- Distinct Value Estimation
- Frequency Estimation
- Heavy Hitters
Setting

- Continuously generated data.
- Volume of data so large that:
  - We cannot store it.
  - We barely get a chance to look at all of it.
- Good example: **Network Traffic Analysis**
  - Millions of packets per second.
  - Hundreds of concurrent queries.
  - How much main memory per query?
Formally

- **Data**: Domain of items $D = \{1, \ldots, N\}$, 
  ... where $N$ is very large!
  - IPv4 address space is $2^{32}$.
- **Stream**: A multi-set $S = \{ i_1, i_2, \ldots, i_M \}$, $i_k \in D$:
  - Keeps expanding.
  - i’s arrive in any order.
  - i’s are inserted and deleted.
  - i’s can even arrive as incremental updates.
- **Essential quantities**: $N$ and $M$. 
Example

- Number of distinct items
  - Distinct destination IP addresses

<table>
<thead>
<tr>
<th>Packet #</th>
<th>Source IP</th>
<th>Destination IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>147.102.1.1</td>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>2:</td>
<td>162.102.1.20</td>
<td>147.102.10.5</td>
</tr>
<tr>
<td>3:</td>
<td>154.12.2.34</td>
<td><a href="http://www.niss.org">www.niss.org</a></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k:</td>
<td>147.102.1.2</td>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
</tbody>
</table>

- Simple solution: Maintain a hash table
- How big will it get?
One-Pass Algorithm

- Design an algorithm that will:
  - Examine arriving items once, and discard.
  - Update internal state fast (O(1) to poly log N).
  - Provide answers fast.
  - Provide guarantees on the answers (\(\epsilon, \delta\)).
  - Use small space (poly log N).
  - ...

- We call the associated structure:
  - A sketch, synopsis, summary
Example (cont.)

- Distinct number of items:
  - Use a memory resident hash table:
    - Examines each item only once.
    - Fairly fast updates
    - Very fast querying
    - Provides exact answer
    - Can get arbitrarily large

- Can we get good, approximate solutions instead?
Outline

- What is a data stream?
- Is sampling good enough?
- Distinct Value Estimation
- Frequency Estimation
- Heavy Hitters
Randomness is key

Maybe we can use sampling:

- Very bad idea (sorry sampling fans!)
- Large errors are unavoidable for estimates derived only from random samples.
- Even worse, negative results have been proved for “any (possibly randomized) strategy that selects a sequence of x values to examine from the input” [CCMN00]
Outline

- Is sampling good enough?
- Distinct Value Estimation
- Frequency Estimation
- Heavy Hitters
We need to be more clever

- Design algorithms that **examine all inputs**
- The FM sketch [FM85]:
  - Assign items *deterministically* to a random variable from a geometric distribution:
    \[
    \Pr[ h(i) = k ] = 1/2^k.
    \]
  - Maintain array A of log N bits, initialized to 0.
  - Insert i: set \( A[ h(i) ] = 1 \).
  - Let \( R = \{ \min j \mid A[j] = 0 \} \).
    \[
    \ldots0010001001101111111
    \]
  - Then, distinct items \( D' \approx 1.29 \cdot 2^R \).
  - This is an unbiased estimate! Long proof…
How clever do we need to be?

- A simpler algorithm.
- The KMV sketch [BHRSG06]:
  - Assign items *deterministically* to uniform random numbers in [0, 1].
  - \( d \) distinct items will cut the unit interval in \( d \) equi-length intervals, of size \( \sim 1/d \).
  - Suppose we maintain the \( k \)-th minimum item:
    - \( h(k) \approx k \cdot 1/d \), hence \( D' \approx k / h(k) \).
  - This estimate is biased upwards, but …
  - \( D' \approx (k – 1) / h(k) \) isn’t! Easy proof…
Let's compare

- **Guarantees**: \( \Pr[|D - D'| < \epsilon D] > 1 - \delta. \)
- **Space** \((\epsilon, \delta\) guarantees):
  - FM: \(1/\epsilon^2 \log(1/\delta) \log N\) bits
  - KMV: the same
- **Update time**:
  - FM: \(1/\epsilon^2 \log(1/\delta)\)
  - KMV: \(\log(1/\epsilon^2) \log(1/\delta)\)

KMV is much faster! But how well does it work?
But first … a practical issue

- How do we define this “perfect” mapping h?
  - Should be pair-wise independent.
  - Collision free.
  - Should be stored in log space.

- This doesn’t exist! Instead:
  - We can use **Pseudo Random Generators**.
  - We can use a **Universal Hash Function**.
  - “Look” random, can be stored in log space.

- We are deviating from theory!
Let’s run some experiments

- **Data:**
  - AT&T backbone traffic

- **Query:**
  - Distinct destination IPs observed every 10000 packets.

- **Measures:**
  - Sketch size (number of bytes)
  - Insertion cost (updates per second)
Sketch size

Average Relative Error vs Sketch Size

- **FM**
- **KMV**

Sketch size (bytes)

Average relative error
Insertion cost

Updates Per Second vs Sketch Size

Sketch size (bytes)

Updates per second

FM

KMV
Speeding up FM

- Instead of updating all $1/\varepsilon^2$ bit vectors:
  - Partition input into $m$ bins.
  - Average over all bins at the end.

- Authors call this approach Stochastic Averaging.
Sketch size

Average Relative Error vs Sketch Size

FM
FM-SA
KMV
RS
Insertion cost

Updates Per Second vs Sketch Size

- FM
- FM-SA
- KMV
- RS
Uniformly distributed data

Averate Relative Error vs Sketch Size

Average relative error

Sketch size (bytes)
Zipf data

Average Relative Error vs Skew (800 bytes)

- FM
- FM-SA
- KMV

Average relative error

Skew
Any conclusion?

- The size of the window matters:
  - The smaller the quantity the harder to estimate.
  - FM-SA: Increasing the number of bit vectors, assigns fewer and fewer items to each bin.
  - Better off using exact solution in some cases.
- The quality of the hash function matters.
- FM-SA best overall … if we can tune the size.
- What about deletions?
Outline

- Distinct Value Estimation
- Frequency Estimation
- Heavy Hitters
The problem

- **Problem:**
  - For each \( i \in D \), maintain the frequency \( f(i) \), of \( i \in S \).

- **Application:**
  - How much traffic does a user generate?
    - Estimate the number of packets transmitted by each source IP.
A Counter-Example!

Puzzle:
1. Assume a skewed distribution. What is the frequency of ... 80% of the items?
2. Assume a uniform distribution. What is the frequency of ... 99% of the items?

Conclusion:
- Frequency counting is not very useful!
Not convinced yet?

The Fast-AMS sketch [AMS96,CG05]:
- Maintain an $m \times n$ matrix $M$ of counters, initialized to zero.
- Choose $m$ 2-wise independent hash functions (image $[1, n]$).
- Choose $m$ 4-wise independent hash functions (image $\{-1, +1\}$).
- Insert $i$:
  - For each $k \in [1, m]: M[ k, h^2_k(i) ] += h^4_k(i)$.
- Query $i$:
  - The median of the $m$ counters corresponding to $i$. 
Theoretical bounds

This algorithm gives $\epsilon$, $\delta$ guarantees:
- Space: $1/\epsilon \log(1/\delta) \log N$

What’s the catch?
- Guarantees: $\Pr[|f_i - f_i'| < \epsilon M] > 1 - \delta$

Not very useful in practice!
Experiments with AT&T data

Averate Relative Error vs Top-k

Average relative error

Top-k
Outline

- Frequency Estimation
- Heavy Hitters
The problem

Problem:
- Given $\theta \in (0, 0.5]$, maintain all $i$ s.t. $f(i) \geq \theta M$.

Application:
- Who is generating most of the traffic?
  - Identify the source IPs with the largest payload.
- Heavy hitters make sense... in some cases!
  - What if the distribution is uniform?

Detect if the distribution is skewed first!
The solutions

- Heavy hitters is an easier problem.

- Deterministic algorithms:
  - Misra-Gries [MG82].
  - Lossy counting [MM02].
  - Quantile Digest [SBAS04].

- Randomized algorithms:
  - Fast AMS + heap.
  - Hierarchical Fast AMS (dyadic ranges).
Misra-Gries

- Maintain k pairs \((i, f_i)\) as a hash table \(H\):
  - Insert \(i\):
    - If \(i \in H\): \(f_i += 1\),
    - else insert \((i, 1)\).
  - If \(|H| > k\), for all \(i\): \(f_i -= 1\).
  - If \(f_i = 0\), remove \(i\) from \(H\).

- Problem:
  - The algorithm is supposed to be deterministic.
  - Hash table implies randomization!
Misra-Gries Cost

- **Space:**
  - $1/\theta$.

- **Update:**
  - Expected $O(1)$:
    - Play tricks to get rid of the hash table.
    - Increase space to use pointers and doubly linked lists.
Lossy Counting

- Maintain list L of \((i, f_i, \delta)\) items:
  - Set \(B = 1\).
  - Insert \(i\):
    - If \(i\) in L, \(f_i + 1\),
    - else add \((i, 1, B)\).
  - On every \(1/\theta\) arrivals:
    - \(B + 1\),
    - Evict all \(i\) s.t. \(f_i + \delta \leq B\).
Lossy Counting Cost

- **Space:**
  - $\frac{1}{\theta} \log \theta N$

- **Update:**
  - Expected $O(1)$
Quantile Digest

- A hierarchical algorithm for estimating quantiles.
- Based on binary tree.
- Can be used to detect heavy hitters.
  - Leaf level of tree are all the items with large frequencies!

- Estimating quantiles is a generalization of heavy hitters.
Quantile Digest Cost

- **Space:**
  - $\frac{1}{\theta} \log N$

- **Update:**
  - $\log \log N$
Experiments

- Uniform distribution: No Heavy Hitters!
- Experiments with AT&T data:
  - **Recall**: Percent of true heavy hitters in the result.
  - **Precision**: Percent of true heavy hitters over all items returned.
  - **Update cost**.
  - **Size**.
- All algorithms consistently had 100% recall.
Conclusion

- Many interesting data stream applications.
- Setting necessitates use of approximate, small space algorithms.
- Some algorithms give theoretical guarantees, but have problems in practice.
- Some algorithms behave very well.
- There is always room for improvement.
Outline

End

- Heavy Hitters
References

- [CCMN00]: Towards estimation error guarantees for distinct values.
- [FM85]: Counting Algorithms for Data Base Applications.
- [BHRSG07]: On synopses for distinct-value estimation under multiset operations.
- [AMS96]: The Space Complexity of Approximating the Frequency Moments.
- [CG05]: Sketching streams through the net: Distributed approximate query tracking.
- [MG82]: Finding repeated elements.
- [MM00]: Approximate frequency counts over data streams.
- [SBAS04]: Medians and beyond: approximate aggregation techniques for sensor networks.