One-Pass Streaming Algorithms

Complaints and Grievances
Theory and Practice
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

Packet #	Source IP		Destination IP
1:	147.102.1.1	→	www.google.com
2:	162.102.1.20	→	147.102.10.5
3:	154.12.2.34	→	www.niss.org
		...	
k:	147.102.1.2	→	www.google.com

- 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 (ε, δ).
  - 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 | A[j] = 0}.
    …0010001001101111111
  - 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 ~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…

Lets compare

- **Guarantees**: $\Pr[|D - D'| < \varepsilon D] > 1 - \delta$.
- **Space** (ε, δ guarantees):
  - FM: $1/\varepsilon^2 \log(1/\delta) \log N$ bits
  - KMV: the same
- **Update time**:
  - FM: $1/\varepsilon^2 \log(1/\delta)$
  - KMV: $\log(1/\varepsilon^2) \log(1/\delta)$
- KMV is much faster! But how well does it work?

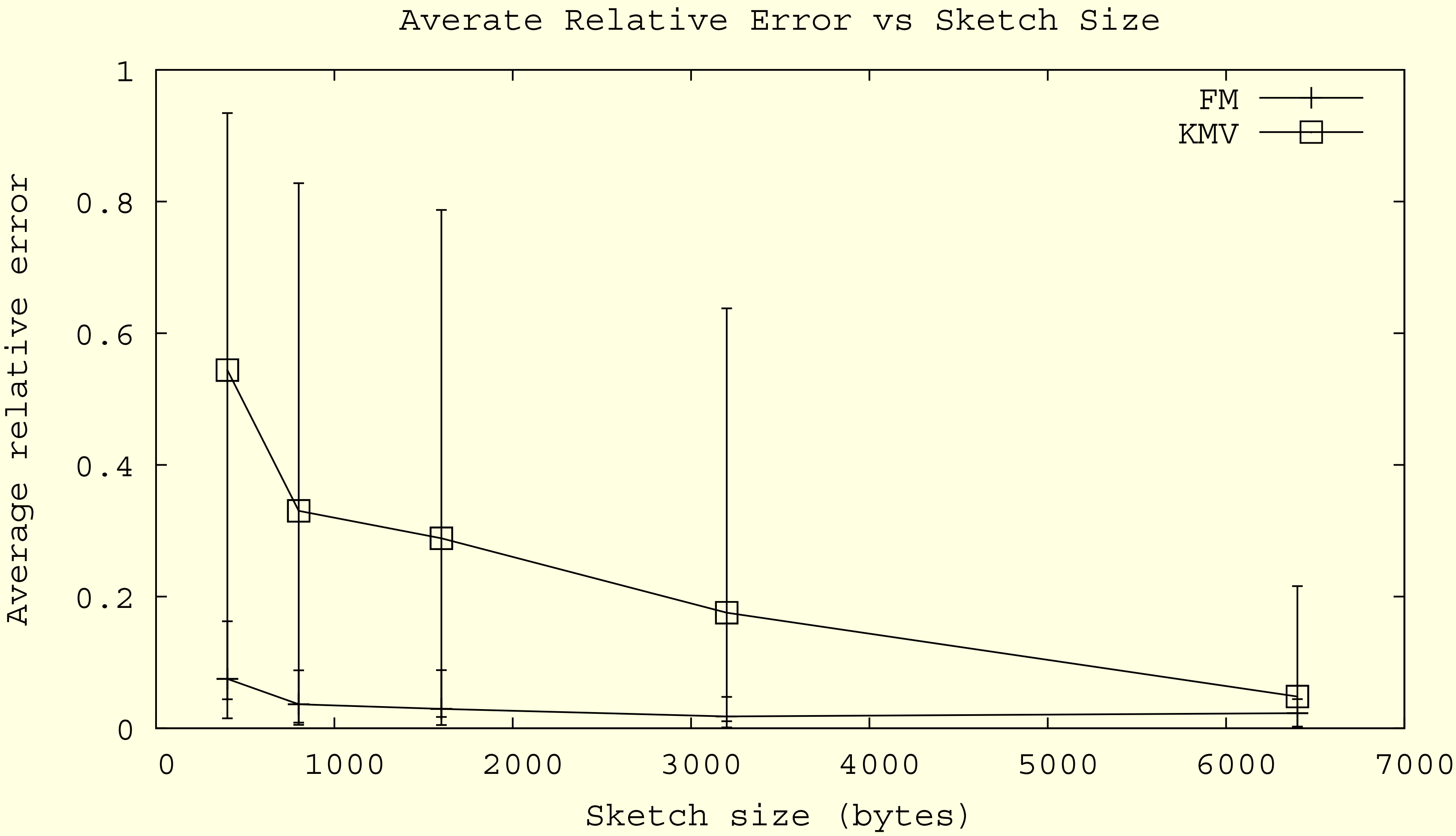
But first … a practical issue

- 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

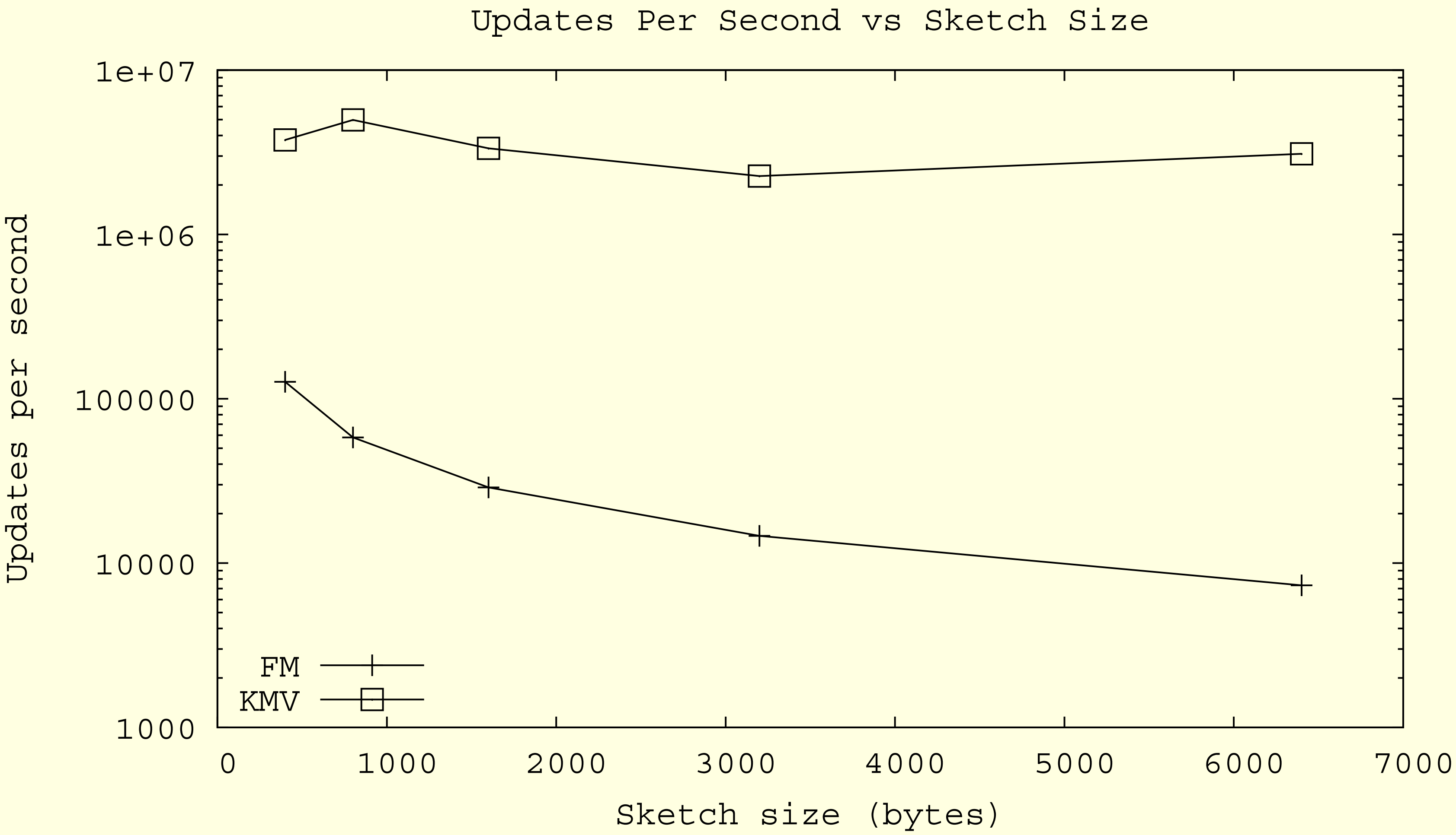
- **Data**:
  - AT&T backbone traffic
- **Query**:
  - Distinct destination IPs observed every 10000 packets.
- **Measures**:
  - Sketch size (number of bytes)
  - Insertion cost (updates per second)

Sketch size



Averate Relative Error vs Sketch Size
Average relative error
Sketch size (bytes)
FM
KMV
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000

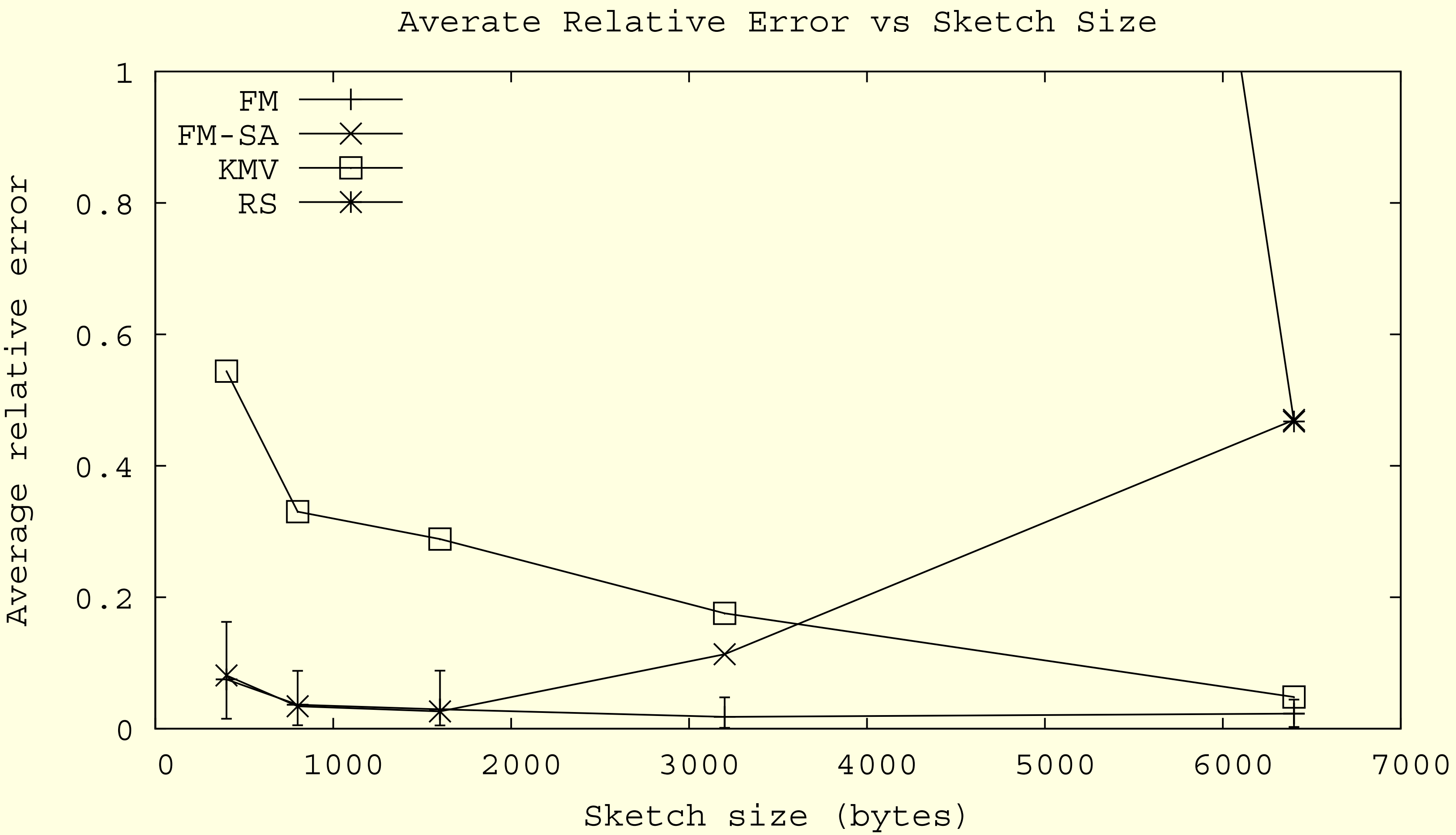
Insertion cost



Speeding up FM

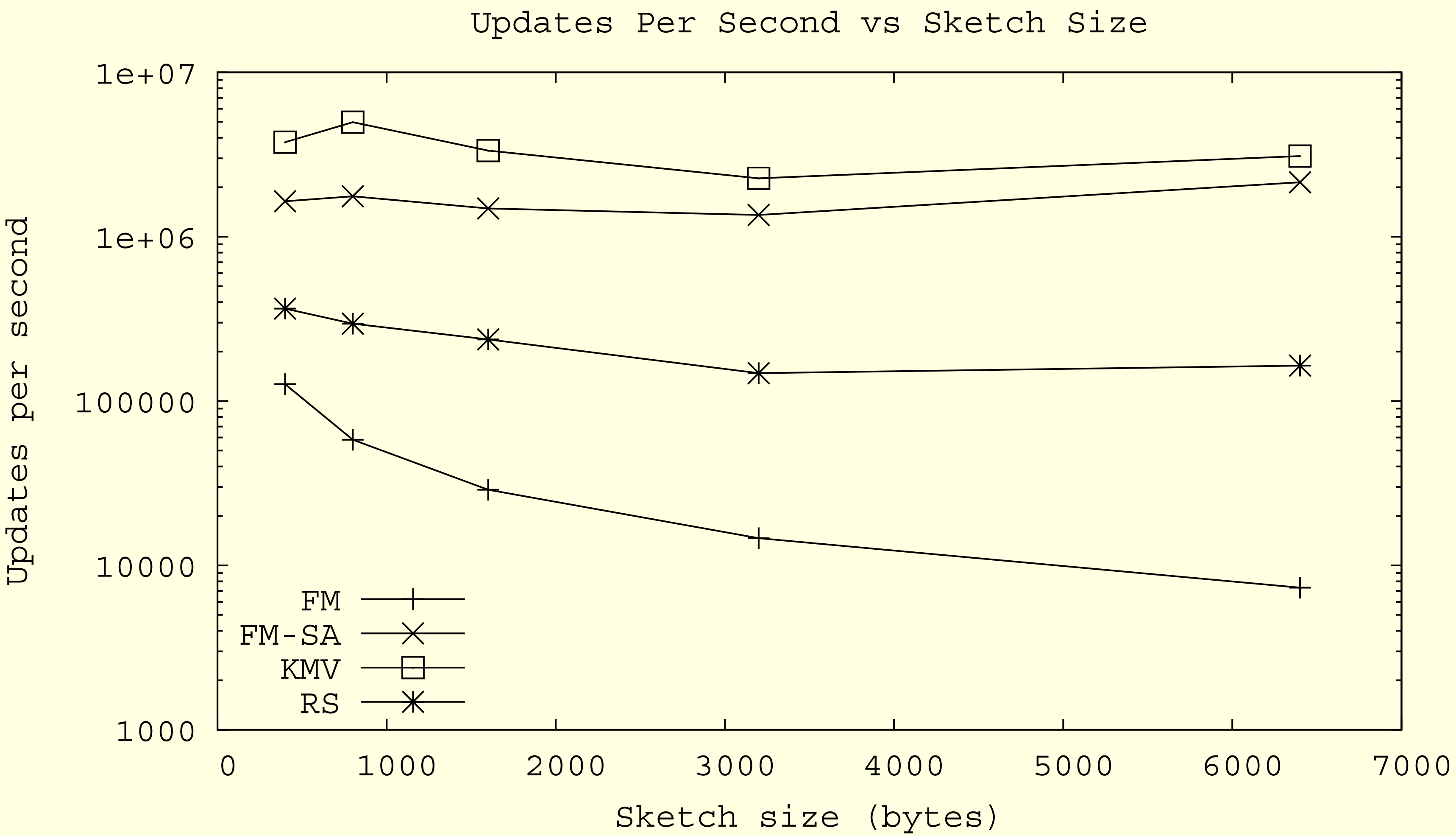
- Instead of updating all $1/\varepsilon^2$ bit vectors:
  - Partition input into m bins.
  - Average over all bins at the end.
- Authors call this approach Stochastic Averaging.

Sketch size

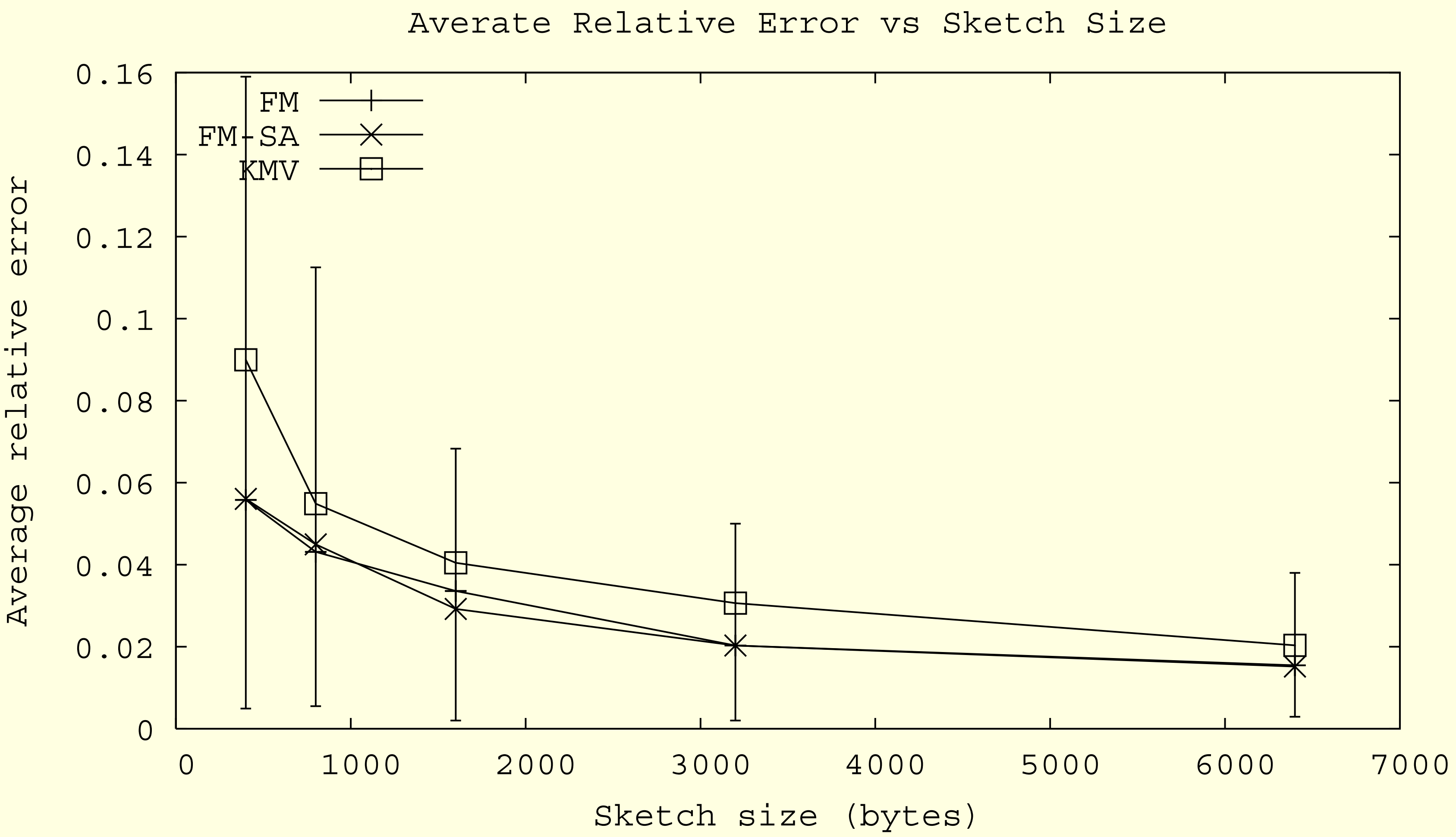


Averate Relative Error vs Sketch Size
FM
FM-SA
KMV
RS
Average relative error
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
Sketch size (bytes)

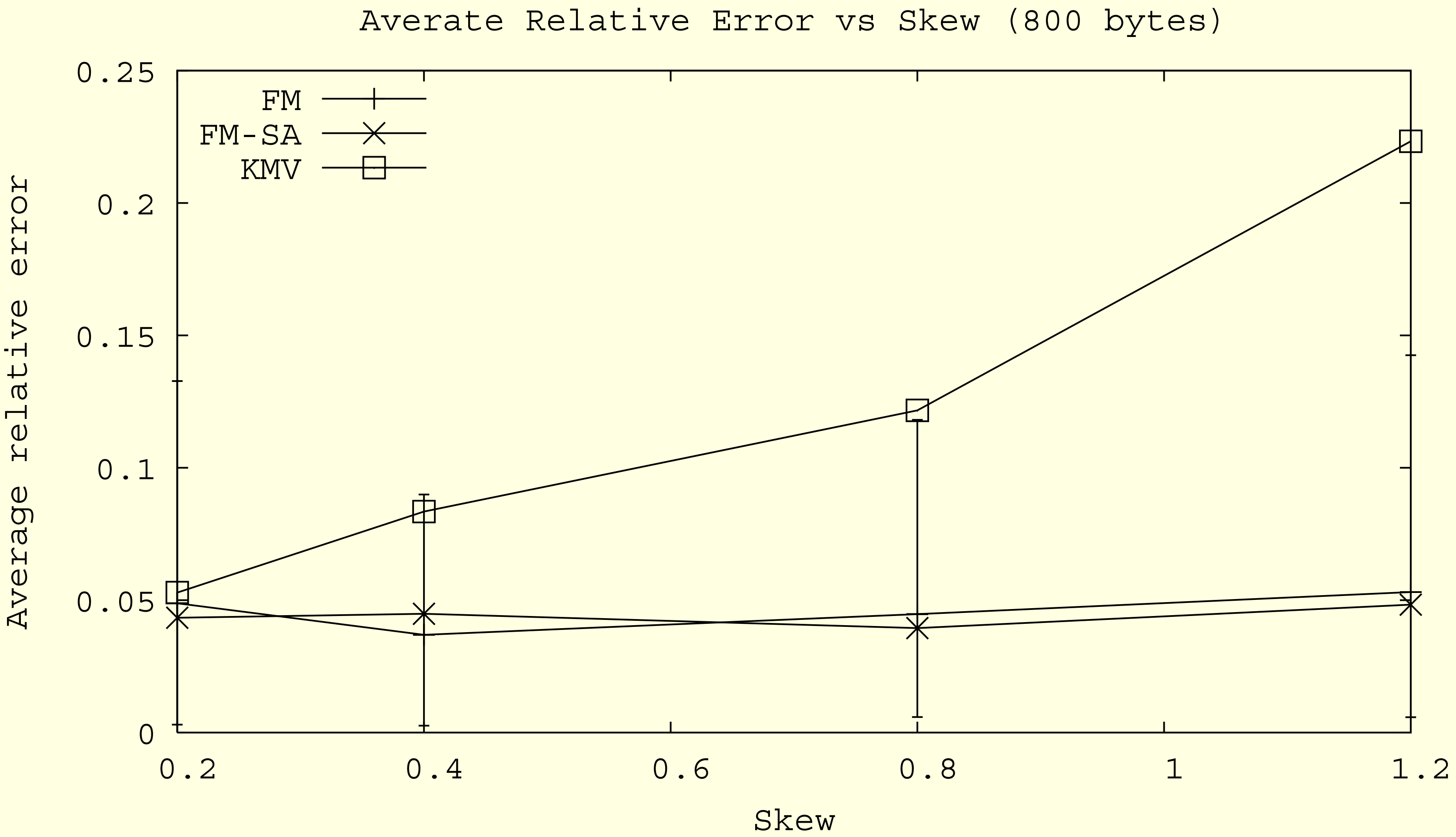
Insertion cost



Uniformly distributed data



Zipf data


Averate Relative Error vs Skew (800 bytes)
0.25
0.2
0.15
0.1
0.05
0
0.2
0.4
0.6
0.8
1
1.2
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!

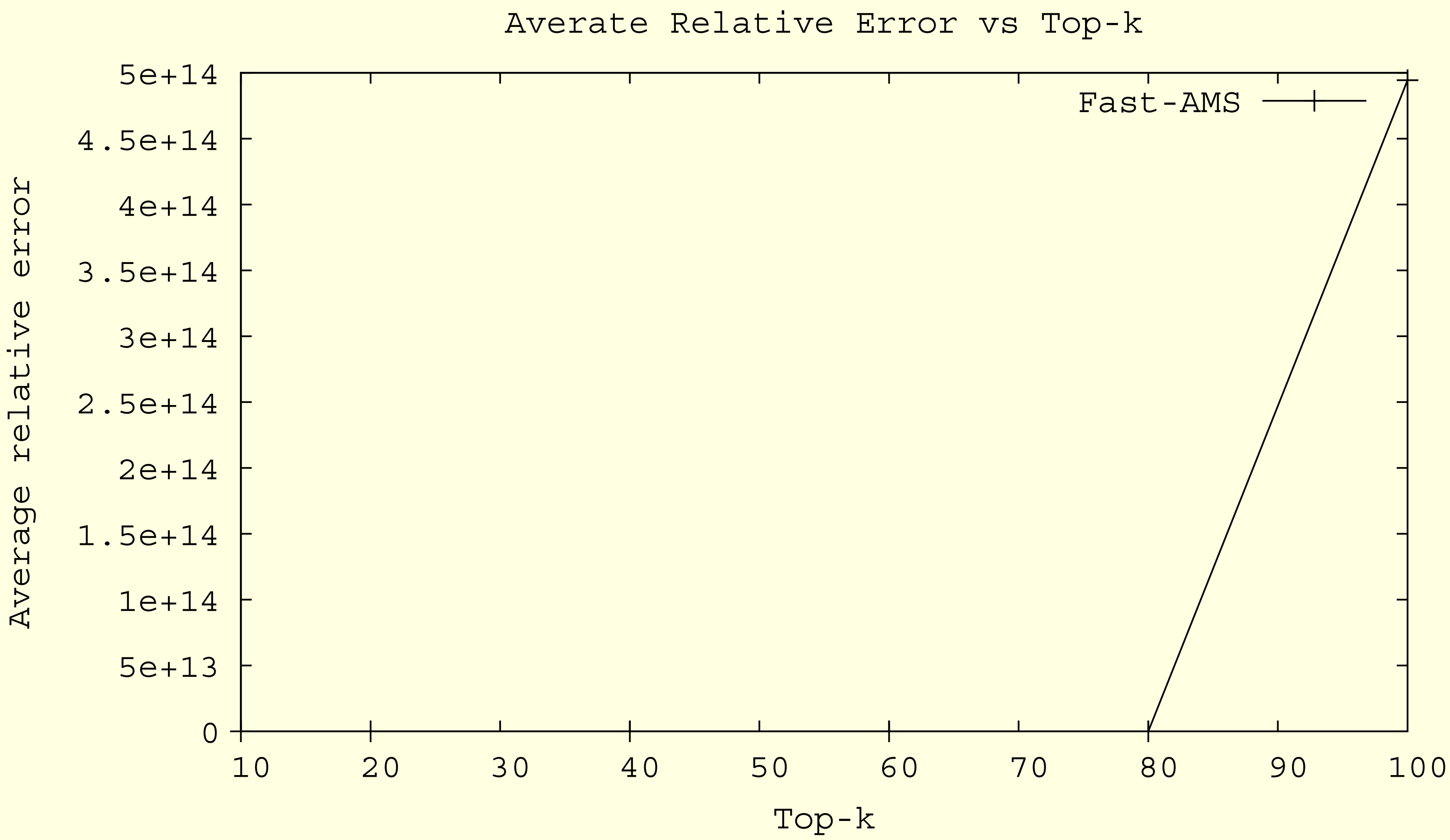
Not convinced yet?

- The Fast-AMS sketch [AMS96,CG05]:
  - Maintain an *m x 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 $k \in [1, m]$: $M[\, k, h^2_k(i)\,] \mathrel{+}= h^4_k(i)$.
  - Query i:
    - The median of the m counters corresponding to i.

Theoretical bounds

- This algorithm gives ε, δ guarantees:
  - Space: $1/\varepsilon \log(1/\delta) \log N$
- What's the catch?
  - Guarantees: $\Pr[|f_i - f_i'| < \varepsilon M] > 1 - \delta$
- Not very useful in practice!

Experiments with AT&T data



Outline

- Frequency Estimation
- Heavy Hitters

The problem

- **Problem:**
  - Given $\theta \in (0, 0.5]$, maintain all i s.t. f(i) >= θM.
- **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, δ) items:
  - Set B = 1.
  - Insert i:
    - If i in L, f_i += 1,
    - else add (i, 1, B).
  - On every 1/θ arrivals:
    - B += 1,
    - Evict all i s.t. $f_i + \delta$ <= B.

Lossy Counting Cost

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

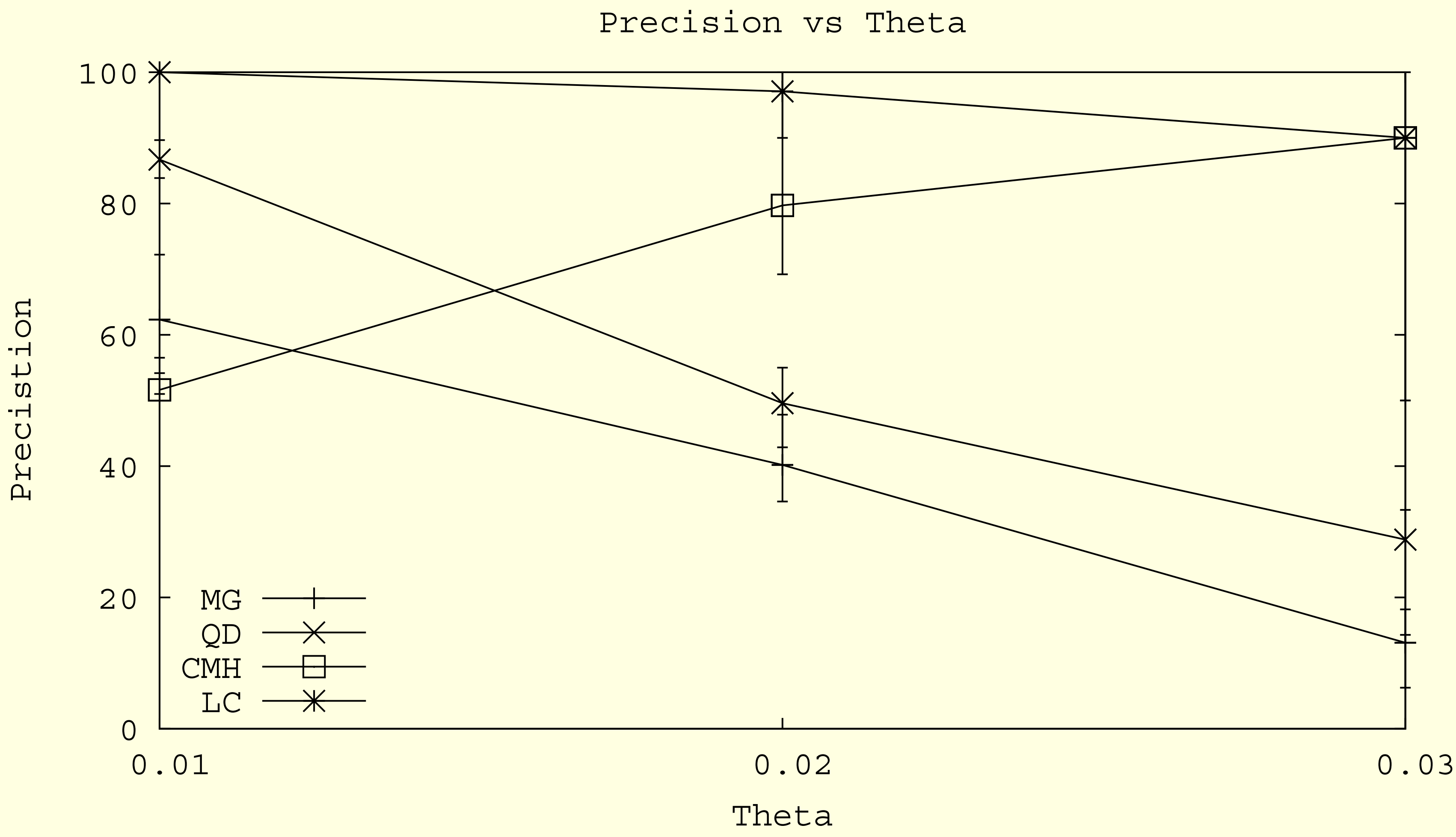
Quantile Digest Cost

- **Space**:
  - $1/\theta \log N$
- **Update**:
  - $\log \log N$

Experiments

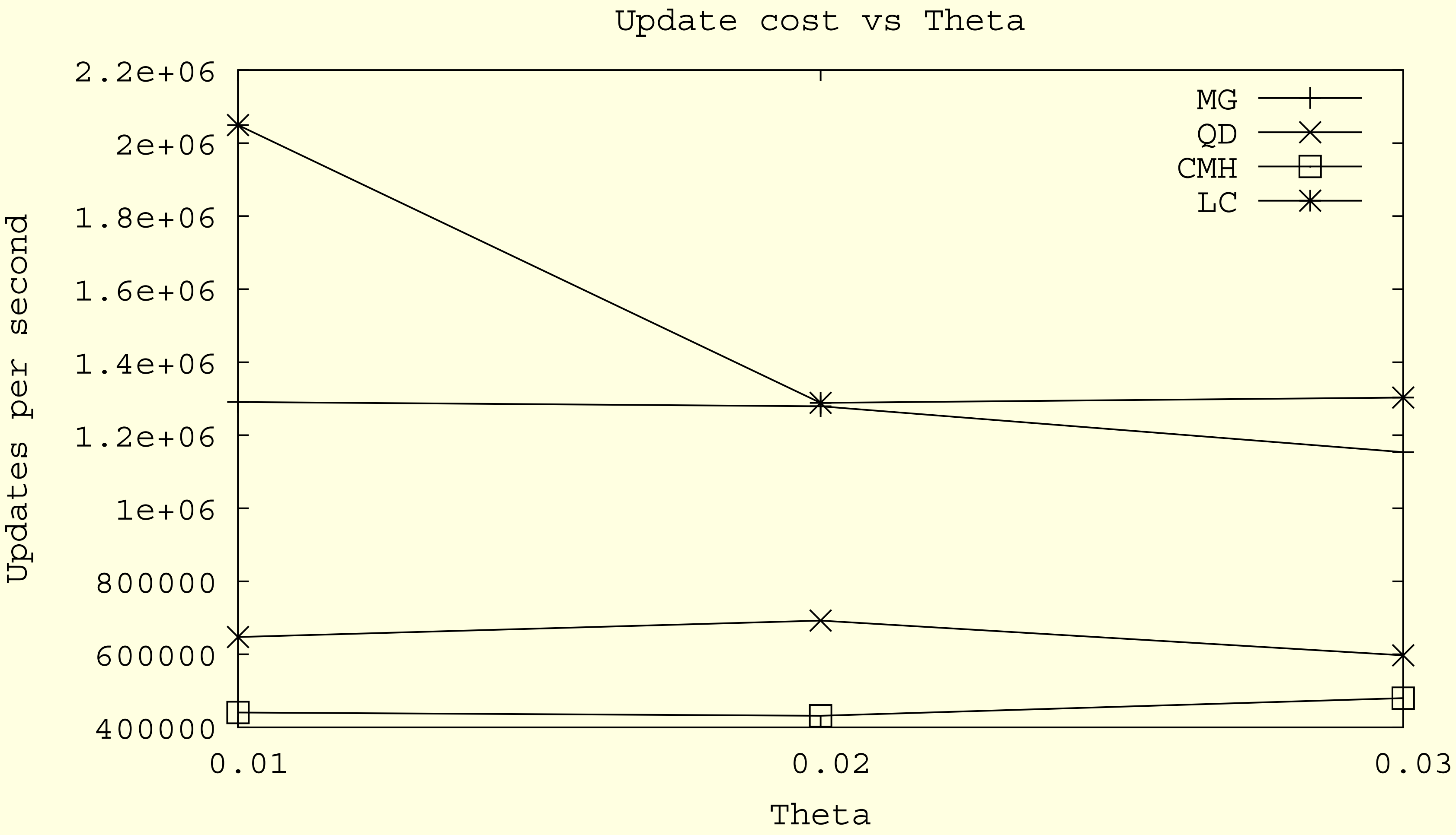
- Uniform distribution: No Heavy Hitters!
- Experiments with AT&T data:
  - **Recall**: Percent of true heavy hitters in the result.
  - **Precision**: Percent of true heavy hitters over all items returned.
  - **Update cost**.
  - **Size**.
- All algorithms consistently had 100% recall.

Precision

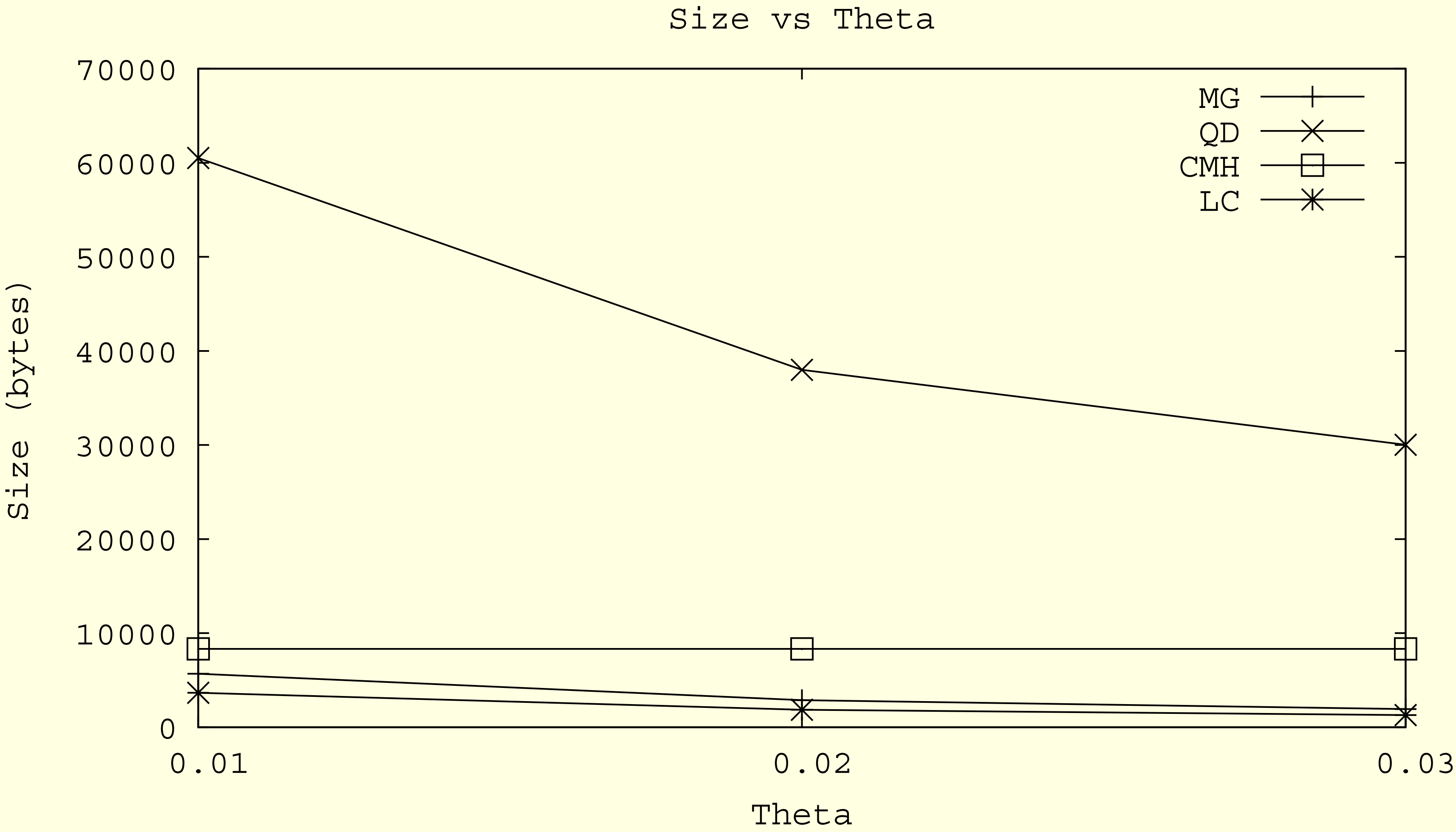


Precision vs Theta
100
80
60
40
20
0
0.01
0.02
0.03
Theta
Precistion
MG
QD
CMH
LC

Update cost


Update cost vs Theta
MG
QD
CMH
LC
2.2e+06
2e+06
1.8e+06
1.6e+06
1.4e+06
1.2e+06
1e+06
800000
600000
400000
Updates per second
0.01
0.02
0.03
Theta

Size



Size vs Theta
70000
60000
50000
40000
30000
20000
10000
0
0.01
0.02
0.03
Theta
Size (bytes)
MG
QD
CMH
LC

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

- **[S. Muthukrishnan 2003]: Data Streams: Algorithms and Applications.**
- [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.