Untangling the Braid: Finding Outliers in a Set of Streams

Luca Foschini
joint work with
Chiranjeeb Buragohain and Subhash Suri

University of California Santa Barbara

January 16, 2010
Outline

Problem Definition

Motivation

Previous Work

Approximated Weight Functions

Outline of Results

Lower Bounds

Experimental Results

Conclusion
Problem Definition

- A braid $\mathcal{B}$ is a set of $m$ streams $S_1, \ldots, S_m$ of real numbers
- for a given weight function $\lambda$ (e.g., average, median, max),
- find useful statistics (e.g., max) of the set $\{\lambda(S_1), \ldots, \lambda(S_m)\}$
Streams and Braids

Problem Definition

- A braid $\mathcal{B}$ is a set of $m$ streams $S_1, \ldots, S_m$ of real numbers
- for a given weight function $\lambda$ (e.g., average, median, max),
- find useful statistics (e.g., max) of the set $\{\lambda(S_1), \ldots, \lambda(S_m)\}$

Assumptions

- $v_{i,j}$ (the $j$-th value in the stream $S_i$) are real values
- $v_{i,j}$ from different streams are intermixed and seen only once
- $|\mathcal{B}| = m \gg 1$, $n_i = |S_i| \gg 1$
Streams and Braids

Problem Definition

▶ A braid $B$ is a set of $m$ streams $S_1, \ldots, S_m$ of real numbers
▶ for a given weight function $\lambda$ (e.g., average, median, max),
▶ find useful statistics (e.g., max) of the set $\{\lambda(S_1), \ldots, \lambda(S_m)\}$

Assumptions

▶ $v_{i,j}$ (the $j$-th value in the stream $S_i$) are real values
▶ $v_{i,j}$ from different streams are intermixed and seen only once
▶ $|B| = m \gg 1$, $n_i = |S_i| \gg 1$

Goal: one-pass algorithm
Example

Worst 2 Streams by Average ($\lambda^{avg}$)

\[ B = \begin{array}{cccccccccccc}
15 & 1 & 12 & 8 & 87 & 12 & 23 & 92 & 9 & 17 & 32 & 37
\end{array} \]
Example

Worst 2 Streams by Average ($\lambda^{avg}$)

\[
\mathcal{B} = \begin{pmatrix}
15 & 1 & 12 & 8 & 87 & 12 & 23 & 92 & 9 & 17 & 32 & 37
\end{pmatrix}
\]

\[
S_1 = \frac{15 + 12 + 23}{3} = 16.7
\]

\[
S_2 = \frac{1 + 12 + 9 + 32}{4} = 13.5
\]

\[
S_3 = \frac{8 + 92 + 17}{3} = 39
\]

\[
S_4 = \frac{87 + 37}{2} = 62
\]
Example

Worst 2 Streams by Average ($\lambda^{avg}$)

$B = \begin{bmatrix} 15 & 1 & 12 & 8 & 87 & 12 & 23 & 92 & 9 & 17 & 32 & 37 \end{bmatrix}$

$S_1 \rightarrow S_4 \quad \frac{87 + 37}{2} = 62$

$S_2 \quad \frac{8 + 92 + 17}{3} = 39$

$S_3 \quad \frac{1 + 12 + 9 + 32}{4} = 13.5$

$S_4 \quad \frac{15 + 12 + 23}{3} = 16.7$
Outline

Problem Definition

Motivation

Previous Work

Approximated Weight Functions

Outline of Results

Lower Bounds

Experimental Results

Conclusion
Motivation

Performance Monitoring in Large Shared Infrastructures

- Each user’s performance profile is a stream of numbers (latencies, response times...)
- the aggregate performance profile is a braid of the intermixed streams
- goal of the monitoring system: find the outliers
- example: Cloud computing, monitoring micro-burstiness/latency at the router level [KLSV09, LMP+08]
Motivation

Performance Monitoring in Large Shared Infrastructures

- Each user’s performance profile is a stream of numbers (latencies, response times...)
- the aggregate performance profile is a braid of the intermixed streams
- goal of the monitoring system: find the outliers
- example: Cloud computing, monitoring micro-burstiness/latency at the router level [KLSV09, LMP+08]

Some natural questions

- Which stream has the highest average latency?
- What is the median latency of the k worst streams?
- How many streams have their 95th percentile latency less than a given value?
Outline

- Problem Definition
- Motivation
- Previous Work
- Approximated Weight Functions
- Outline of Results
- Lower Bounds
- Experimental Results
- Conclusion
Previous Work

Aggregate Measures on Streams

- One pass algorithms [MG82, MP80, AMS96]
- frequent items, heavy hitters [CMH08, CFC04, KSP03, MM02, MAA05, SLC07]
- quantiles/inverse quantiles [SBAS04, GK01]
Previous Work

**Aggregate Measures on Streams**

- One pass algorithms [MG82, MP80, AMS96]
- frequent items, heavy hitters [CMH08, CFC04, KSP03, MM02, MAA05, SLC+07]
- quantiles/inverse quantiles [SBAS04, GK01]

Previous work focus mainly on cumulative size/count of items in a stream.
Previous Work

Aggregate Measures on Streams

- One pass algorithms [MG82, MP80, AMS96]
- frequent items, heavy hitters [CMH08, CFC04, KSP03, MM02, MAA05, SLC+07]
- quantiles/inverse quantiles [SBAS04, GK01]

Previous work focus mainly on cumulative size/count of items in a stream.
Our model requires peering into individual streams, at the “micro” level
Outline

Problem Definition

Motivation

Previous Work

Approximated Weight Functions

Outline of Results

Lower Bounds

Experimental Results

Conclusion
Approximated Weight Functions

We focus on \textit{approximated} measures, let $\lambda$ an arbitrary weight function.
Approximated Weight Functions

We focus on *approximated* measures, let $\lambda$ an arbitrary weight function.

**Rank Approximation**

- $\text{rank}(x, S)$ denotes the rank of the element $x$ in stream $S$. For example, if $x_{\text{max}}$ is the largest element of $S$ then $\text{rank}(x_{\text{max}}, S) = 1$.
Approximated Weight Functions

We focus on *approximated* measures, let \( \lambda \) an arbitrary weight function.

Rank Approximation

- \( \text{rank}(x, S) \) denotes the rank of the element \( x \) in stream \( S \).
  - For example, if \( x_{\text{max}} \) is the largest element of \( S \) then
    \( \text{rank}(x_{\text{max}}, S) = 1 \)
- \( \lambda'_i \) is a rank approximation of \( \lambda_i = \lambda(S_i) \) with error \( E \) if:
  \[
  |\text{rank}(\lambda'_i, S_i) - \text{rank}(\lambda_i, S_i)| \leq E
  \]
Approximated Weight Functions

We focus on approximated measures, let $\lambda$ an arbitrary weight function.

Rank Approximation

- $\text{rank}(x, S)$ denotes the rank of the element $x$ in stream $S$. For example, if $x_{\text{max}}$ is the largest element of $S$ then $\text{rank}(x_{\text{max}}, S) = 1$

- $\lambda'_i$ is a rank approximation of $\lambda_i = \lambda(S_i)$ with error $E$ if:

$$|\text{rank}(\lambda'_i, S_i) - \text{rank}(\lambda_i, S_i)| \leq E$$

Example (median): $|\text{rank}(\lambda'_{i\text{med}}, S_i) - \lfloor|S_i|/2\rfloor| \leq E$
Approximated Weight Functions

We focus on *approximated* measures, let $\lambda$ an arbitrary weight function

**Rank Approximation**

- $\text{rank}(x, S)$ denotes the rank of the element $x$ in stream $S$. For example, if $x_{\text{max}}$ is the largest element of $S$ then $\text{rank}(x_{\text{max}}, S) = 1$
- $\lambda_i'$ is a rank approximation of $\lambda_i = \lambda(S_i)$ with error $E$ if:

$$|\text{rank}(\lambda_i', S_i) - \text{rank}(\lambda_i, S_i)| \leq E$$

**Example (median):**

$$|\text{rank}(\lambda_i^{\text{med}}, S_i) - \lfloor |S_i|/2 \rfloor| \leq E$$

**Value Approximation**

- $\lambda_i'$ is a value approximation of $\lambda_i$ with relative error $c$ if

$$|\lambda_i' - \lambda_i| \leq c\lambda_i$$
Outline

Problem Definition

Motivation

Previous Work

Approximated Weight Functions

Outline of Results

Lower Bounds

Experimental Results

Conclusion
A Taste of the Results

Top $k$ of $\lambda_{\text{max}}$ is Easy

- top $k$ streams under $\lambda_{\text{max}}$ (or $\lambda_{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
A Taste of the Results

Top $k$ of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

Top \( k \) of \( \lambda_{\text{max}} \) is Easy

- Top \( k \) streams under \( \lambda_{\text{max}} \) (or \( \lambda_{\text{min}} \)) can be computed exactly in \( O(k) \)-space, \( O(\log k) \) per-item processing
- Achieved by maintaining a heap of \( k \) distinct streams with the largest item values
A Taste of the Results

Top $k$ of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

Top $k$ of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

Top $k$ of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

**Top k of $\lambda^{\text{max}}$ is Easy**

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values

What if a more robust indicator than $\lambda^{\text{max}}$ is sought?
A Taste of the Results

Top k of $\lambda_{\text{max}}$ is Easy

- top k streams under $\lambda_{\text{max}}$ (or $\lambda_{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of k distinct streams with the largest item values
A Taste of the Results

Top $k$ of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed \textit{exactly} in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

Top $k$ of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

Top $k$ of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

Top k of $\lambda^{\text{max}}$ is Easy

- top $k$ streams under $\lambda^{\text{max}}$ (or $\lambda^{\text{min}}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values
A Taste of the Results

Top $k$ of $\lambda^\text{max}$ is Easy

- top $k$ streams under $\lambda^\text{max}$ (or $\lambda^\text{min}$) can be computed exactly in $O(k)$-space, $O(\log k)$ per-item processing
- achieved by maintaining a heap of $k$ distinct streams with the largest item values

What if a more robust indicator than $\lambda^\text{max}$ is sought?
A Surprising Negative Result

Top $k$ of $\lambda^{2\max}$

- $\lambda_i^{2\max}$ returns the *second largest* item in the stream $S_i$
A Surprising Negative Result

Top k of $\lambda^{2\max}$ is hard

- $\lambda_i^{2\max}$ returns the second largest item in the stream $S_i$
- finding the top stream by $\lambda^{2\max}$ requires $\Omega(m)$ memory (recall $m = |\mathcal{B}|$)

Theorem

Determining the top stream by $\lambda^{2\max}$ value within approximation factor $t$ requires at least $\Omega(m/t^{2+\gamma})$ space, for any $\gamma > 0$, where $m$ is the number of streams in the braid and $t \geq 2$ is an integer.
A Surprising Negative Result

Top $k$ of $\lambda^{2\text{max}}$ is hard

- $\lambda^{2\text{max}}_i$ returns the second largest item in the stream $S_i$
- finding the top stream by $\lambda^{2\text{max}}$ requires $\Omega(m)$ memory (recall $m = |\mathcal{B}|$)

Theorem

Determining the top stream by $\lambda^{2\text{max}}$ value within approximation factor $t$ requires at least $\Omega(m/t^{2+\gamma})$ space, for any $\gamma > 0$, where $m$ is the number of streams in the braid and $t \geq 2$ is an integer.

- Similar results for $\lambda^{\text{med}}$, $\lambda^{\text{avg}}$, $\lambda^{\text{spread}}$
Outline

Problem Definition

Motivation

Previous Work

Approximated Weight Functions

Outline of Results

Lower Bounds

Experimental Results

Conclusion
A Generic Lower Bound Framework

The *Multi-party Set-disjointness* Problem [NK97]

- $t$ players, a set of items $\mathcal{A} = \{1, 2, \ldots, m\}$
- The player $i$ ($i = 1, 2, \ldots, t$) holds a subset $\mathcal{A}_i \subseteq \mathcal{A}$
- promise: either all $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_t$ are pairwise disjoint (YES instance) or they *all* share a common element but are otherwise disjoint (NO instance).
A Generic Lower Bound Framework

The Multi-party Set-disjointness Problem [NK97]

- $t$ players, a set of items $A = \{1, 2, \ldots, m\}$
- The player $i$ ($i = 1, 2, \ldots, t$) holds a subset $A_i \subseteq A$
- promise: either all $A_1, A_2, \ldots, A_t$ are pairwise disjoint (YES instance) or they all share a common element but are otherwise disjoint (NO instance).
- Goal: decide if a given instance is YES or NO
A Generic Lower Bound Framework

The *Multi-party Set-disjointness* Problem [NK97]

- $t$ players, a set of items $A = \{1, 2, \ldots, m\}$
- The player $i$ ($i = 1, 2, \ldots, t$) holds a subset $A_i \subseteq A$
- promise: either all $A_1, A_2, \ldots, A_t$ are pairwise disjoint (YES instance) or they *all* share a common element but are otherwise disjoint (NO instance).
- **Goal**: decide if a given instance is YES or NO
- only the bits exchanged among the players are counted.

The Reduction

Simulate a one-way multi-party set-disjointness protocol [BYJKS04] using a streaming algorithm for the top $k$ streams.

[BYJKS04] requires $\Omega(\frac{m}{t} + \gamma)$ bits, then the algorithm requires $\Omega(\frac{m}{t} + \gamma)$ bits.
A Generic Lower Bound Framework

The Multi-party Set-disjointness Problem [NK97]

- t players, a set of items $A = \{1, 2, \ldots, m\}$
- The player $i$ ($i = 1, 2, \ldots, t$) holds a subset $A_i \subseteq A$
- promise: either all $A_1, A_2, \ldots, A_t$ are pairwise disjoint (YES instance) or they all share a common element but are otherwise disjoint (NO instance).
- Goal: decide if a given instance is YES or NO
- only the bits exchanged among the players are counted.

The Reduction

- Simulate a one-way multi-party set-disjointness protocol [BYJKS04] using a streaming algorithm for the top $k$ streams.
- [BYJKS04] requires $\Omega(m/t^{1+\gamma})$ bits, then the algorithm requires $\Omega(m/t^{2+\gamma})$ bits
Illustration: LB for Top 1 of $\lambda^{med}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream $a$

Stream $b$

Stream $c$

Stream $d$

Stream $e$
Illustration: LB for Top 1 of $\lambda^{\text{med}}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream $a$

Stream $b$

Stream $c$

Stream $d$

Stream $e$
Illustration: LB for Top 1 of $\lambda^{med}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream a

Stream b

Stream c

Stream d

Stream e
Illustration: LB for Top 1 of $\lambda^{med}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream $a$

Stream $b$

Stream $c$

Stream $d$

Stream $e$

$p$
Illustration: LB for Top 1 of $\lambda^{\text{med}}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream $a$

Stream $b$

Stream $c$

Stream $d$

Stream $e$
Illustration: LB for Top 1 of $\lambda^{med}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream $a$

Stream $b$

Stream $c$

Stream $d$

Stream $e$
Illustration: LB for Top 1 of $\lambda_{med}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream a
Stream b
Stream c
Stream d
Stream e
Illustration: LB for Top 1 of $\lambda^\text{med}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

Stream a

Stream b

Stream c

Stream d

Stream e
Illustration: LB for Top 1 of $\lambda^{\text{med}}$

Player 1: $A_1 = \{a, b\}$

Player 2: $A_2 = \{a, c\}$

Player 3: $A_3 = \{a, d, e\}$

Player 4: $A_4 = \{a\}$

$$\max(1, 0, 0, 0, 0) = 1$$
Other Lower Bound Results

- **Theorem**
  Let $\mathcal{B}$ be a braid of $m$ streams, where each stream has $\Theta(n)$ elements. Then, determining the top stream in $\mathcal{B}$ by median value, within rank error $\varepsilon n$ ($0 < \varepsilon < 1/2$) requires space at least
  \[
  \Omega \left( m \left( \frac{1 - 2\varepsilon}{1 + 2\varepsilon} \right)^{2+\gamma} \right) .
  \]
  for arbitrarily small $\gamma > 0$.

- **Theorem**
  Determining the top stream by average value within relative error at most $t$ requires at least $\Omega \left( \frac{m}{t^{2+\gamma}} \right)$ space, where $m$ is the number of streams in the braid, $t \geq 2$ and $\gamma > 0$.

- **Theorem**
  Determining the top stream by the spread requires at least $\Omega(m)$ space, where $m$ is the number of streams in the braid.
The lower bounds rule out any rank approximation with error $O(\varepsilon n_i)$ ($n_i$ is the size of the top stream) using $o(m)$ memory.
The lower bounds rule out any rank approximation with error $O(\varepsilon n_i)$ ($n_i$ is the size of the top stream) using $o(m)$ memory.

We give algorithms with worst case approximation $O(\varepsilon \sum_i n_i)$. 
Algorithms

- The lower bounds rule out any rank approximation with error $O(\varepsilon n_i)$ ($n_i$ is the size of the top stream) using $o(m)$ memory.
- We give algorithms with worst case approximation $O(\varepsilon \sum_i n_i)$.

Algorithm design

- General idea: assume items $v_{i,j}$ are $\in [1, U]$.
- Subdivide $[1, U]$ into buckets.
- All stream entries with a value $v$ are mapped to the bucket that contains $v$.
- Within a bucket, a Count-Min sketch [CM05] keeps track of the number of items belonging to different streams.
Algorithms (cont.)

Algorithm 1: **ExponentialBucket**

- [1, U] is split into pre-determined buckets \([l_b, r_b]\) such that the ratio \(r_b/l_b\) is constant.
- for each bucket, a Count-Min [CM05] keeps track of the number of items belonging to different streams.
Algorithm 1: **ExponentialBucket**

- [1, U] is split into pre-determined buckets [l_b, r_b] such that the ratio \( r_b / l_b \) is constant.
- for each bucket, a Count-Min [CM05] keeps track of the number of items belonging to different streams.

Algorithm 2: **VariableBucket**

- [1, U] is adaptively partitioned by a q-digest [SBAS04] data structure.
- each of the \( O(\rho^{-1} \log U) \) buckets contains \( O(\rho n) \) values.
- for each bucket, a Count-Min keeps track of the number of items belonging to different streams.
Guarantees for **ExponentialBucket** and **VariableBucket**

**Theorem**

*The ExponentialBucket is a data structure of size $O(\varepsilon^{-1} \log_{1+\rho} U \log \delta^{-1})$ that, with probability at least $1 - \delta$, can find the top $k$ streams in a set of $m$ streams by average, median, or any quantile value.*

**Theorem**

*The VariableBucket is a data structure of size $O(\varepsilon^{-2} \log U \log \delta^{-1})$ that, with probability at least $1 - \delta$, can find the top $k$ streams in a set of $m$ streams by average, median, or any quantile value, with (additive) rank approximation error $\varepsilon \sum_i n_i$.***
Outline

Problem Definition

Motivation

Previous Work

Approximated Weight Functions

Outline of Results

Lower Bounds

Experimental Results

Conclusion
Figures of Merit for Top $k$

For a given weight function $\lambda$:

**Precision** $S(k)$ is the true set of top $k$ streams, $S'(k)$ is the set of top $k$ streams returned by our algorithms, then

$$P(k) = \frac{|S(k) \cap S'(k)|}{|S'(k)|} = \frac{|S(k) \cap S'(k)|}{k}$$

**Distortion** $r(S_i)$ is the true rank for a stream $S_i$, $r'(S_i)$ is the rank given by our heuristics

$$d_i = \begin{cases} r(S_i)/r'(S_i), & \text{if } r(S_i) \geq r'(S_i) \\ r'(S_i)/r(S_i), & \text{otherwise.} \end{cases}$$

The distortion $D(k)$ is the average $d_i$ for $i = 1, \ldots, k$

**Value error** the average value error $E(k)$ for the top $k$ is the average of the relative errors $e(i)$ over the $k$ streams.

$$e(i) = \frac{\lambda(S_i) - \lambda(S'_i)}{\lambda(S_i)}$$

**Memory Consumption** the memory usage of our schemes.
Datasets

For all datasets:

- The braid $\mathcal{B}$ is composed of 1000 streams
- stream $S_i$ are composed of 5000 items each
- values $v_{i,j}$ are chosen within $[1, 2^{16}]$ ($U = 2^{16}$)
- values within each stream $S_i$ are normally distributed with standard deviation $\gamma_i = U/20$
Datasets

For all datasets:

- The braid $\mathcal{B}$ is composed of 1000 streams
- stream $S_i$ are composed of 5000 items each
- values $v_{i,j}$ are chosen within $[1, 2^{16}]$ ($U = 2^{16}$)
- values within each stream $S_i$ are normally distributed with standard deviation $\gamma_i = U/20$

Means ($\mu_i$) for the Gaussian distributions are chosen differently for different datasets:
Datasets

For all datasets:

- The braid $\mathcal{B}$ is composed of 1000 streams
- Stream $S_i$ are composed of 5000 items each
- Values $v_{i,j}$ are chosen within $[1, 2^{16}]$ ($U = 2^{16}$)
- Values within each stream $S_i$ are normally distributed with standard deviation $\gamma_i = U/20$

Means ($\mu_i$) for the Gaussian distributions are chosen differently for different datasets:

Uniform distribution $\mu_i$ for stream $S_i$ are uniformly chosen from the range $U = [1, 2^{16}]$.

Outlier distributions $\mu_i$ for 900 of the 1000 streams are chosen uniformly at random in the range $[0, 0.6U]$ and for the remaining 100 streams in the range $[aU, (a + 0.2)U]$, with $a < 1$, for different $a$.

Normal distribution $\mu_i$s are chosen from a normal distribution with mean $2^{15}$, and standard deviation $2^{14}$. 
Results

Precision and Error for VARIABLE_BUCKET

(a) Precision for $\lambda=\text{average}$

(b) Precision for $\lambda=\text{median}$

(c) Precision for $\lambda=95\text{th percentile}$

(d) Average value error for $\lambda=\text{average}$

(e) Average value error for $\lambda=\text{median}$

(f) Precision for median $\lambda=95\text{th percentile}$
Results (cont.)

- **Comparison between VARIABLE_BUCKET and EXPONENTIAL_BUCKET**

- **(g) Precision for median, EXPONENTIAL_BUCKET**

- **(h) Precision for median, VARIABLE_BUCKET**

- **Memory usage of VARIABLE_BUCKET for up to 10000 streams**
Outline

Problem Definition
Motivation
Previous Work
Approximated Weight Functions
Outline of Results
Lower Bounds
Experimental Results
Conclusion
Conclusion and Future Work

Conclusion

- Last decade’s research in DS focused on aggregated level: heavy hitters, quantiles of a single stream
Conclusion and Future Work

Conclusion

- Last decade’s research in DS focused on aggregated level: heavy hitters, quantiles of a single stream
- Recently the interest has shifted towards finer level statistics motivated by monitoring of performance in large, shared systems.

Future Work

Small-footprint algorithms for natural distributions
Conclusion

- Last decade’s research in DS focused on aggregated level: heavy hitters, quantiles of a single stream.
- Recently the interest has shifted towards finer level statistics motivated by monitoring of performance in large, shared systems.
- We define the problem of tracking outlier streams in a large set (braid) of streams in one-pass. Different measures analyzed: average, median, quantiles.
- We prove space lower bounds for several natural measures.
- We propose two heuristics with error guarantees, and analyzed their performance.

Future Work

Small-footprint algorithms for natural distributions.
Conclusion and Future Work

Conclusion

▶ Last decade’s research in DS focused on aggregated level: heavy hitters, quantiles of a single stream
▶ Recently the interest has shifted towards finer level statistics motivated by monitoring of performance in large, shared systems.
▶ We define the problem of tracking outlier streams in a large set (braid) of streams in one-pass. Different measures analyzed: average, median, quantiles.
▶ We prove space lower bounds for several natural measures.
▶ We propose two heuristics with error guarantees, and analyzed their performance.

Future Work
Small-footprint algorithms for natural distributions
References I


References II


