IEEE International Conference on Computer Communications
1-4 May 2017 // Atlanta, GA, USA

Better with Fewer Bits: Improving the Performance of Cardinality Estimation of Large Data Streams

Qingjun Xiao

School of Computer Science & Engineering,
Southeast University, China.
Email: csqjxiao@seu.edu.cn

You Zhou, Shigang Chen

Dept. of Computer and Information Science & Engineering,
University of Florida, USA.
Email: {youzhou, sgchen}@cise.ufl.edu

What is the cardinality estimation problem?

Elements occur multiple times, we want to count the number of *distinct* elements.

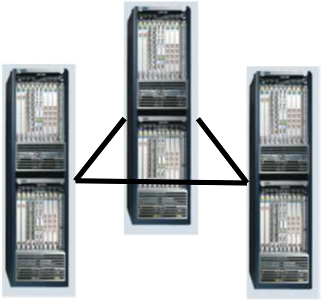
- Number of distinct element is $\boldsymbol{n}$ (= 6 in example)
- Number of elements in this example is 11

32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

A list of applications

Counting the number of unique visitors (100m+ daily visits) -- the most important metric in **online advertising**

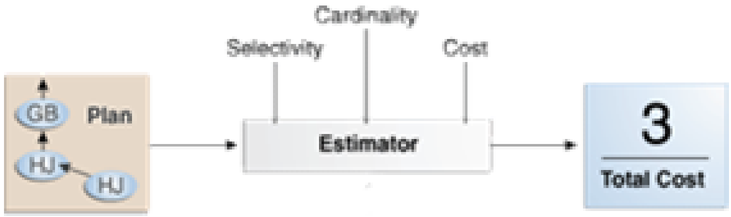
- See Redis HyperLogLog data structure



ISP measurement of traffic usage

Routers traffic in the range of Terabits/sec (10^{12} b/s)

Internet-scale data measurement: Google indexes 6+ billions pages, and counts the number of accesses to each pages, and also counts the number of searches towards each keyword

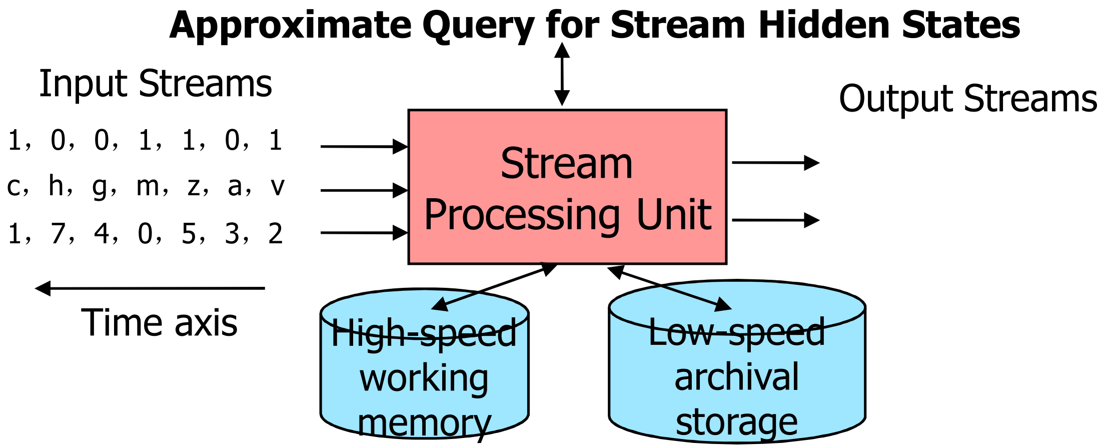


Cardinality in **DB queries optimization**

- the number of rows returned by each operation in an execution plan

Rules of the game

- **Limited memory**: cannot store all the stream elements; can use just one page of memory footprint, about 4 kB, in order to fit into the high-speed working memory.
- **Limited time**: online processing of the stream data, read the data by a single pass, or read-once data.



- Allow to generate an **approximate estimate** of the cardinality n, rather than compute its exact value
- Assume there is a **uniform hash function** $h : \mathcal{D} \to [0,1]$, to map the stream elements uniformly and pseudo-randomly

One of fundamental stream processing techniqs

Give a (large) sequence of data values over a (large) domain $\boldsymbol{\mathcal{D}}$

$$\boldsymbol{S = s_1\, s_2\, \cdots\, s_\ell}, \qquad \boldsymbol{s_j \in \mathcal{D}}$$

View the stream $\boldsymbol{S}$ as a multiset $\boldsymbol{MS}$:

$$\boldsymbol{MS = {e_1}^{f_1}\, {e_2}^{f_2}\, \cdots\, {e_n}^{f_n}}, \qquad \boldsymbol{s_j \in \mathcal{D}}$$

Element $\boldsymbol{e_i}$ has $\boldsymbol{f_i}$ repetitions, or say the frequency of $\boldsymbol{e_i}$ is $\boldsymbol{f_i}$.

Stream Processing Problems:

►Size Estimation: What is the size $\boldsymbol{\ell}$ of the stream?

►Cardinality Estimation: How many different elements are present?

►Elephants Identification: What are the elements with absolute frequencies above a threshold, e.g., $\boldsymbol{f_i > 500}$?

►Icebergs Identification: What are the elements with relative frequencies above a threshold, e.g., $\boldsymbol{\frac{1}{\ell} f_i > \frac{1}{100}}$?

►Frequency moment estimation: $(\sum \boldsymbol{f_i}^r)^{1/r}$

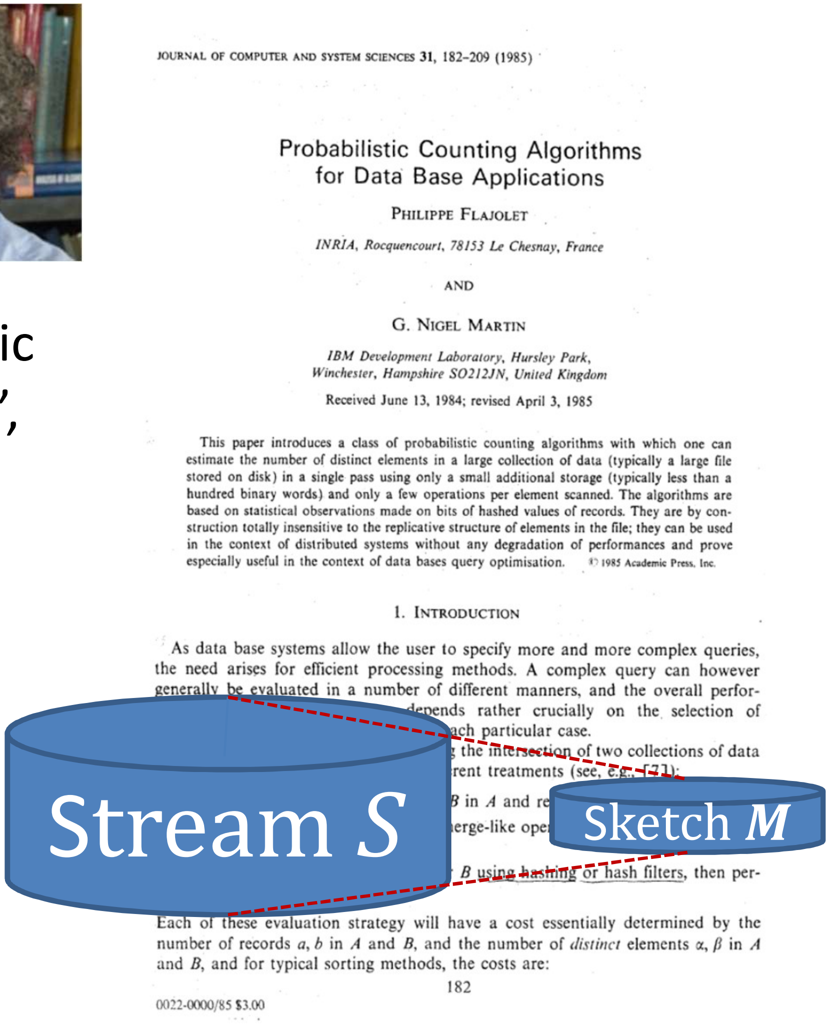
Probabilistic counting with stochastic averaging

Philippe Flajolet, 1948-2011,
mathematician and computer scientist extraordinaire

Philippe Flajolet, G. Nigel Martin, "Probabilistic counting algorithms for database applications", Proc. of FOCS 1983, JCSS 1985

Contributions of PCSA

- Introduced the problem
- Idea of streaming algorithm
- Idea of "small" sketch of "big" data
- Detailed analysis that yields tight bounds on accuracy



JOURNAL OF COMPUTER AND SYSTEM SCIENCES 31, 182-209 (1985)

Probabilistic Counting Algorithms
for Data Base Applications

PHILIPPE FLAJOLET

INRIA, Rocquencourt, 78153 Le Chesnay, France

AND

G. NIGEL MARTIN

IBM Development Laboratory, Hursley Park,
Winchester, Hampshire SO212JN, United Kingdom

Received June 13, 1984; revised April 3, 1985

This paper introduces a class of probabilistic counting algorithms with which one can estimate the number of distinct elements in a large collection of data (typically a large file stored on disk) in a single pass using only a small additional storage (typically less than a hundred binary words) and only a few operations per element scanned. The algorithms are based on statistical observations made on bits of hashed values of records. They are by construction totally insensitive to the replicative structure of elements in the file; they can be used in the context of distributed systems without any degradation of performances and prove especially useful in the context of data bases query optimisation. © 1985 Academic Press, Inc.

1. INTRODUCTION

As data base systems allow the user to specify more and more complex queries, the need arises for efficient processing methods. A complex query can however generally be evaluated in a number of different manners, and the overall perfor-
depends rather crucially on the selection of
ach particular case.
the intersection of two collections of data
rent treatments (see, e.g., [7]):
B in A and re
erge-like ope
B using hashing or hash filters, then per-

Each of these evaluation strategy will have a cost essentially determined by the number of records a, b in A and B, and the number of *distinct* elements α, β in A and B, and for typical sorting methods, the costs are:

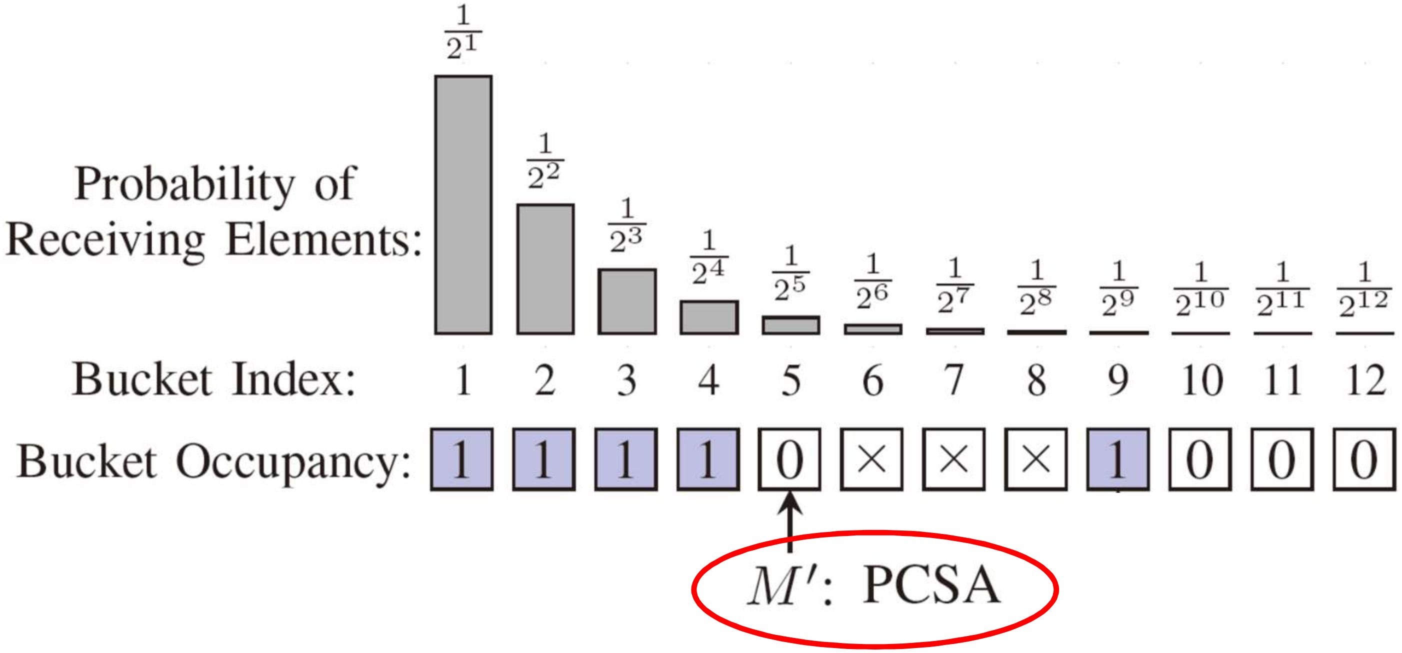
182

0022-0000/85 $3.00

Two key ideas of PCSA (1)

Probabilistic Counting Register

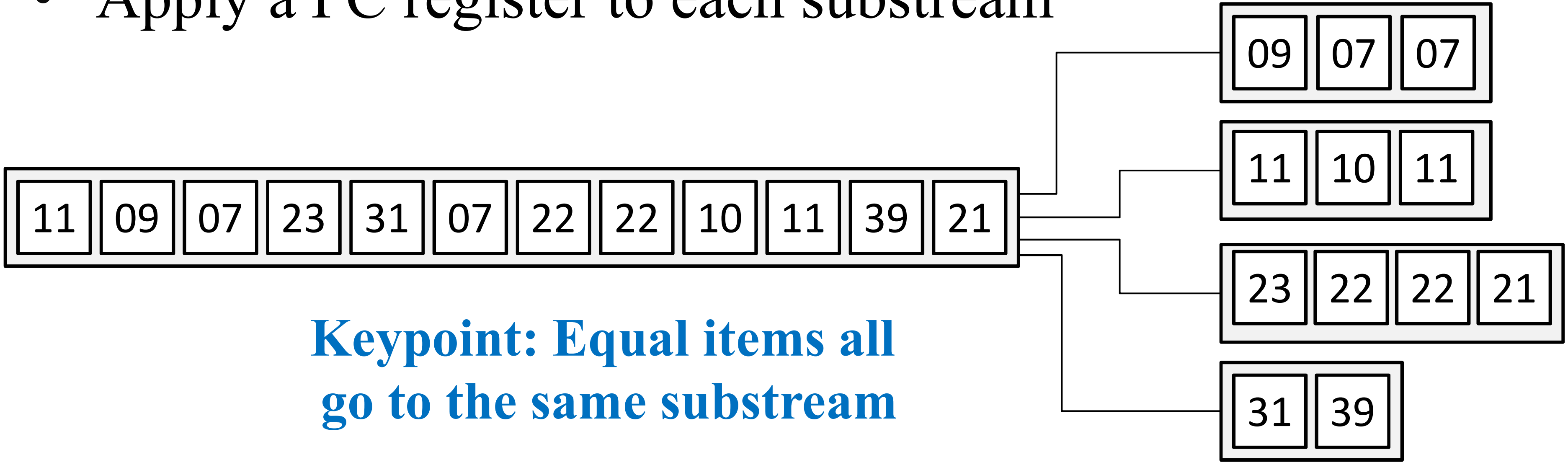
- Use a hash function to map incoming stream elements to a bit array with **exponentially deceasing probabilities**
- Find the position M' of the leftmost zero bit
- The bit array, or called a PC register, can generate a pretty coarse estimation for the cardinality of input stream as $2^{M'}$



Two key ideas of PCSA (2)

Stochastic Averaging

- Use a second hash function to divide the input stream into 2^m independent substreams
- Apply a PC register to each substream



Keypoint: Equal items all go to the same substream

- Compute $mean$ = average position of the left-most zero bits in the 2^m registers
- Return the result: $2^{mean}/0.77531$

Space-accuracy tradeoff for PCSA

Relative Accuracy: $\frac{\mathbf{0.78}}{\sqrt{\boldsymbol{m}}}$, where m is the number of registers, and each register is given 32 bits memory.

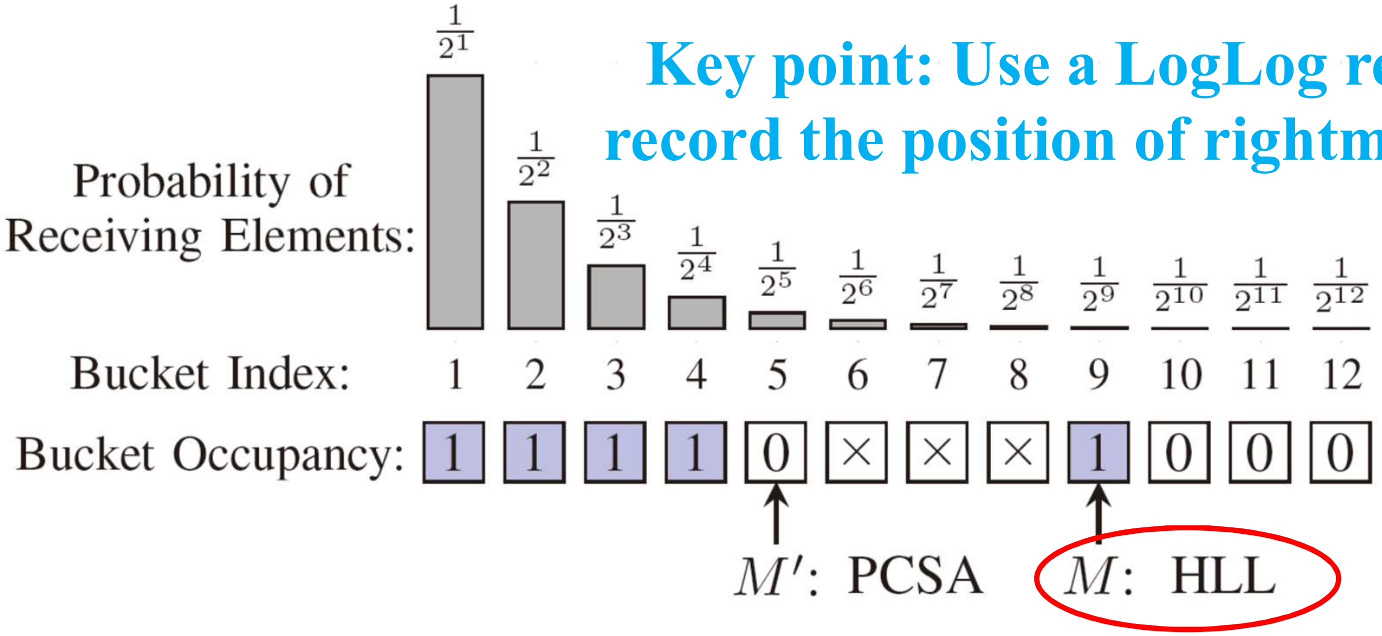
But the space-accuracy tradeoff of PCSA is no satisfactory. A plethora of algorithms are proposed for improvement.

	Algorithm	**Std. Err.(σ)**	**Mem Units**	**Mem(σ=2%)**
RANDOM'02	MinCount	$1.00/\sqrt{m}$	32-bit keys	10000 bytes
FOCS'83	PCSA	$0.78/\sqrt{m}$	32-bit registers	6084 bytes
IMC'03	MultiresBitmap	$\approx 4.4/\sqrt{m}$	1 bit	6050 bytes
ESA'03	LogLog	$1.30/\sqrt{m}$	5-bit registers	2641 bytes
AOFA'07	HyperLogLog	$1.04/\sqrt{m}$	5-bit registers	1690 bytes

HyperLogLog is the state-of-the-art!!! It can reduce memory cost by 72% for attaining the same accuracy

Valuable Ideas of HyperLogLog

- Use **loglog registers**, which are of log log(n) bits each



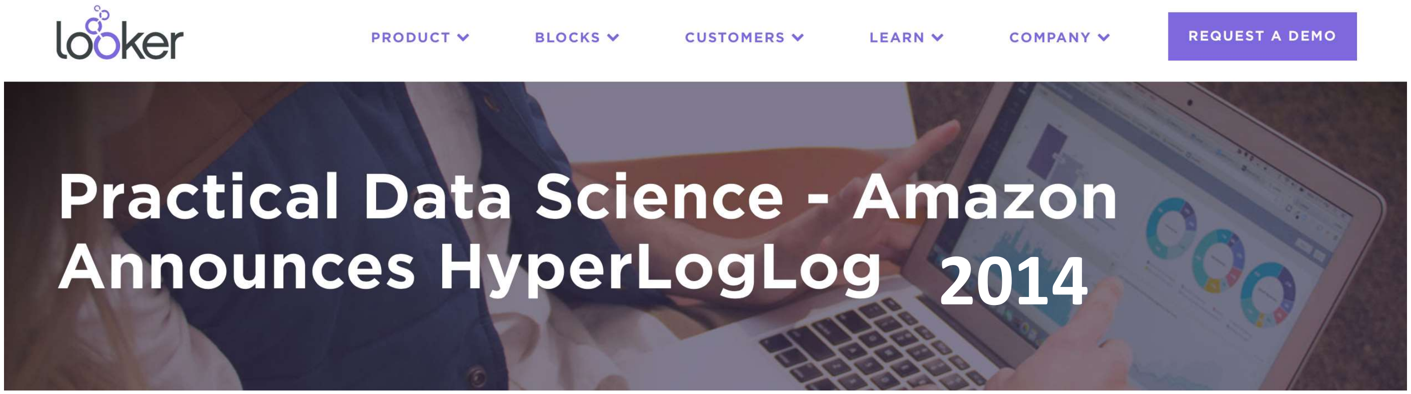
Probability of Receiving Elements:
$\frac{1}{2^1}$ $\frac{1}{2^2}$ $\frac{1}{2^3}$ $\frac{1}{2^4}$ $\frac{1}{2^5}$ $\frac{1}{2^6}$ $\frac{1}{2^7}$ $\frac{1}{2^8}$ $\frac{1}{2^9}$ $\frac{1}{2^{10}}$ $\frac{1}{2^{11}}$ $\frac{1}{2^{12}}$
Bucket Index: 1 2 3 4 5 6 7 8 9 10 11 12
Bucket Occupancy: 1 1 1 1 0 × × × 1 0 0 0
M': PCSA
M: HLL

Key point: Use a LogLog register to record the position of rightmost one-bit

- Use **harmonic averaging,** instead of geometric mean, to summarize the estimation results of m loglog registers

Philippe Flajolet, Éric Fusy, Olivier Gandouet, Frédéric Meunier, “HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm”, Proc. of AOFA (International Conference on the Analysis of Algorithms), 2007

HyperLogLog warmly embraced by industries



Practical Data Science - Amazon Announces HyperLogLog

2014

It appears difficult to further improve the space-accuracy tradeoff of HyperLogLog

HyperLogLog+ fixes some minor problems

S. Heule, M. Nunkesser and A. Hall, “HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm,” Proc. of EDBT (International Conference on Extending Database Technology), 2013.

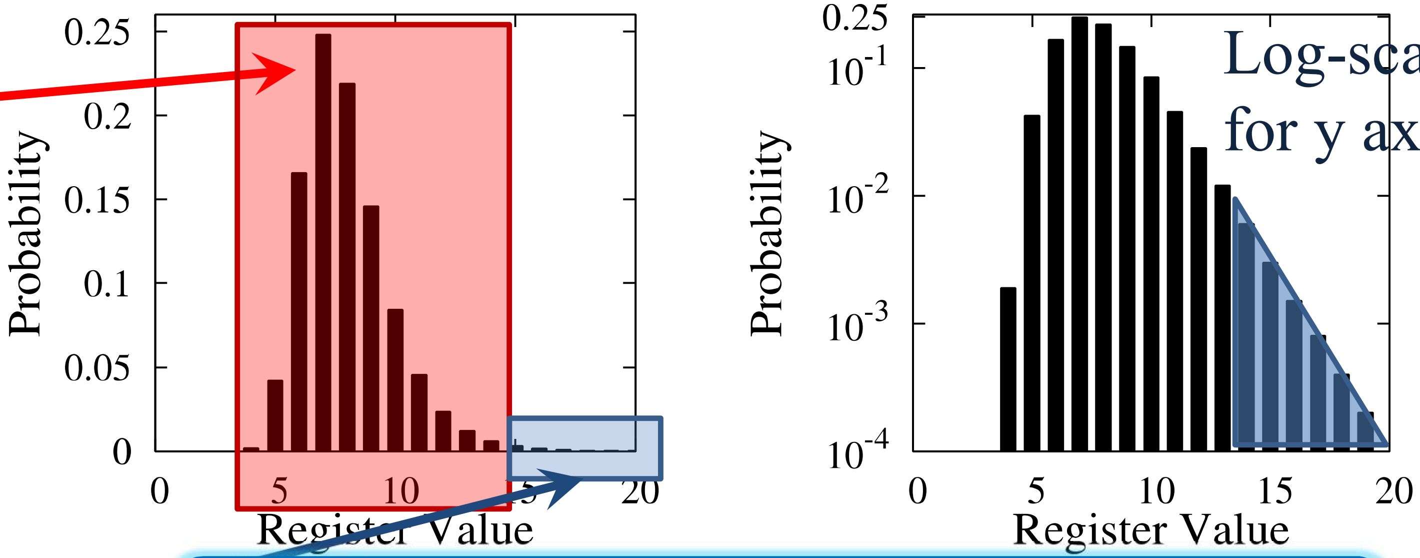
HyperLogLog has two major shortcomings

We discover that the HyperLogLog register values exhibit a right-skewed distribution, implying the following two facts.

- **Outliers with large values** exist in the rightside long tail
- **Inefficient to use 5 bits** to encode the register histogram

Much less than 2^5=32 effective bars

Long tail has not much useful information



Log-scale for y axis

(a)

Fig. 2. Pr

nd

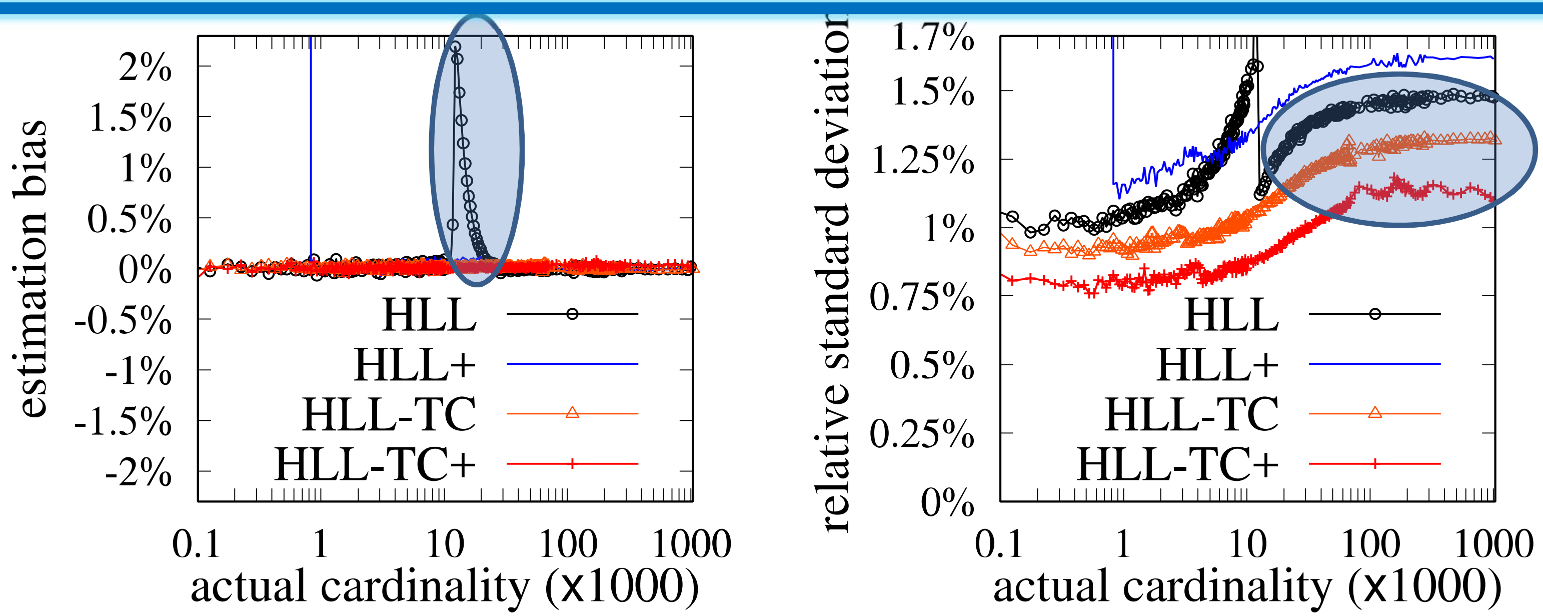
the number

Our technique: truncate the right-side tail to reject the outliers, and use less than five bits to encode the histogram

HyperLogLog has another minor shortcoming

- HyperLogLog has a **small biased region** from $2m$ to $5m$, since it uses LinearCounting for cardinalities n $< 2.5m$.

- Our HLL-TC and HLL-TC+ can remove the bias
- We greatly improve accuracy at the same memory cost



(a) Estimation bias

(b) Standard deviation

Fig. 6. Compare cardinality estimators with the same 24.58k bits memory.

What do we promise?

- We will propose two algorithms
  - HLL-TailCut needs $\boldsymbol{m}$ registers of four bits each, and provides relative accuracy $\frac{1.04}{\sqrt{m}}$
  - HLL-TailCut+ needs $\boldsymbol{m}$ registers of three bits each, and provides relative accuracy $\frac{1.00}{\sqrt{m}}$
- Both can support the counting of Tera- and Peta-scale data, while HyperLogLog can only support Giga-scale data.

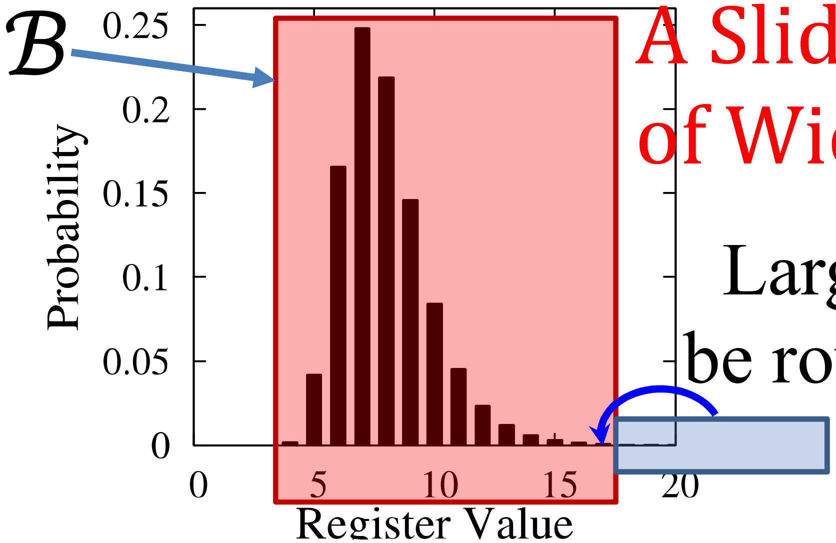
Algorithm	**Std. Err.(σ)**	**Mem Units**	**Mem(σ=2%)**
……	……	……	……
HyperLogLog			1690 bytes
HLL-TailCut			1352 bytes
HLL-TailCut+			938 bytes

HLL-TC needs 20% less memory

HLL-TC+ needs 45% less memor

How do we implement HLL-TailCut?

- **Improve the space-accuracy tradeoff (20% less memory cost)**, by reducing the size of a register to four bits only
  - Use a base register $\mathcal{B}$ to keep track of the value of the smallest HyperLogLog register
  - Use m offset registers to record the offsets of m HyperLogLog registers relative to the base $\mathcal{B}$
  - Each offset register is given four bits memory
  - A negligibly small portion of the long tail has been cut off



A Sliding Window of Width $\mathcal{K} = 2^4$

Large overflowed values will be rounded down to $\mathcal{B} + \mathcal{K} - 1$

How do we implement HLL-TailCut? (conti.)

- **Have addressed the problem of small biased region**
  - When the estimated cardinality is larger than $5m$, we still use the HyperLogLog equation

$$\hat{n} = \alpha_m \cdot m^2 \cdot \left(\sum_{0 \leq j < m} 2^{-(\mathcal{B}+\tilde{M}_j)} \right)^{-1} \qquad (9)$$

  - When the estimated cardinality is smaller than $2m$, we still use the LinearCounting equation
  - When the estimated cardinality is **between $2m$ and $5m$**, we use the following maximum likelihood estimation formula

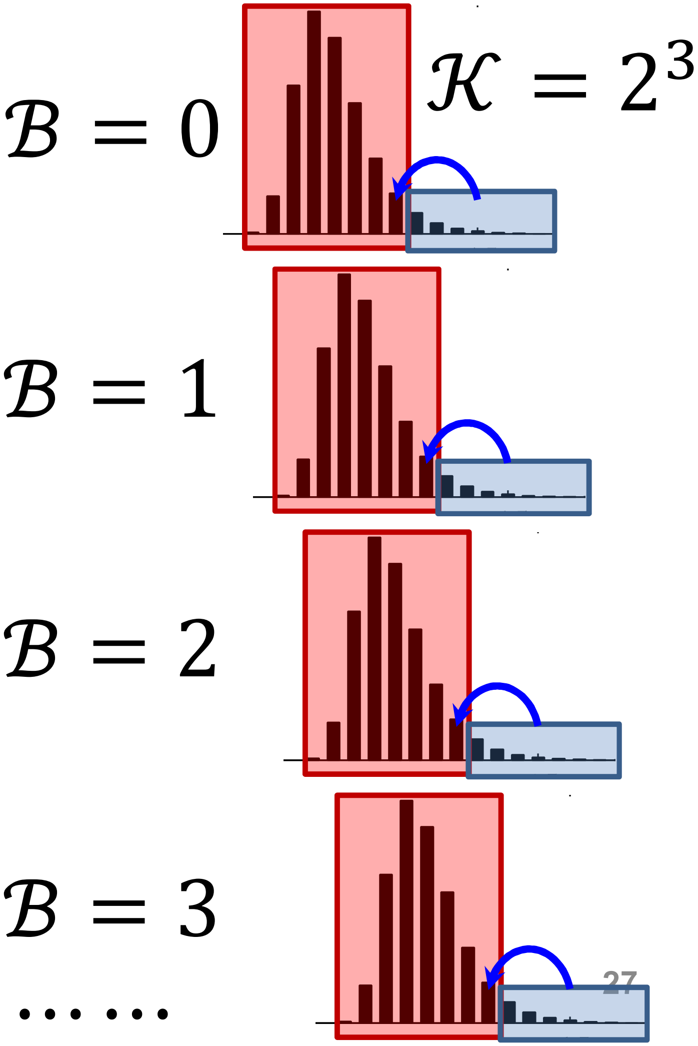
$$\hat{n} = \arg\max_{n} \log \mathcal{L}(n \mid N_0, N_1, \ldots, N_{\mathcal{K}-1}) \qquad (5)$$

$$\mathcal{L}(n \,|\, N_0, N_1, \ldots, N_{\mathcal{K}-1}) \approx \frac{m!}{N_1!N_2!\ldots N_{\mathcal{K}-1}!} \prod_{k=0}^{\mathcal{K}-1} Pr\{M_j{=}k\}^{N_k} (4)$$

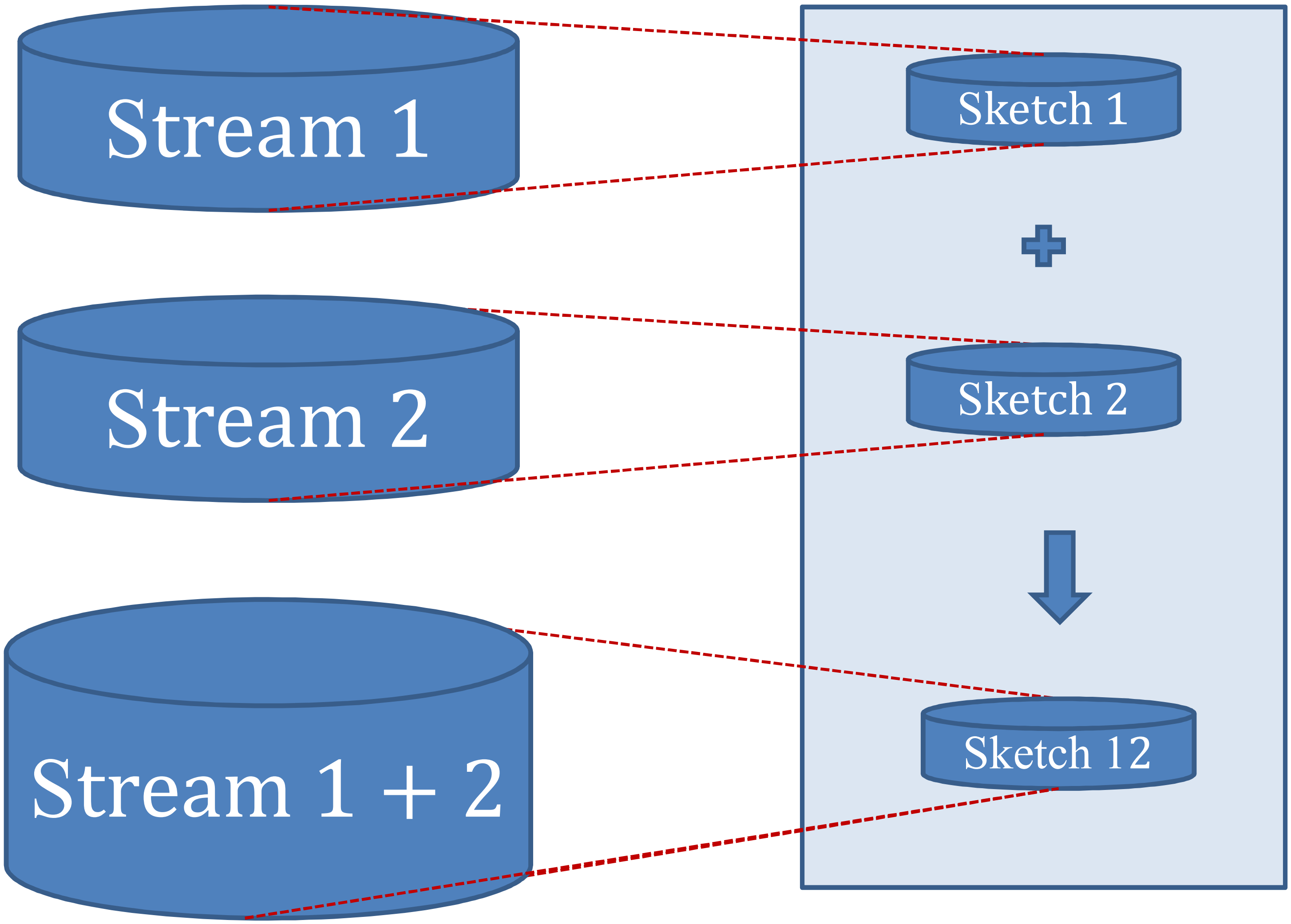
$$Pr\{M_j = k\} \approx \begin{cases} \left(1 - \frac{1}{m}\right)^n & \textit{if } k = 0, \\ (1 - \frac{1}{m2^k})^n - (1 - \frac{1}{m2^{k-1}})^n & \textit{if } k \geq 1. \end{cases} \quad (3)$$

How do we implement HLL-TailCut+?

- **Further improve the space-accuracy tradeoff (i.e., 45% less memory cost)**, by reducing the size of a register to three bits only, i.e., $\mathcal{K} = 2^3 = 8$
- When $\mathcal{K} = 8$, a non-negligible proportion of the right-side long tail has been truncated
- MUST consider the dynamically increasing process of the base $\mathcal{B}$
- Use a maximum likelihood estimator to determine the cardinality of newly arrived stream elements, when the base $\mathcal{B}$ equals $0, 1, 2, \cdots$, respectively

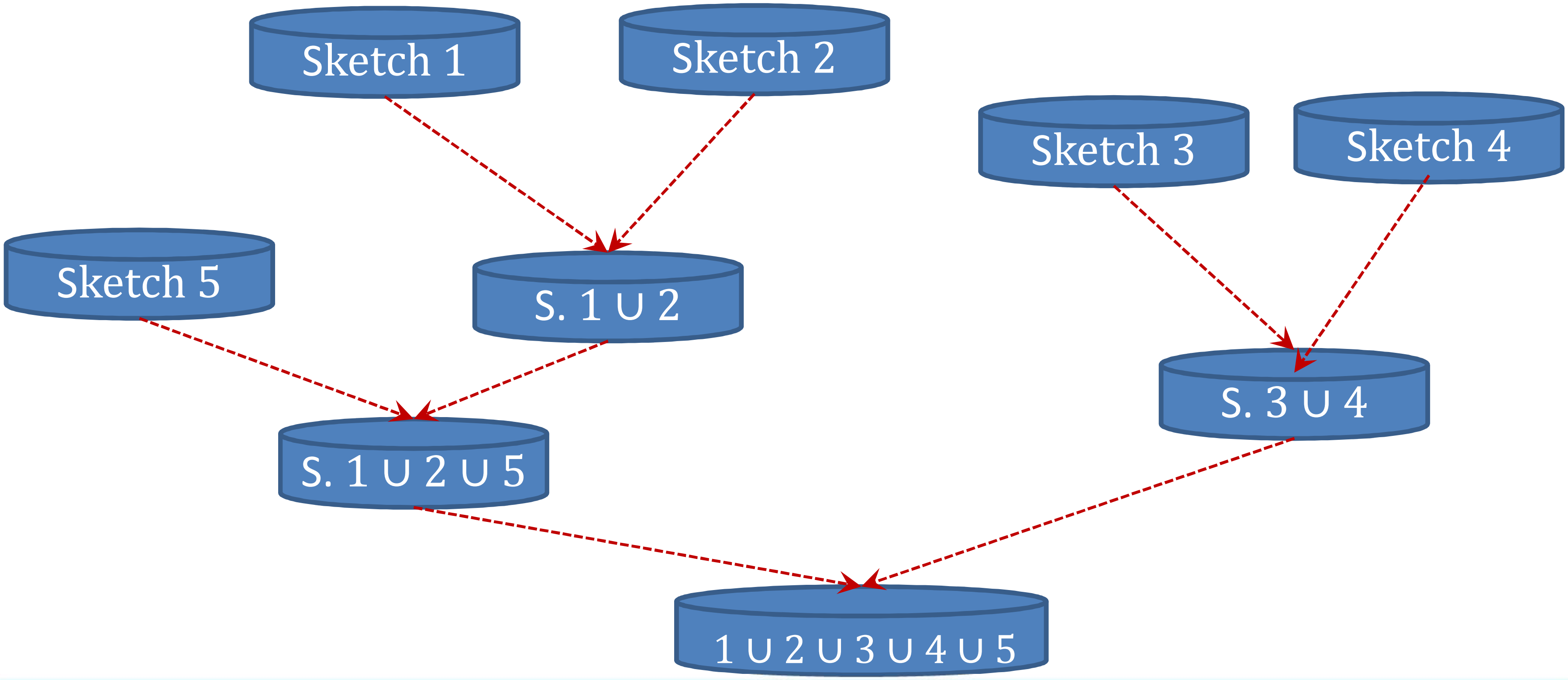


Beyond space-accuracy tradeoff: Mergeability of multiple sketches



Enough to consider merging two sketches

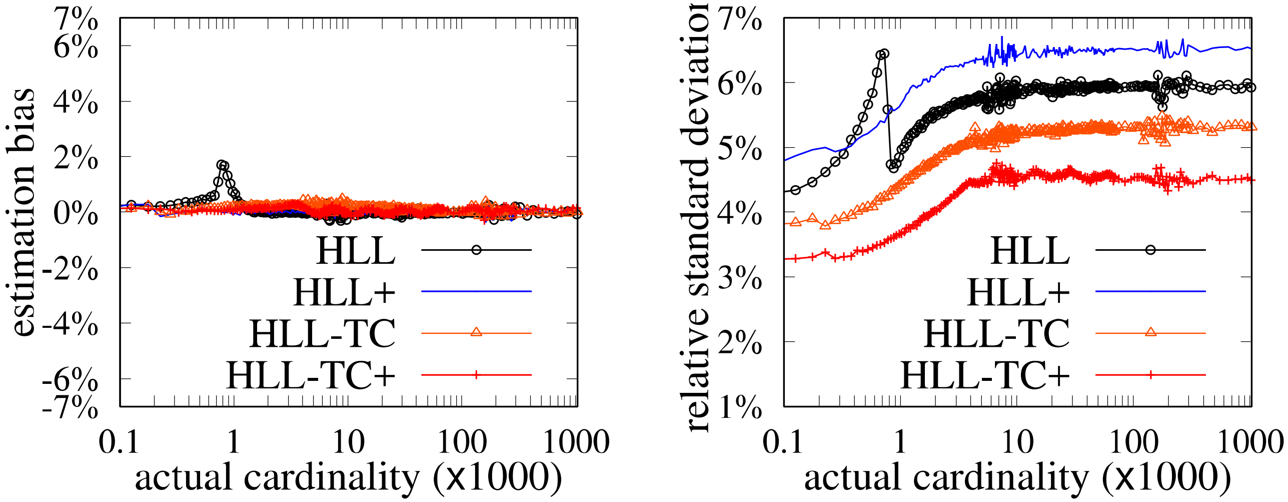
The side-effect of our HLL-TC+ algorithm



- Our HLL-TC is able to merge multiple sketches.
- But our HLL-TC+ may not support the sketch merging.
- Our HLL-TC+ more fits low-end devices that desire the highest memory-efficiency and does not need mergability.

Simulation Result on Space-Accuracy Efficiency

Coarse accuracy comparison given the same memory

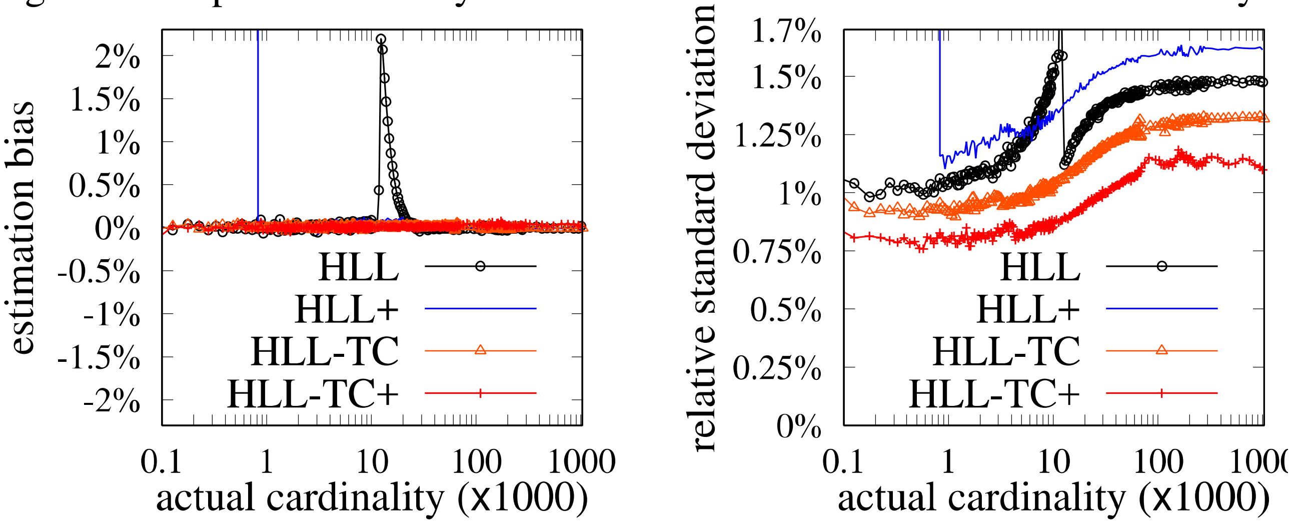


(a) Estimation bias

(b) Standard deviation

Fig. 5. Compare cardinality estimators with the same 1.54k bits memory.

Fine accuracy comparison given the same memory



(a) Estimation bias

(b) Standard deviation

Fig. 6. Compare cardinality estimators with the same 24.58k bits memory.

Summary

- Propose two new cardinality estimation algorithms, HLL-TailCut and HLL-TailCut+, and upload source code
  - https://www.dropbox.com/s/l0eaexhzvi34x9u/HLLPlus.zip
- Improve the space-accuracy tradeoff of HyperLogLog
  - HLL-TailCut needs 20% less memory at the same accuracy
  - HLL-TailCut needs 45% less memory at the same accuracy
- Address the small biased region problem of HyperLogLog
- Extend the effective operating range of HyperLogLog from Giga-scale data streams to Peta-scale or even Tera-scale data streams
- HLL-TailCut can support the merging of multiple sketches

IEEE International Conference on Computer Communications
1-4 May 2017 // Atlanta, GA, USA

Question and Answer