A Comparison of Performance and Accuracy of Measurement Algorithms in Software

Omid Alipourfard, Masoud Moshref\textsuperscript{1}, Yang Zhou\textsuperscript{2}, Tong Yang\textsuperscript{2}, Minlan Yu\textsuperscript{3}

Yale University, Barefoot Networks\textsuperscript{1}, Peking University\textsuperscript{2}, Harvard University\textsuperscript{3}
Network function virtualization is trending

Data centers
Use cloud to manage cloud

Edge networks
Mini-clouds at the edge

Firewall
Load balancer
CDN
WAN opt.
NAT
Network function virtualization is trending

Data centers
Use cloud to manage cloud

Edge networks
Mini-clouds at the edge

Virtualization, dynamic scale-out, fast iterations ...
Network function virtualization is trending

Measurement

Firewall  Load balancer  CDN  WAN opt.  NAT

Control loop
Measurement algorithms come with many implementations

<table>
<thead>
<tr>
<th>Measurement Task</th>
<th>Tree/Heap</th>
<th>Sketch</th>
<th>Hash table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavy hitter</td>
<td>ANCS ’11, ICDT’ 05</td>
<td>NSDI’ 13, SIGCOMM’ 17</td>
<td>SIGCOMM’ 02</td>
</tr>
<tr>
<td>Super spreader</td>
<td></td>
<td>SIGCOMM’ 17, PODS’ 05</td>
<td>IMC’ 10, NDSS’ 05</td>
</tr>
<tr>
<td>Flow size distrib.</td>
<td></td>
<td>SIGMETRICS’ 04</td>
<td>IMC’ 10</td>
</tr>
<tr>
<td>Change detection</td>
<td>CoNEXT’ 13</td>
<td>TON’ 07</td>
<td>IMC’ 10</td>
</tr>
<tr>
<td>Entropy estimation</td>
<td></td>
<td>COLT’ 11</td>
<td>SIGMETRICS’ 06</td>
</tr>
<tr>
<td>Quantiles</td>
<td>SIGMOD’ 01, 99, 13</td>
<td>Hot ICE’ 11</td>
<td></td>
</tr>
</tbody>
</table>
Measurement algorithms come with many implementations

<table>
<thead>
<tr>
<th>Measurement Task</th>
<th>Tree/Heap</th>
<th>Sketch</th>
<th>Hash table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavy hitter</td>
<td>ANCS ’11, ICDT’ 05</td>
<td>NSDI’ 13, SIGCOMM’ 17</td>
<td>SIGCOMM’ 02</td>
</tr>
</tbody>
</table>

Which algorithm works best for NFs running on software ...

<table>
<thead>
<tr>
<th></th>
<th>Tree/Heap</th>
<th>Sketch</th>
<th>Hash table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entropy estimation</td>
<td>SIGMOD’ 01, 99, 13</td>
<td>COLT’ 11</td>
<td>SIGMETRICS’ 06</td>
</tr>
<tr>
<td>Quantiles</td>
<td>Hot ICE’ 11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Design concerns for software switches

<table>
<thead>
<tr>
<th>Domain</th>
<th>Hardware switches</th>
<th>Software switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>Limited memory size</td>
<td></td>
</tr>
<tr>
<td>Objective</td>
<td><em>Fit in memory</em></td>
<td></td>
</tr>
<tr>
<td>Opportunity</td>
<td>Deterministic throughput</td>
<td></td>
</tr>
</tbody>
</table>
Design concerns for software switches

<table>
<thead>
<tr>
<th>Domain</th>
<th>Hardware switches</th>
<th>Software switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>Limited memory size</td>
<td>Limited cache size</td>
</tr>
<tr>
<td>Objective</td>
<td><em>Fit in memory</em></td>
<td><em>Maximize throughput</em></td>
</tr>
<tr>
<td>Opportunity</td>
<td>Deterministic throughput</td>
<td>Large memory (hierarchical)</td>
</tr>
</tbody>
</table>
### Closer look at heavy hitter detection

Find the most popular items (flows) in a packet stream.

<table>
<thead>
<tr>
<th>Hash table based</th>
<th>Count sketch</th>
<th>Heap based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update the entry (e) in the hash table.</td>
<td>Hash the header n times and update relevant entries (e_s).</td>
<td>Keep a heap of counters.</td>
</tr>
<tr>
<td>Report if e &gt; threshold.</td>
<td>Report if min(e_s) &gt; threshold.</td>
<td>Replace the smallest counter if no space available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Report if entries &gt; threshold.</td>
</tr>
</tbody>
</table>
Closer look at heavy hitter detection

Find the most popular items (flows) in a packet stream.

**Hash table based**
- Update the entry \((e)\) in the hash table.
- Report if \(e >\) threshold.

**Count sketch**
- Hash the header \(n\) times and update relevant entries \((e)\).
- Report if \(e >\) threshold.

**Heap based**
- Keep a heap of counters.
- Replace the smallest counter if no space available.
- Report if number of entries > threshold.

What hash table works best?
Cuckoo vs. linear hash table

Two popular hash tables: Cuckoo hash table and Linear hash table.

Linear hash table

Cuckoo hash table
Evaluation settings

Settings

- DPDK Framework
- Intel Xeon-E5 2650 v3, 10G NIC
- CAIDA (1.4 mil flows, 40 mil pkts, 64B pkts)
- Zero packet loss test - RFC 2544
- Reporting interval 100ms ~ control loop frequency

Metrics

- Performance: average packet processing time
- We also measure precision/recall in the paper
Linear hashing outperforms Cuckoo hashing

- **Performance**: Linear table is 10~30% faster than Cuckoo table.

Why?

- **Computation**: Two hashes (Cuckoo) vs one hash (Linear).
- **Random access**: Two for Cuckoo vs. one for Linear.

Different from the database world - Memory is not an issue!

- Make the table large so collisions are rare!
Cuckoo vs. linear hash table

Two popular hash tables: Cuckoo hash table and Linear hash table.

Takeaways

- Use the least # of computations and random memory accesses.
- If you can, use large memory to reduce your computations.

● Memory is not an issue! Make the table large so collisions are rare.
## Comparison of algorithm classes

<table>
<thead>
<tr>
<th>Hash table based</th>
<th>Count sketch</th>
<th>Heap based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update the entry (e) in the hash table.</td>
<td>Hash the header n times and update relevant entries (e_s).</td>
<td>Keep a heap of counters.</td>
</tr>
<tr>
<td>Report if e &gt; threshold.</td>
<td>Report if min(e_s) &gt; threshold.</td>
<td>Replace the smallest counter if no space avail.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Report if entries &gt; threshold.</td>
</tr>
</tbody>
</table>
# Comparison of algorithm classes

<table>
<thead>
<tr>
<th>Hash table based</th>
<th>Count sketch</th>
<th>Heap based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear hash table</td>
<td>Count sketch with one hash (Count-array)</td>
<td>Heap + Linear hash table</td>
</tr>
</tbody>
</table>
## Simplest data structure works best

<table>
<thead>
<tr>
<th>Hash table based</th>
<th>Count sketch</th>
<th>Heap based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear hash table</td>
<td>Count sketch with one hash (Count-array)</td>
<td>Heap + Linear hash table</td>
</tr>
</tbody>
</table>

### Results

- Count array is the fastest.
- Hash table performance converges to count-array with larger tables.
- Heap based algorithms are slow because of random memory access.
How general are the results?

- Other measurement tasks
- Other traffic skews
- Amount of data kept per packet/flow
- Shared vs. separate data structure
Results hold for other measurement tasks

Change detection

- Computationally heavy
- Model flow’s traffic
- Report flows outside model’s predictions

Superspreader detection

- Memory heavy
- Update a bloom filter per packet

Does CPU behave differently dealing with other measurement task types?
Superspreaders: Count-array is the fastest
Superspreaders: Count-array is the fastest

96% Precision
Superspreaders: Count-array is the fastest

The trend is similar for change detection:
Fastest Count-array with Linear hash table a close second.
Impact of traffic skew on latency

Concerns

- Working set gets larger with lower skew.
- More items read in cache per packet batch.
Impact of traffic skew on latency

Concerns

- Working set gets larger with lower skew.
- More items read in cache per packet batch.

Observations

- Perf. degradation depends on the # of memory accesses per pkt.
- Count-array and linear hash table still the fastest.
Impact of bytes kept per flow on latency

Concerns

- Less number of items fit in the cache.
- Traverse multiple cache lines on a miss.
Impact of bytes kept per flow on latency

Concerns

- Less number of items fit in the cache.
- Traverse multiple cache lines on a miss.

Observations

- 1.9x higher latency - 4 bytes (70ns~) to 60 bytes (130ns~)
- **Solution**: Separate keys and values in the hash table.
  - 1.16x higher latency - 4 byte (90ns~) to 60 byte (105ns~)
Impact of shared/separate data-structure

**Shared**: Easy to report measured results.
- More cache bouncing between cores.

**Separate**: Merging to report is difficult.
- No cache bouncing between cores.
Impact of shared/separate data-structure

Observations

Sharing is expensive.
- Cache bouncing causes L3 latency for most memory accesses.
- Does not scale to many cores.

Merging is cheap.
- Very low memory bandwidth (even at 10ms reporting intervals).
Conclusions

Measurement in software servers is different than hardware:

- Use more memory to do less computation.
- Reduce data pulled into the cache per packet.

Calls for new:

- **Algorithms**, e.g., “sketch” over computation not memory.
- **Data structures**, e.g., seq. access pattern to match the CPU arch.
Thanks!

The code and benchmarks are available at:

https://github.com/SiGe/measure-pkt
Simplest data structure works best

<table>
<thead>
<tr>
<th>Hash table based</th>
<th>Count sketch</th>
<th>Heap based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear hash table</td>
<td>Count sketch with one hash</td>
<td>Heap + Linear hash table</td>
</tr>
</tbody>
</table>

Least amount of computation wins.

- Count array is the fastest.
- Hash table performance converges to count-array with larger tables.
- Heap based algorithms are slow because of random memory access.
Change-detection: Count-array is the fastest
Change-detection: Count-array is the fastest
Change-detection: Count-array is the fastest

Large # of heapify ops.

Deep heaps

Latency (ns)

MB
Impact of traffic skew on latency

![Graph showing the impact of traffic skew on latency for different storage structures (Count array, Heap, Linear). The x-axis represents the traffic dist (Zipf param), and the y-axis represents the 99th Latency (ns). The graph illustrates a decreasing trend in latency as the traffic dist increases, with different storage structures showing varying performance.](image-url)
Impact of traffic skew on latency

Prefetching can mask the memory access latency.

Traffic dist (Zipf param)
Impact of traffic skew on latency

More uniform packet count makes it more likely that heapify traverses multiple levels.
Bytes fetched impacts the performance
Mask the latency by keeping the values away
Impact of other apps. on measurement

- **Cache exhaustion**: working set not fitting in memory.
- **Memory BW exhaustion**: higher latency to fetch data.
Impact of other apps. on measurement

- Cuckoo-1 entry
- Cuckoo-4 entries
- Linear
- Count array
Impact of other apps. on measurement