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
Background
Data stream model

Dedicated Platforms:
Network Switches
FPGA/ASIC

Query Results

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

....... e4 e4 e3 e4 e1 e2 e3 e1

a data stream

\[ f_1 = 2 \]
\[ f_2 = 1 \]
\[ f_3 = 2 \]
\[ f_4 = 3 \]
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 \( e \)

Reported value: 5
We have to set a large enough counter (i.e., 32bit)
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.
Related Work
Self-adjusting

Prior art --- SALSA

**Pros**
- Finer segmentation inside the counter
- High accuracy

**Cons**
- Additional bitmaps & complex operations
- Reduce speed

![Diagram](image-url)

- (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 ≠ 64k bits

**Insert(e)**
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)
- Augmented sketch (AS)
- Pyramid sketch (PCU)
- Elastic sketch (EL)
- Dynamic hierarchical sketch (DHS)
- SALSA

**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** = $\frac{1}{|E|} \sum_{(e_i \in E)} \frac{|f_i - f'_i|}{f_i}$

- **$F_1$ score** = $\frac{2 \times PR \times RR}{PR + RR}$, $PR$ is Precision Rate, $RR$ is Recall Rate

- **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 $= i$

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

Frequency Estimation

(a) Insert Throughput
(b) Query Throughput
(c) AAE
(d) ARE
### Experimental Results

**Frequency Estimation**

#### Key point

<table>
<thead>
<tr>
<th></th>
<th>0.01 MB</th>
<th>0.1 MB</th>
<th>1 MB</th>
<th>10 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM</td>
<td>√</td>
<td>2.0 √</td>
<td>1.8 √</td>
<td>1.6</td>
</tr>
<tr>
<td>EL</td>
<td>—</td>
<td>2.3</td>
<td>1.8</td>
<td>1.6</td>
</tr>
<tr>
<td>SALSA</td>
<td>—</td>
<td>2.4</td>
<td>2.0</td>
<td>1.5</td>
</tr>
<tr>
<td>AS</td>
<td>—</td>
<td>2.7</td>
<td>2.0</td>
<td>1.4</td>
</tr>
<tr>
<td>CM</td>
<td>—</td>
<td>2.6</td>
<td>2.0</td>
<td>1.4</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Logics</th>
<th>RAM</th>
<th>Max Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elastic Sketch</td>
<td>2,939</td>
<td>1,978,368 bits</td>
<td>162.6 MHz</td>
</tr>
<tr>
<td>BitMatcher</td>
<td>11,639</td>
<td>1,216,512 bits</td>
<td>192.3 MHz</td>
</tr>
</tbody>
</table>
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