

# 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

# Streams and Braids

## Problem Definition

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

# Streams and Braids

## Problem Definition

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

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

## 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
- ▶  $|\mathcal{B}| = m \gg 1$ ,  $n_i = |S_i| \gg 1$

**Goal: one-pass algorithm**

# Example

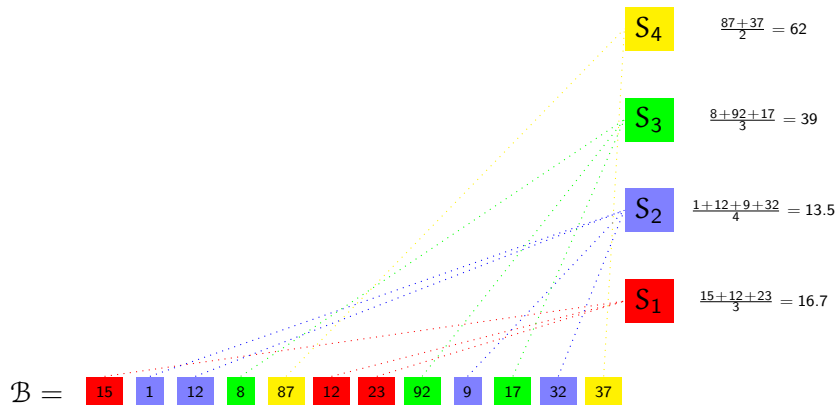
Worst 2 Streams by Average ( $\lambda^{\text{avg}}$ )

$\mathcal{B} =$ 

15	1	12	8	87	12	23	92	9	17	32	37
----	---	----	---	----	----	----	----	---	----	----	----

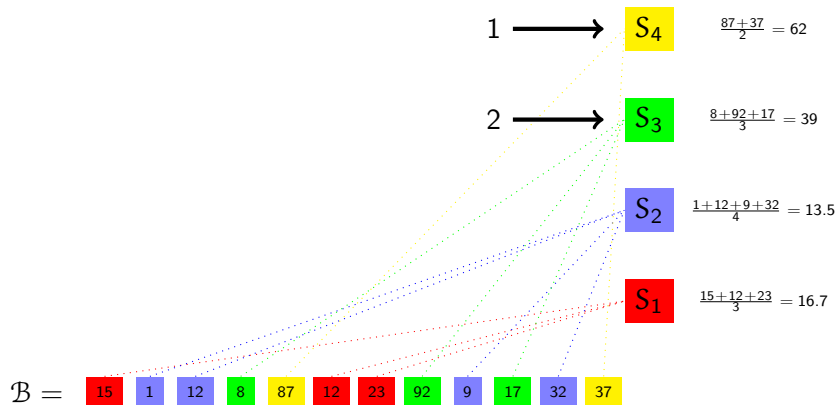
# Example

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



# Example

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



# 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<sup>+</sup>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<sup>+</sup>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, SLC<sup>+</sup>07]
- ▶ 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<sup>+</sup>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<sup>+</sup>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 *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  $x_{\max}$  is the largest element of  $S$  then  
 $\text{rank}(x_{\max}, S) = 1$

# 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  $x_{\max}$  is the largest element of  $S$  then  $\text{rank}(x_{\max}, S) = 1$
- ▶  $\lambda'_i$  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  $x_{\max}$  is the largest element of  $S$  then  $\text{rank}(x_{\max}, S) = 1$
- ▶  $\lambda'_i$  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  $x_{\max}$  is the largest element of  $S$  then  $\text{rank}(x_{\max}, S) = 1$
- ▶  $\lambda'_i$  is a rank approximation of  $\lambda_i = \lambda(S_i)$  with error  $\epsilon$  if:

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

**Example (median):**  $|\text{rank}(\lambda_i^{\text{med}}, S_i) - \lfloor |S_i|/2 \rfloor| \leq \epsilon$

## 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

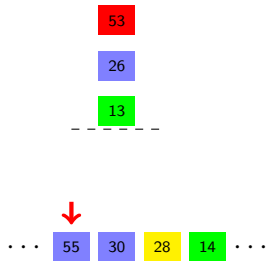
## Top $k$ of $\lambda^{\max}$ is Easy

- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing

# A Taste of the Results

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

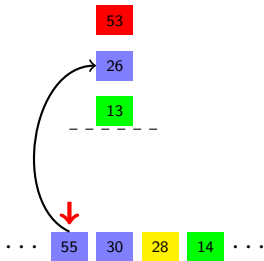
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

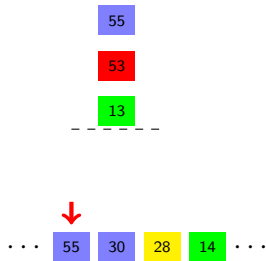
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

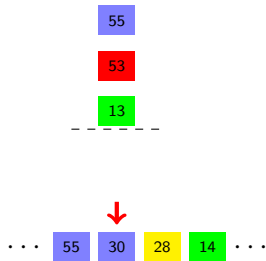
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

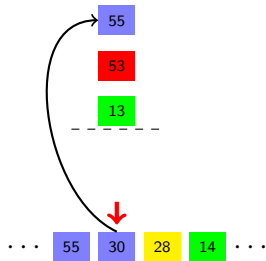
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

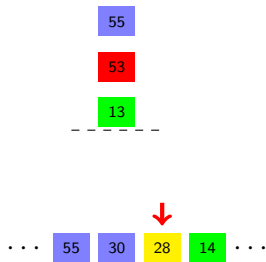
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

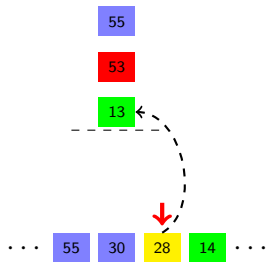
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

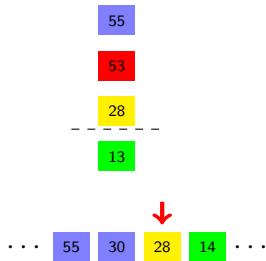
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

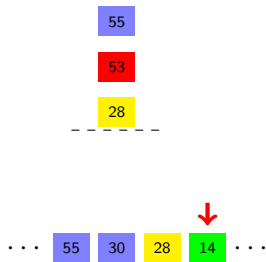
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

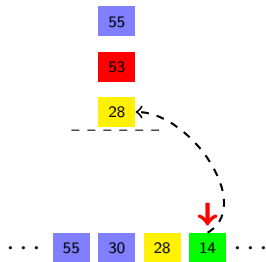
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

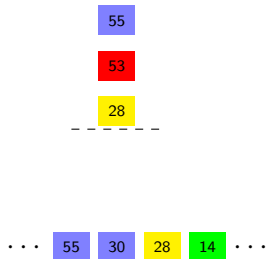
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

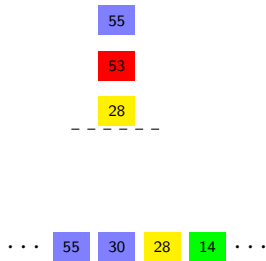
- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) can be computed *exactly* in  $O(k)$ -space,  $O(\log k)$  per-item processing
- ▶ achieved by maintaining a heap of  $k$  distinct streams with the largest item values



# A Taste of the Results

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

- ▶ top  $k$  streams under  $\lambda^{\max}$  (or  $\lambda^{\min}$ ) 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  $\lambda^{\max}$  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  $\lambda^{2m_{\max}}$  is hard

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

## Theorem

*Determining the top stream by  $\lambda^{2m_{\max}}$  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  $\lambda^{2m_{\max}}$  is hard

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

## Theorem

*Determining the top stream by  $\lambda^{2m_{\max}}$  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  $A = \{1, 2, \dots, m\}$
- ▶ The player  $i$  ( $i = 1, 2, \dots, t$ ) holds a subset  $A_i \subseteq A$
- ▶ promise: either all  $A_1, A_2, \dots, A_t$  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  $A = \{1, 2, \dots, m\}$
- ▶ The player  $i$  ( $i = 1, 2, \dots, t$ ) holds a subset  $A_i \subseteq A$
- ▶ promise: either all  $A_1, A_2, \dots, A_t$  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  $A = \{1, 2, \dots, m\}$
- ▶ The player  $i$  ( $i = 1, 2, \dots, t$ ) holds a subset  $A_i \subseteq A$
- ▶ promise: either all  $A_1, A_2, \dots, A_t$  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.

# A Generic Lower Bound Framework

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

- ▶  $t$  players, a set of items  $A = \{1, 2, \dots, m\}$
- ▶ The player  $i$  ( $i = 1, 2, \dots, t$ ) holds a subset  $A_i \subseteq A$
- ▶ promise: either all  $A_1, A_2, \dots, A_t$  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 $\lambda^{\text{med}}$

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

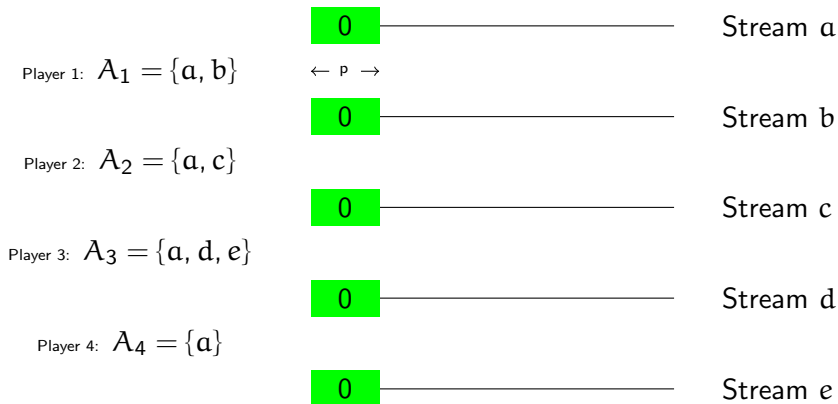
---

Stream d

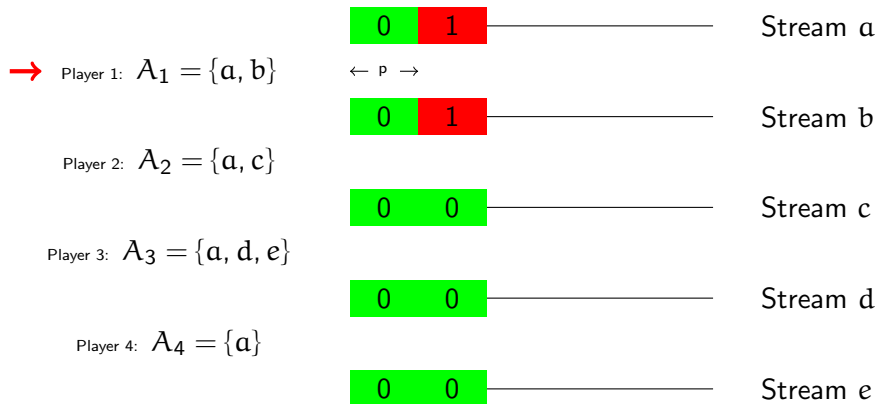
---

Stream e

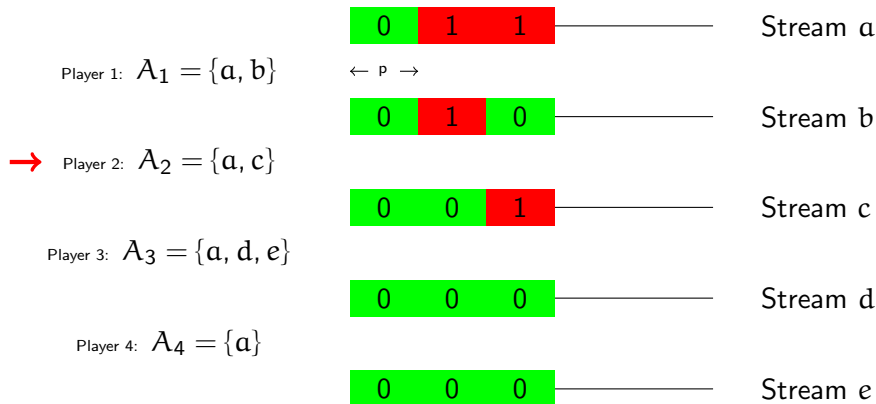
# Illustration: LB for Top 1 of $\lambda^{\text{med}}$



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



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



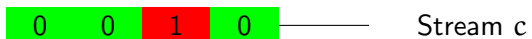
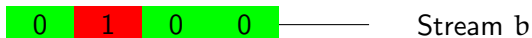
# Illustration: LB for Top 1 of $\lambda^{\text{med}}$

Player 1:  $\mathcal{A}_1 = \{a, b\}$



← P →

Player 2:  $\mathcal{A}_2 = \{a, c\}$



→ Player 3:  $\mathcal{A}_3 = \{a, d, e\}$



Player 4:  $\mathcal{A}_4 = \{a\}$



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

Player 1:  $\mathcal{A}_1 = \{\mathbf{a}, \mathbf{b}\}$



← P →

Player 2:  $\mathcal{A}_2 = \{\mathbf{a}, \mathbf{c}\}$



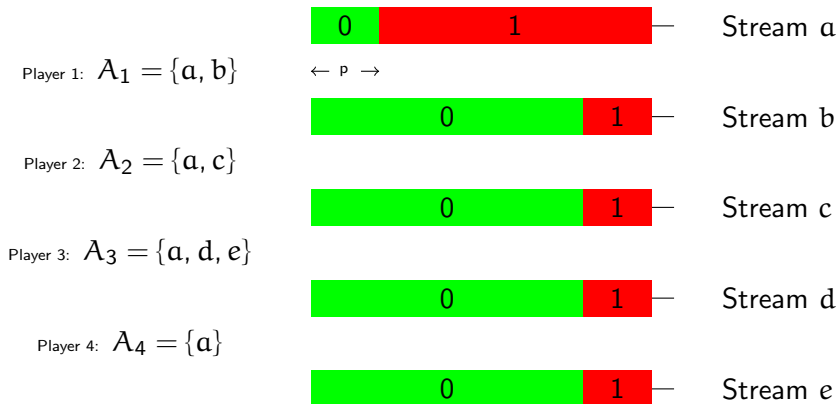
Player 3:  $\mathcal{A}_3 = \{\mathbf{a}, \mathbf{d}, \mathbf{e}\}$



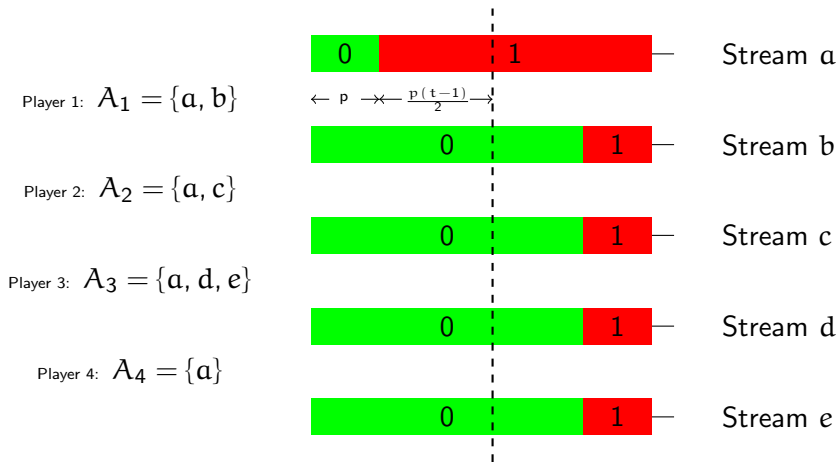
→ Player 4:  $\mathcal{A}_4 = \{\mathbf{a}\}$



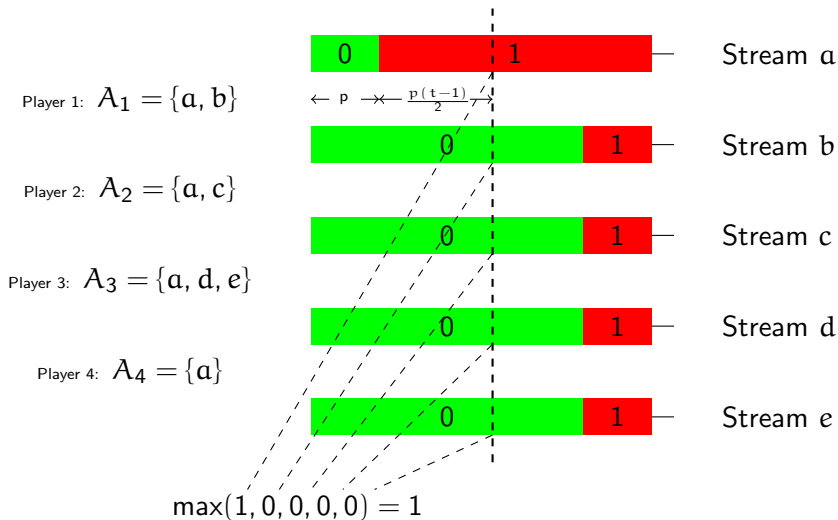
# Illustration: LB for Top 1 of $\lambda^{\text{med}}$



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



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



## 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  $\Omega(m/t^{2+\gamma})$  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.*

# Algorithms

- ▶ The lower bounds rule out any rank approximation with error  $O(\epsilon n_i)$  ( $n_i$  is the size of the top stream) using  $o(m)$  memory

# Algorithms

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

# Algorithms

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

## 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.

# 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 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**  $\mathcal{S}(k)$  is the true set of top  $k$  streams,  $\mathcal{S}'(k)$  is the set of top  $k$  streams returned by our algorithms, then

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

**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  $i = 1, \dots, k$

**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.

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

**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  $[1, 2^{16}]$  ( $\mathcal{U} = 2^{16}$ )
- ▶ values within each stream  $S_i$  are normally distributed with standard deviation  $\gamma_i = \mathcal{U}/20$

# 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  $[1, 2^{16}]$  ( $\mathcal{U} = 2^{16}$ )
- ▶ values within each stream  $S_i$  are normally distributed with standard deviation  $\gamma_i = \mathcal{U}/20$

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:

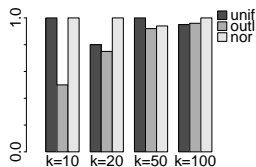
**Uniform distribution**  $\mu_i$  for stream  $S_i$  are uniformly chosen from the range  $U = [1, 2^{16}]$ .

**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  $a$ .

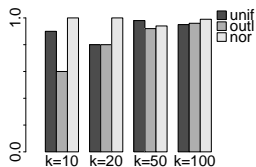
**Normal distribution**  $\mu_i$ s are chosen from a normal distribution with mean  $2^{15}$ , and standard deviation  $2^{14}$ .

# Results

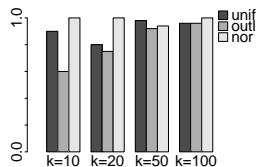
## Precision and Error for VARIABLEBUCKET



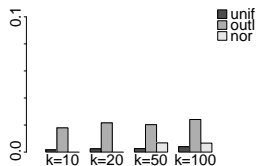
(a) Precision for  $\lambda$ =average



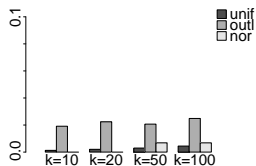
(b) Precision for  $\lambda$ =median



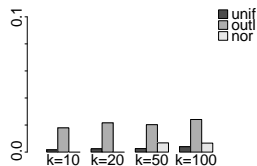
(c) Precision for  $\lambda$ =95th percentile



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



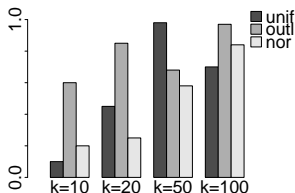
(e) Average value error for  $\lambda$ =median



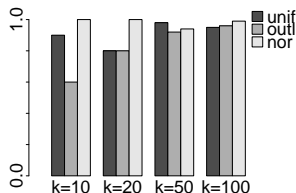
(f) Precision for median  $\lambda$ =95th percentile

## Results (cont.)

- ▶ Comparison between `VARIABLEBUCKET` and `EXPONENTIALBUCKET`

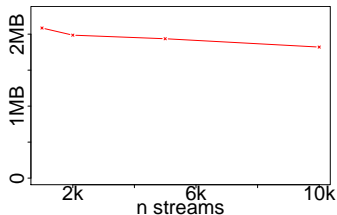


(g) Precision for median, `EXPONENTIALBUCKET`



(h) Precision for median, `VARIABLEBUCKET`

- ▶ Memory usage of `VARIABLEBUCKET` 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.

# 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.

# 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



N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating the frequency moments*, Proc. of STOC, 1996.



Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, *An information statistics approach to data stream and communication complexity*, J. Comput. Syst. Sci. **68** (2004), no. 4, 702–732.



M. Charikar and K. Chen M. Farach-Colton, *Finding frequent items in data streams*, Theoretical Computer Science **312** (2004), no. 1, 3–15.



G. Cormode and S. Muthukrishnan, *An improved data stream summary: the count-min sketch and its applications*, Journal of Algorithms **55** (2005), no. 1, 58–75.



G. Cormode and M. M. Hadjieleftheriou, *Finding frequent items in data streams*, Proceedings of VLDB **1** (2008), no. 2, 1530–1541.



M. Greenwald and S. Khanna, *Space-efficient online computation of quantile summaries*, Proc. of ACM SIGMOD, 2001, pp. 58–66.



R.R. Kompella, K. Levchenko, A.C. Snoeren, and G. Varghese, *Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator*, ACM SIGCOMM Computer Communication Review **39** (2009), no. 4, 255–266.



R. M. Karp, S. Shenker, and C. H. Papadimitriou, *A simple algorithm for finding frequent elements in streams and bags*, ACM TODS **28** (2003), no. 1, 51–55.



Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, *Counter braids: a novel counter architecture for per-flow measurement*, Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, ACM, 2008, pp. 121–132.



A. Metwally, D. Agrawal, and A. El Abbadi, *Efficient computation of frequent and top-k elements in data streams*, Proceedings of ICDT, 2005, pp. 398–412.

# References II



J. Misra and D. Gries, *Finding repeated elements*, *Sci. Comput. Programming* (1982), 143–152.



G. S. Manku and R. Motwani, *Approximate frequency counts over data streams*, *Proceedings of VLDB*, 2002, pp. 346–357.



I. Munro and M. S. Paterson, *Selection and sorting with limited storage*, *Theoretical Computer Science* (1980), 315–323.



Noam Nisan and Eyal Kushilevitz, Cambridge University Press, 1997.



N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, *Medians and beyond: new aggregation techniques for sensor networks*, *Proc. of ACM SenSys*, 2004, pp. 239–249.



R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M. Kao, and G. Memik, *Reversible sketches: enabling monitoring and analysis over high-speed data streams*, *IEEE/ACM Transactions on Networking* **15** (2007), 1059–1072.