40th IEEE International Conference on Data Engineering 2024
Utrecht, Netherlands | 13th - 17th May

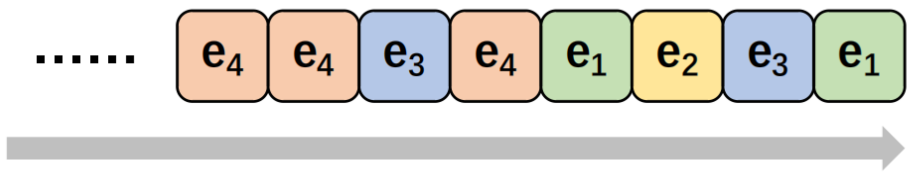
BitMatcher: Bit-level Counter Adjustment for Sketches

Qilong Shi, Chengjun Jia, Wenjun Li*, Zaoxing Liu, Tong Yang,
Jianan Ji, Gaogang Xie, Weizhe Zhang, and Minlan Yu

2024/5/17

Background

Data stream model



a data stream

$$f_1 = 2$$
$$f_2 = 1$$
$$f_3 = 2$$
$$f_4 = 3$$

Dedicated Platforms:
Network Switches
FPGA/ASIC

Query Results

- Frequency Estimation
- Heavy Hitter Detection
- Heavy Change Detection
- Item Size Distribution
- Entropy Estimation
- ……

Approximate Algorithms

in data stream processing

Exact & nearly-exact solutions

- Idea: Store all items in the stream and build many indexes.
- Weakness: Not practical for dedicated soft/hardware platforms.
  - Huge data volume (GBs): up to **billions** of items (network packets) in the 1-second time window.
  - Small memory size (<30 MB): FPGA, ASIC and Switches.

Approximate algorithms (Sketch)

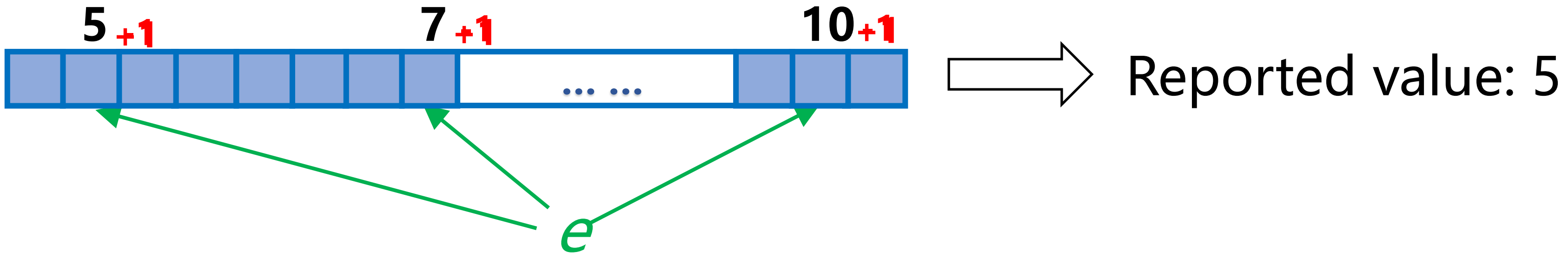
- memory efficient & tolerable errors
- Including: CM sketch, Bloom filter and many kinds of sketches……

Related Work

Common sketch

Prior art --- CM Sketch

Insertion: when a new item e comes
Query: query for the frequency of the item e
Deletion: delete item e

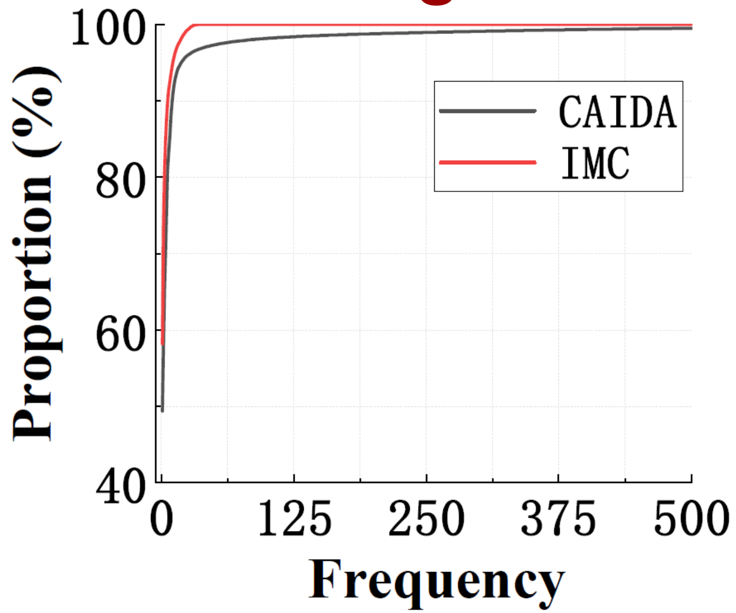


Related Work

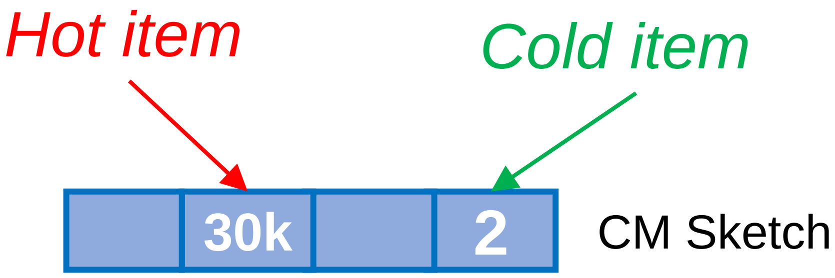
Common sketch

Real data streams

High-skewness!



Fixed-size counter



We have to set a large enough counter (i.e., 32bit)

Memory waste!

Related Work

Various improved sketches

Fixed-size counter

CM-sketch
CU-sketch
Space-saving
……

To better accommodate both hot & cold items

Self-adjusting

DHS
SALSA
……

Hierarchical

Elastic sketch
Augmented sketch
Pyramid sketch
……

Related Work

Hierarchical

Augmented Sketch

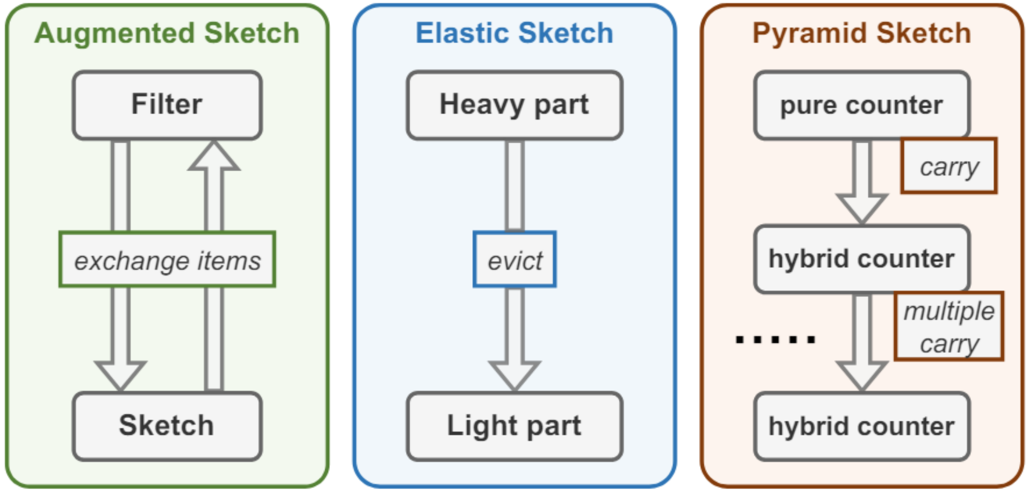
Pros: hot items always in the filter.
Cons: exchange greatly reduce speed.

Elastic Sketch

Pros: No exchange 👉 very high speed.
Cons: hot item may be accidentally expelled.

Pyramid Sketch

Pros: Automatically handle overflow.
Cons: access multiple layers for hot items 👉 unsuitable for tasks with hot items.



Hierarchical sketch outline

Related Work

Self-adjusting

Prior art --- **SALSA**

Pros

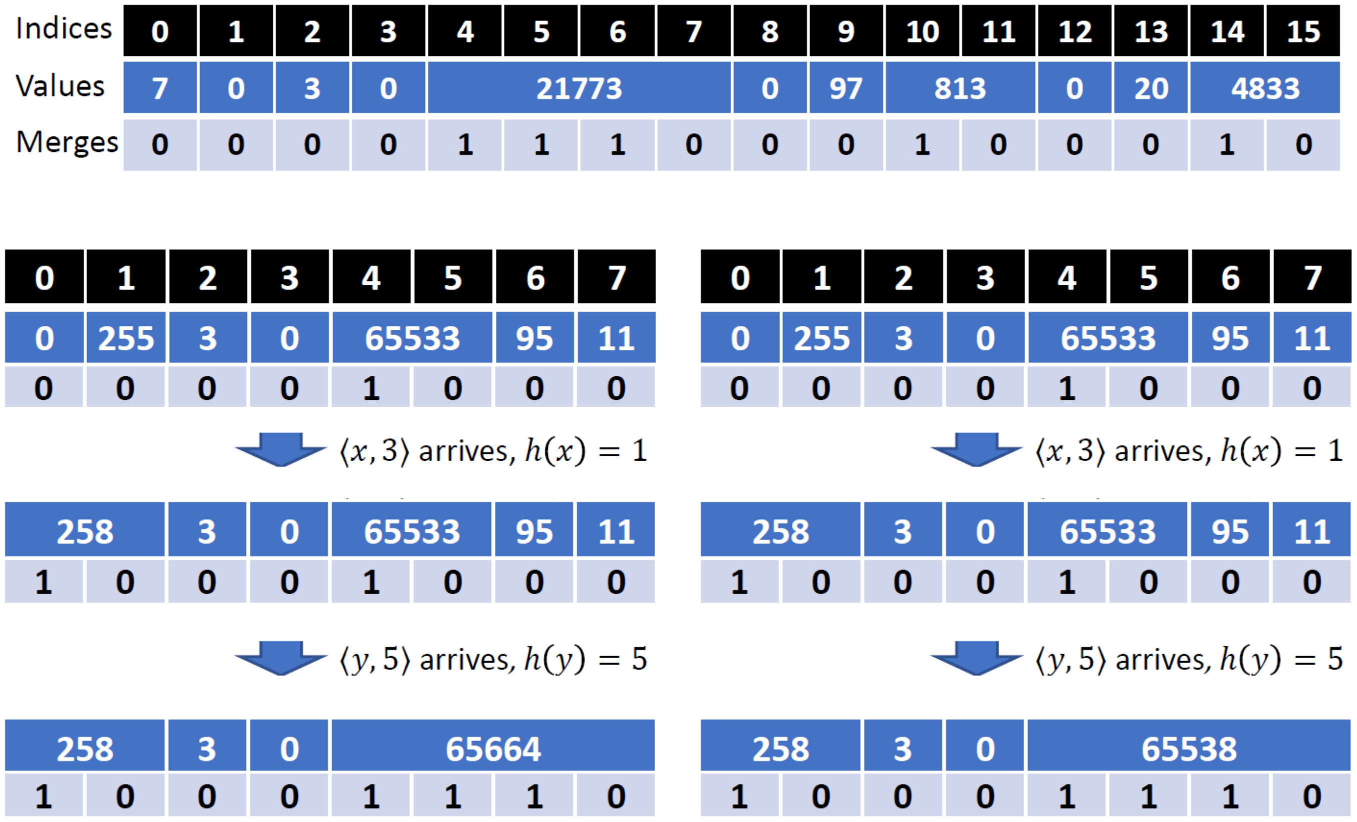
Finer segmentation inside the counter

👉 high accuracy

Cons

additional bitmaps&complex operations

👉 reduce speed



(a) Sum merging of counters (b) Max merging of counters

Related Work

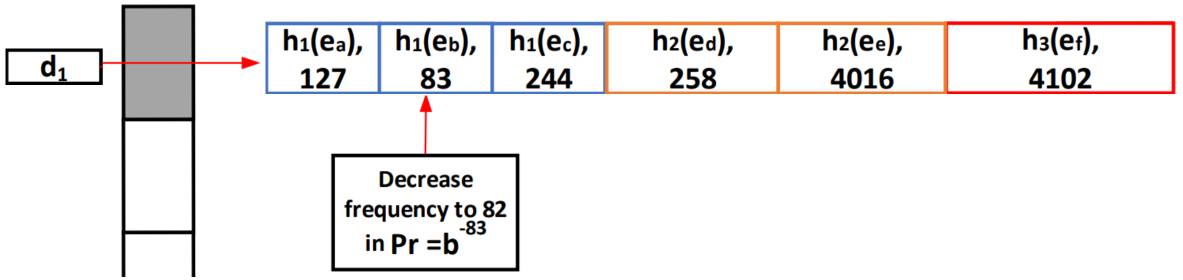
Self-adjusting

Prior art --- **DHS (Dynamic Hierarchical Sketch)**

Pros

Adjustments are limited to a single bucket.

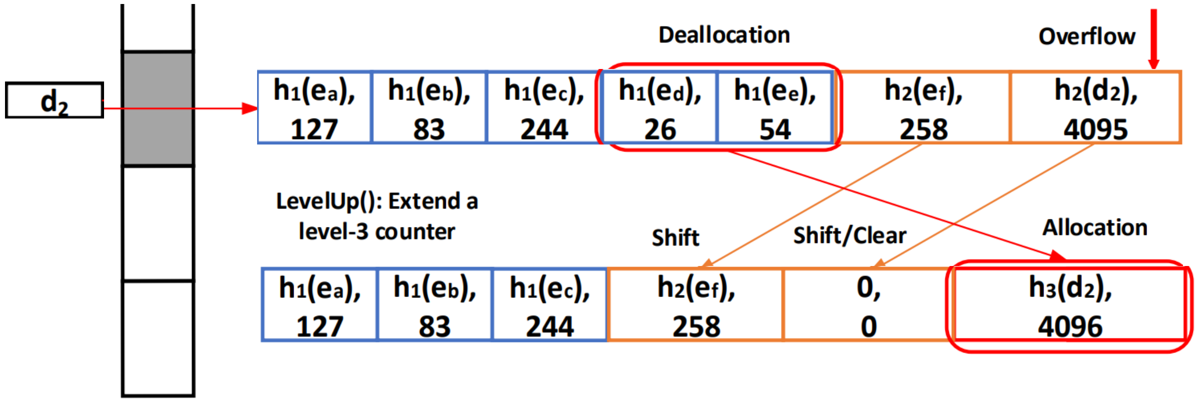
👉 high accuracy and speed



Cons

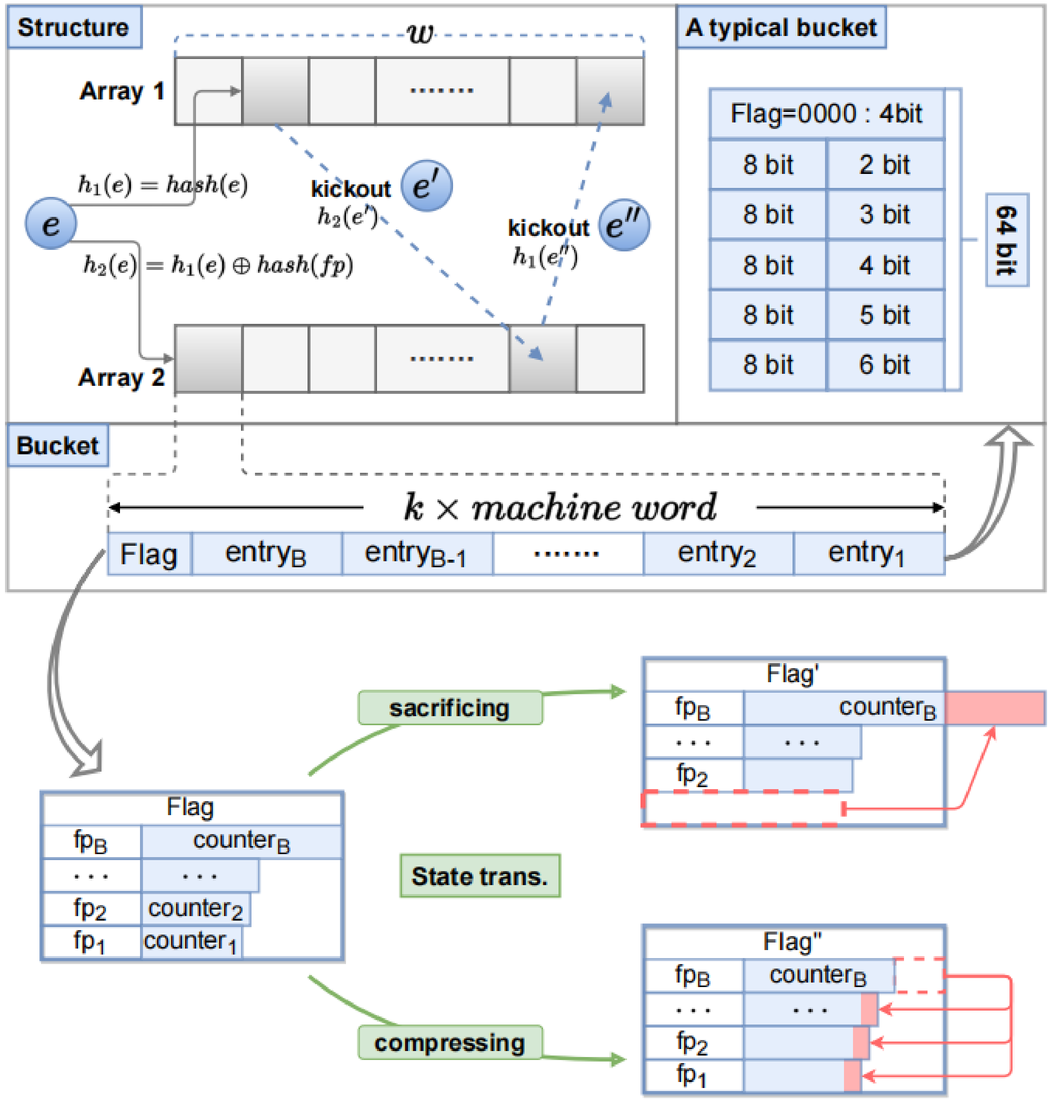
The adjusting strategy is limited to three types of counters: 8/12/16 bits.

👉 can`t store when the data traffic is heavy.
👉 too large adjustment granularity.



BitMatcher Framework

Data Structure

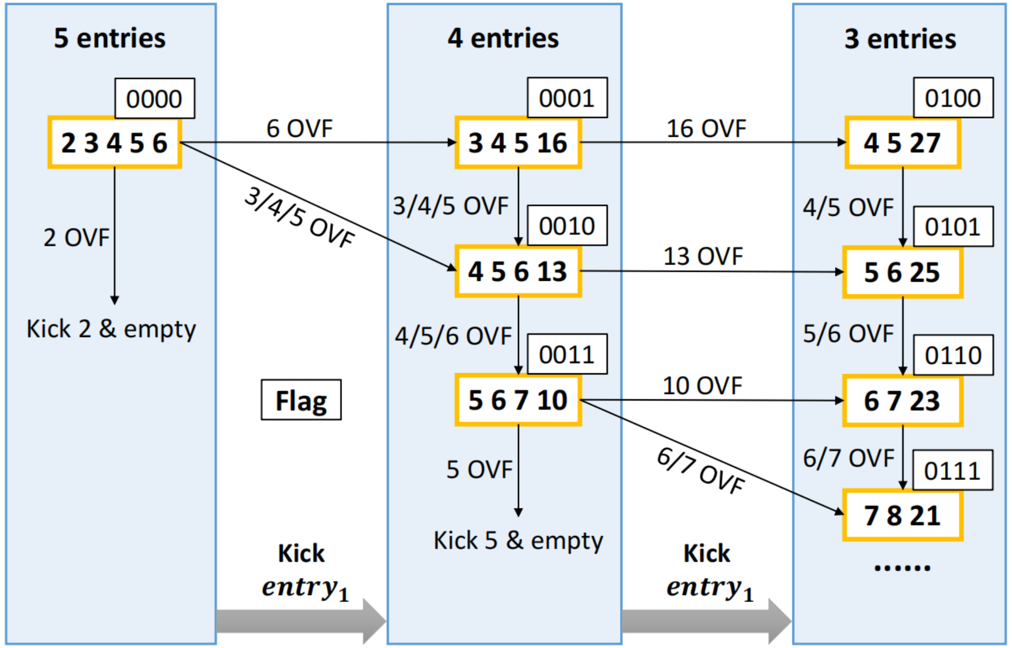


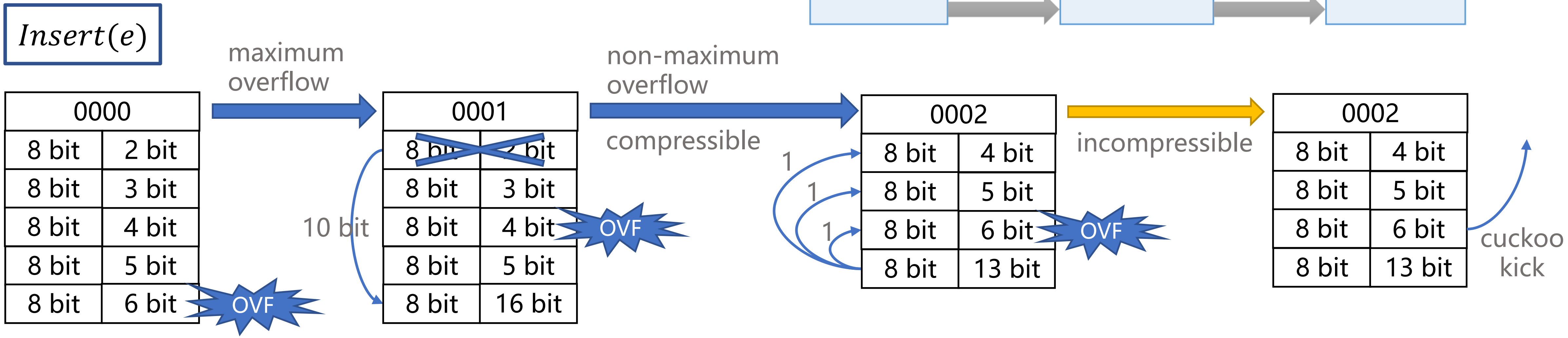
BitMatcher Framework

State transition table

BM: 64-bit bucket & small memory

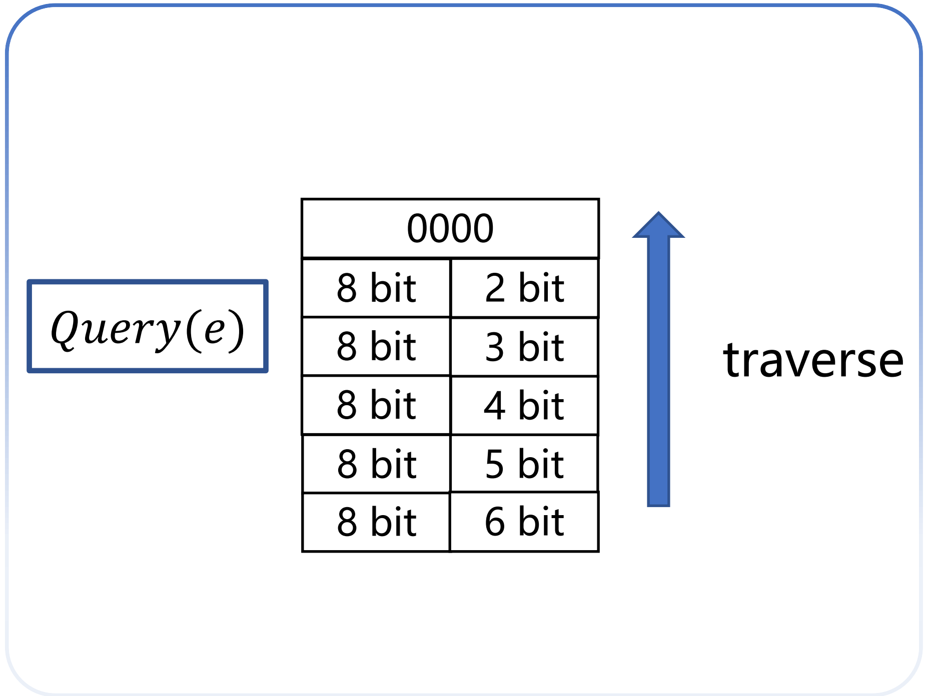
DHS: bucket size $\neq 64k$ bits





BitMatcher Framework

Design ideas



``**Cuckoo kick**`` are used to balance the load among buckets.
👉 **``Global coordination``**

Decode with the ``flag bits`` in the bucket.
👉 **High processing speed**

Accurate to 1-bit space allocation.
👉 **High accuracy and memory saving**

Experimental Results

Settings

Platform

CPU

Software

FPGA

Hardware

Datasets

CAIDA

(Network traffic):
2.49M items
max_freq=17K

IMC

(Network traffic):
19.86M items
max_freq=0.69M

Zipf

(Synthetic):
32M items
max_freq=123K~18.1M

Experimental Results

Settings

Compared Algorithm

CM sketch (CM)
Nitro sketch (NI)

Fixed-size counter

Augmented sketch (AS)
Pyramid sketch (PCU)
Elastic sketch (EL)

Hierarchical

Dynamic hierarchical sketch (DHS)
SALSA

Self-adjusting

Measurement tasks

- Frequency Estimation
- Heavy Hitter Detection
- Heavy Change Detection
- Item size distribution
- Entropy Estimation:

$\sum_{e_i \in E} p_i \log_2 \frac{1}{p_i}$, where p_i is $\frac{f_i}{N}$ (probability of occurrence of e_i).

Experimental Results

Settings

Metrics

- **AAE** = $\frac{1}{|E|}\sum_{(e_i \in E)} |f_i - f_i'|$, where f_i & f_i' are real & estimated frequency of e_i
- **ARE** = $\frac{1}{|E|}\sum_{(e_i \in E)} \frac{|f_i - f_i'|}{f_i}$
- $\boldsymbol{F_1 - score} = \frac{2 \times PR \times RR}{PR + RR}$, PR is Precision Rate, RR is Recall Rate

HH&HC detection

- **WMRE** (weighted mean relative error) = $\frac{\sum_{i=1}^{z} |n_i - n_i'|}{\sum_{i=1}^{z} (\frac{n_i + n_i'}{2})}$, n_i & n_i' are the real & estimated numbers of items with frequency= i

Item size distribution

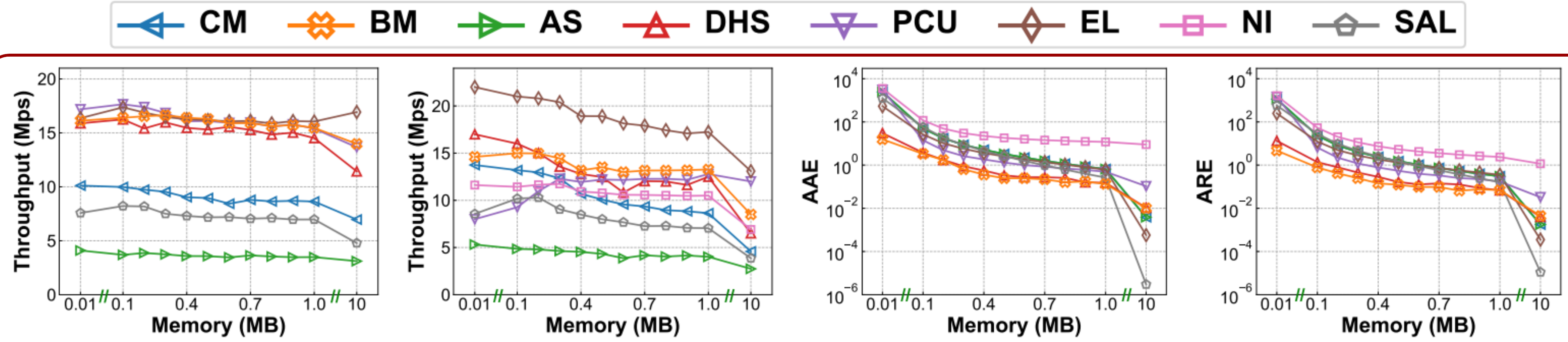
- **RE** (relative error) = $\frac{|True - Estimate|}{True}$

Entropy estimation

Experimental Results

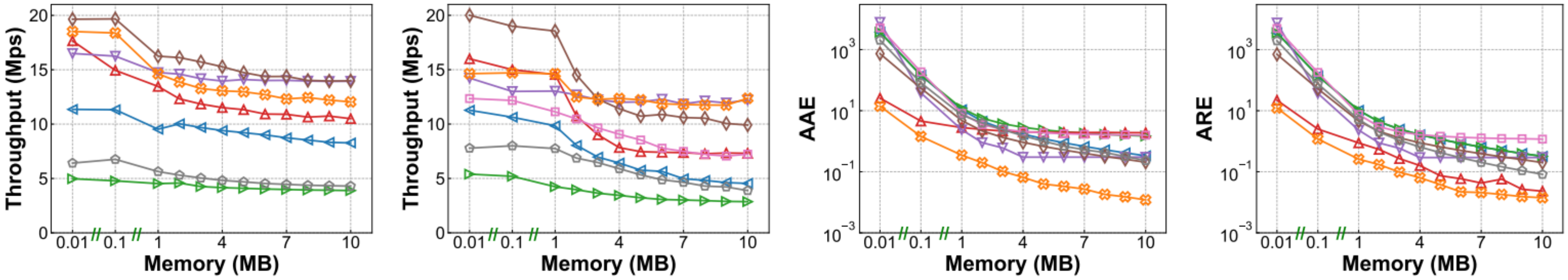
Frequency Estimation

CAIDA



(a) Insert Throughput (b) Query Throughput (c) AAE (d) ARE

IMC

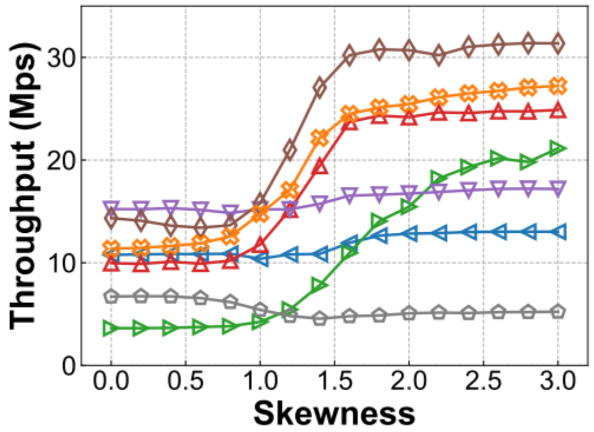


(a) Insert Throughput (b) Query Throughput (c) AAE (d) ARE

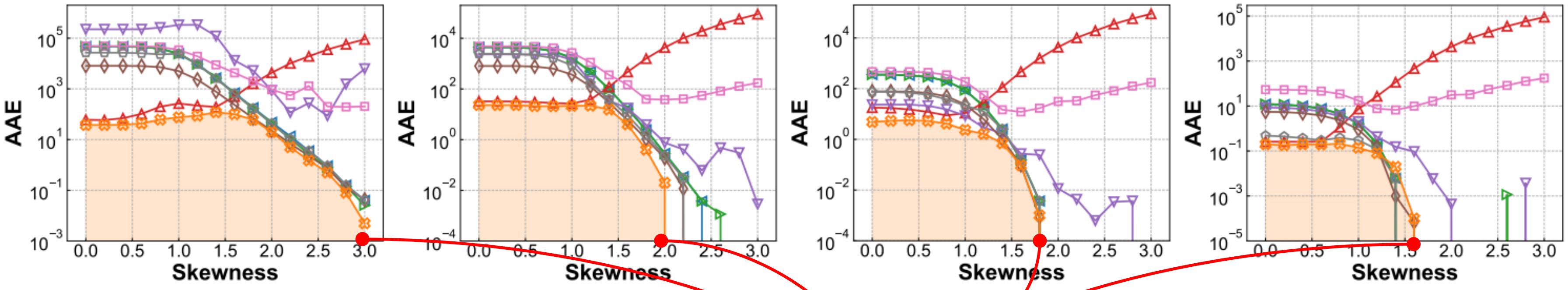
Experimental Results

Frequency Estimation

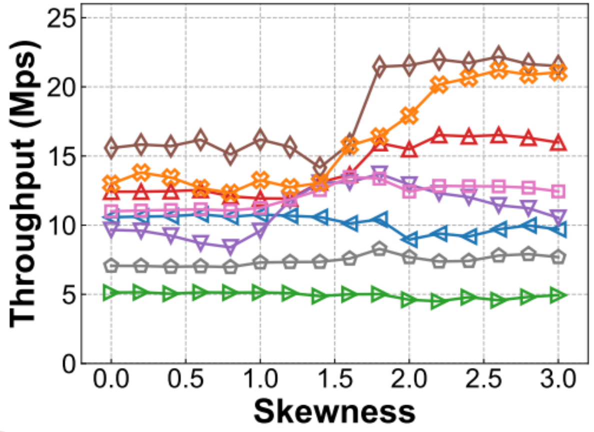
ZIPF



(a) Insert Throughput



(b) AAE - 0.01 MB (c) AAE - 0.1 MB (d) AAE - 1.0 MB (e) AAE - 10.0 MB



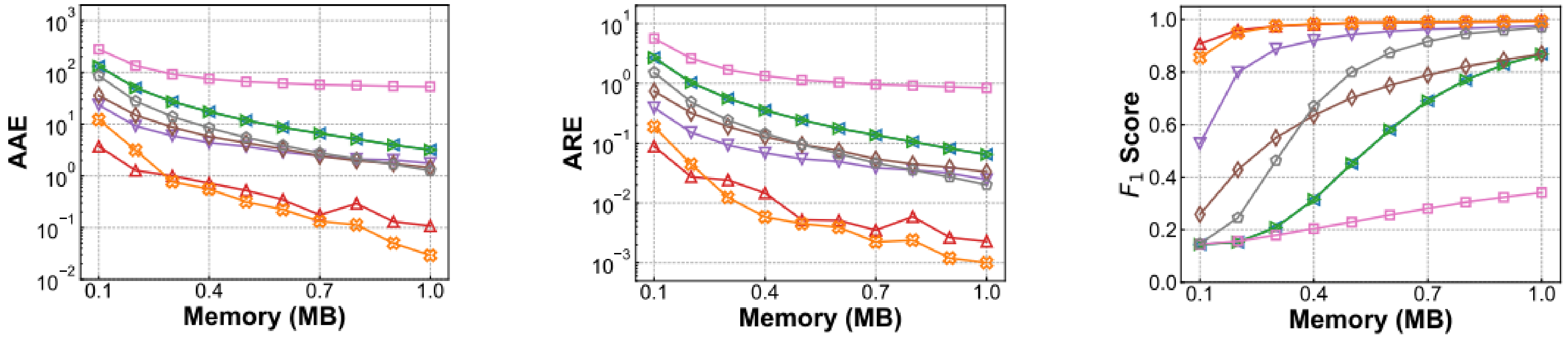
Key point

	0.01 MB	0.1 MB	1 MB	10 MB
BM	√	2.0 √	1.8 √	1.6
EL	——	2.3	1.8	1.6
SALSA	——	2.4	2.0	1.5
AS	——	2.7	2.0	1.4
CM	——	2.6	2.0	1.4

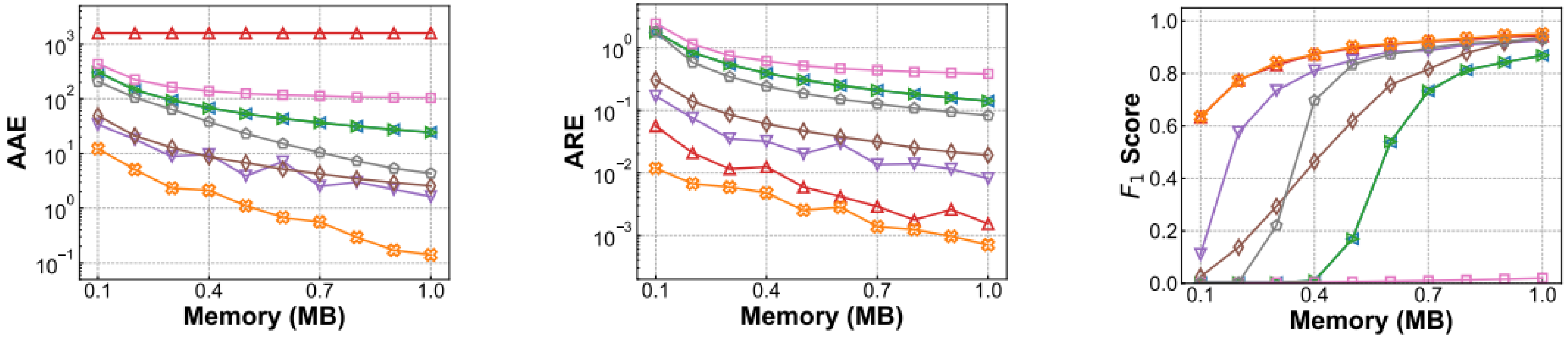
Experimental Results

Heavy **Hitter** Detection

CAIDA



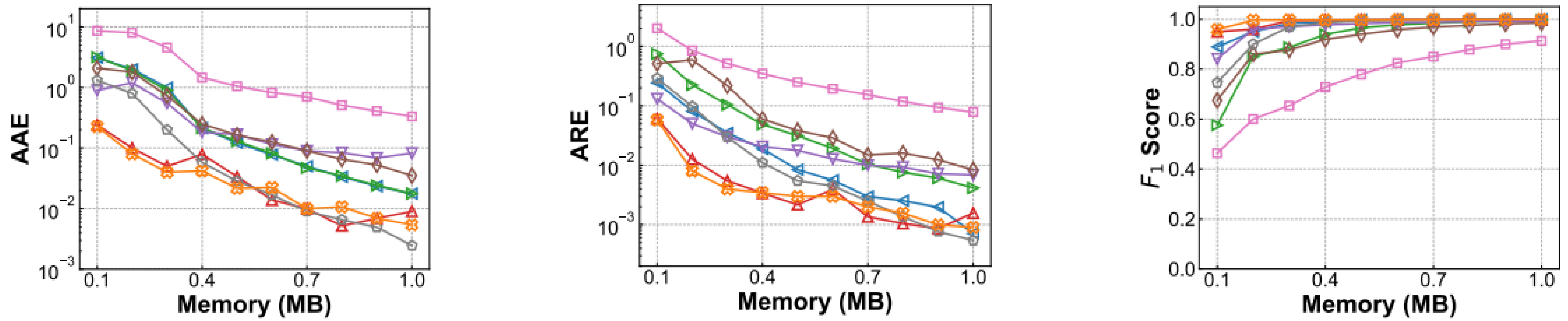
IMC



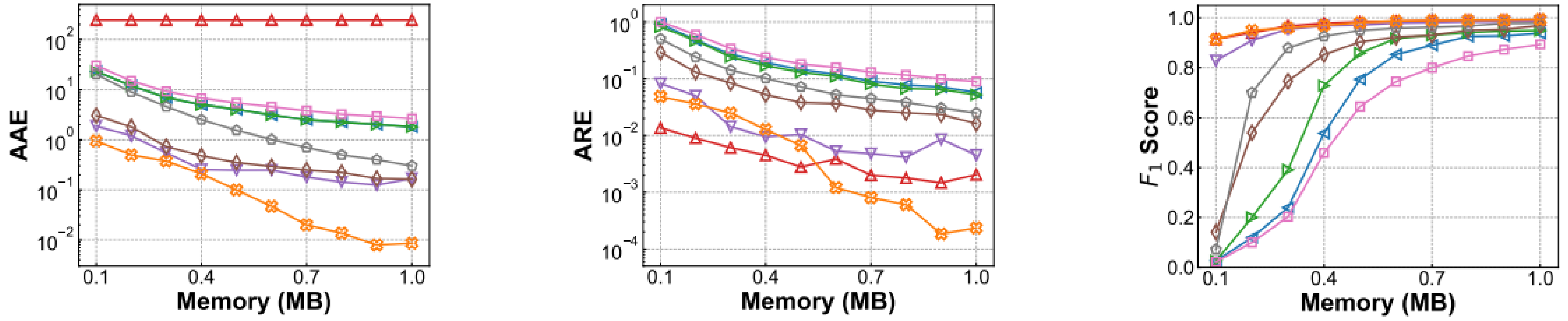
Experimental Results

Heavy **Change** Detection

CAIDA

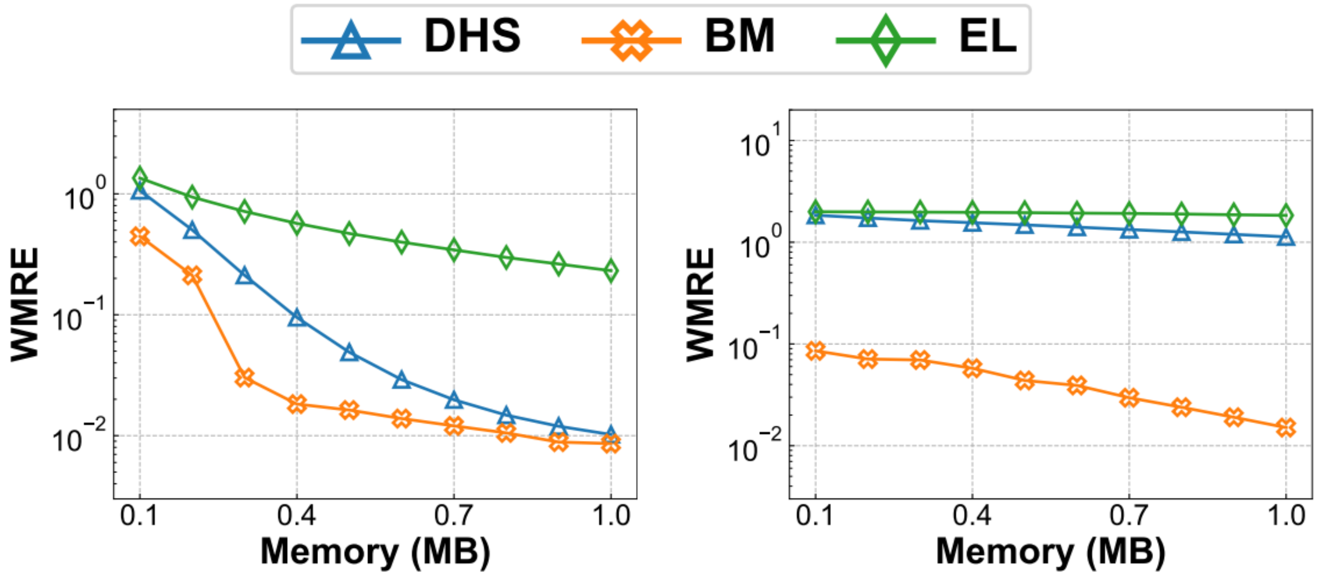


IMC



Experimental Results

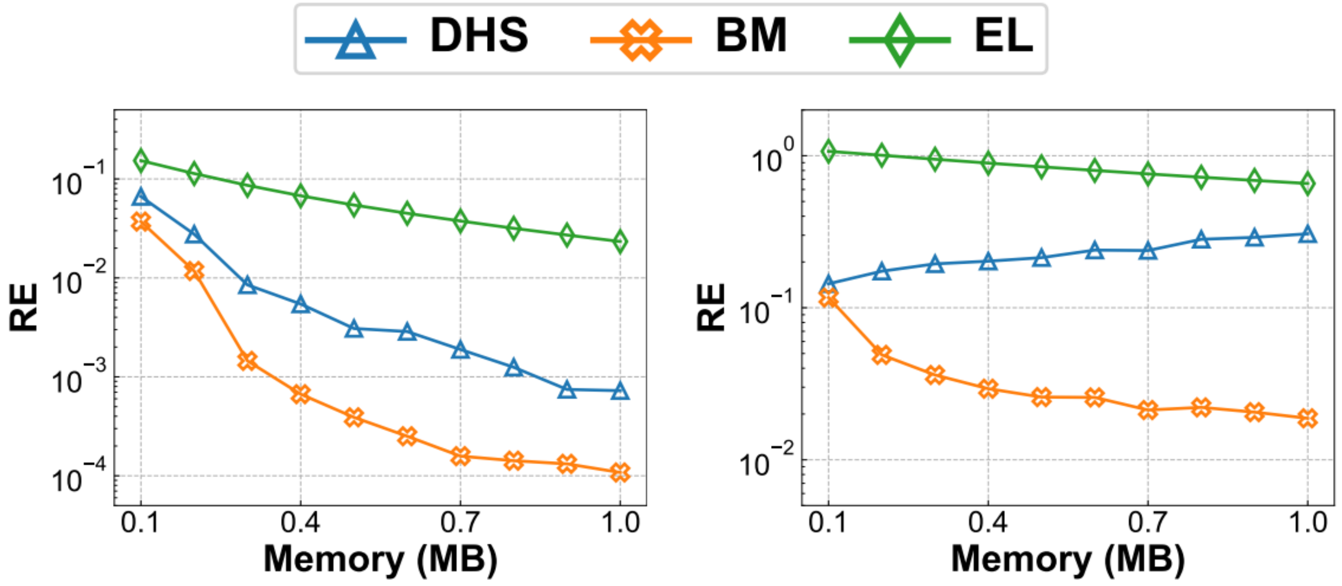
Item size distribution



(a) Common dataset (CAIDA)　(b) Large dataset (IMC)

Experimental Results

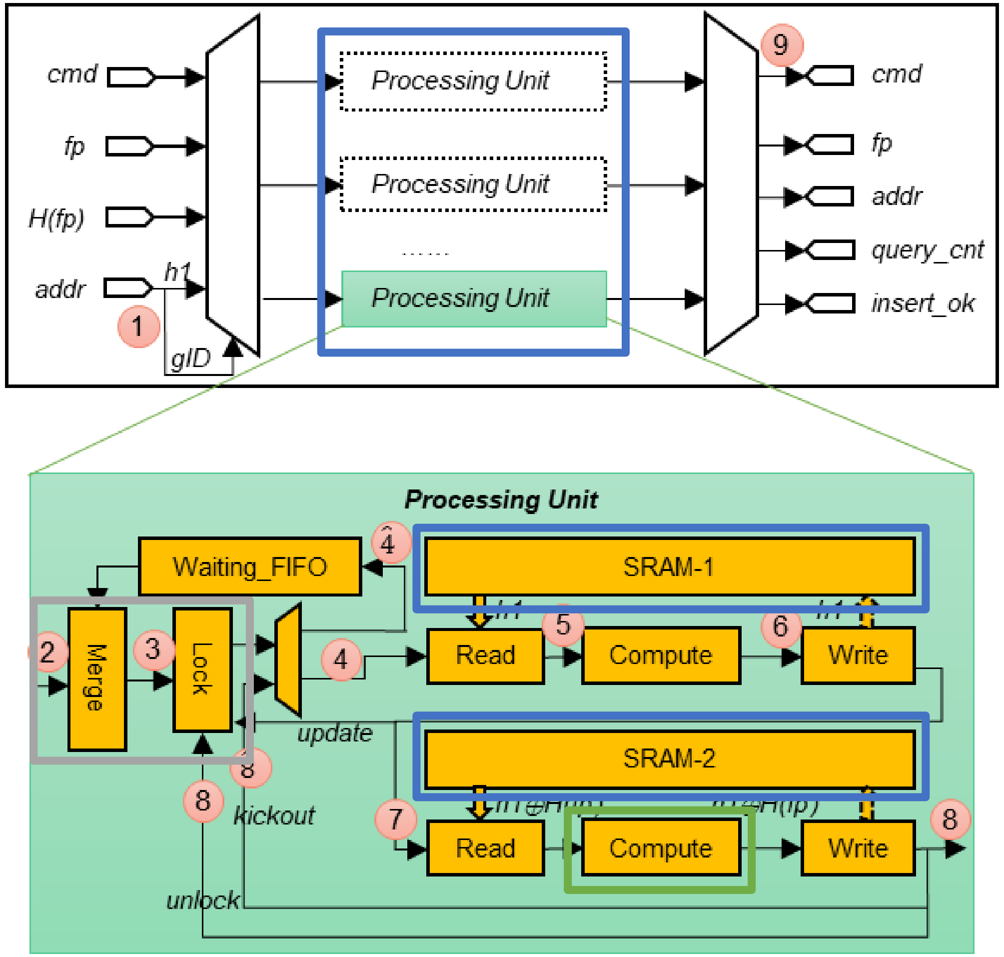
Entropy Estimation



(a) Common dataset (CAIDA) (b) Large dataset (IMC)

FPGA Implementation

Results



Multi-engine

Parallelism

Pipeline

Solve the dependency

Bitmatcher can achieve **192Mpps** at most with **3%** FPGA resources.

Algorithms	Logics	RAM	Max Frequency
Elastic Sketch	2,939	1,978,368 bits	162.6 MHz
BitMatcher	11,639	1,216,512 bits	192.3 MHz

Conclusion

1. BitMatcher: a bit-level counter adjustment that can perfectly match the data stream distribution.

2. Small memory cost, high speed, high accuracy, and good soft / hardware scalability.

3. We use BitMatcher to process five typical measurement tasks.

4. We implemented BitMatcher on CPU and FPGA. All codes are released at Github.

Thank you!
Q&A