ElasticBF: Elastic Bloom Filter with Hotness Awareness for Boosting Read Performance in Large Key-Value Stores

Yongkun Li, Chengjin Tian, Fan Guo, Cheng Li, Yinlong Xu
University of Science and Technology of China
USENIX ATC 2019
Background

- Key-value (KV) store has become an important storage engine for many applications
  - Cloud storage
  - Social networks
  - NewSQL database

- Examples of KV stores
  - Hbase @ Apache
  - LevelDB @ Google
  - RocksDB @ Facebook
  - ...
The most common design of KV stores is based on LSM-tree (log structured merge tree).

Design Highlights
Layering
Log-structured writes
Sorted in each level

Compaction with merge sort
The most common design of KV stores is based on LSM-tree (log structured merge tree). Level 0, Level 1, Level 2, ..., Level 6 are search order. Read Amplification!!! Key lookup: Check SSTables from lower levels to higher levels, one from each level (sorted). Bloom Filters improve read performance (also cached in memory).
Limitation of Bloom Filters

Bloom filters suffer from false positive rate

- False positive rate (FPR): \( 0.6185^b \) (b: Bits-per-key)

<table>
<thead>
<tr>
<th>Bits-per-key</th>
<th>2bits</th>
<th>3bits</th>
<th>4bits</th>
<th>5bits</th>
<th>6bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPR</td>
<td>38.3%</td>
<td>23.7%</td>
<td>14.6%</td>
<td>9.1%</td>
<td>5.6%</td>
</tr>
</tbody>
</table>

- How to reduce false positive rate?
  - Allocate more bits for each key
  - Incur large memory overhead (as Bloom filters are cached in memory)

Question: how to improve the Bloom filter design with limited memory cost?
Main Idea

Observation: unevenness of access frequencies
- Vary from different levels, SSTables, or even different regions within an SSTable

ElasticBF: Elastic Bloom filter management with locality awareness

Hot SSTables
- More bits/key
- Lower FPR

Cold SSTables
- Fewer bits/key
- Limited mem. usage
ElasticBF Design

**Challenge 1:** fixed data organization in SSTables limits BF adjustment

Bloom filters in SSTables are immutable.

**Fine-grained pre-allocation**

![Diagram of Bloom filter organization]

Rationale: **Separability** (Multiple filters have the same FPR as a single filter with the same bits-per-key, i.e., $\prod_{i=1}^{n} 0.6185^{b_i} = 0.6185^{b} (\sum_{i=1}^{n} b_i = b)$)
ElasticBF Design

**Challenge 2:** How to determine the most appropriate number of filter units for each SSTable and how to realize dynamical adjustment?

**Determine: Cost-benefit analysis**

Adjust Bloom filters only when the expected number of I/Os caused by false positive $E[\text{ExtraIO}]$ can be reduced.

$$E[\text{ExtraIO}] = \sum_{i=1}^{M} f_i \times r_i$$

- $M$: # of segments in the system
- $f_i$: access frequency of segment $i$
- $r_i$: false positive rate of the BF allocated for seg. $i$
ElasticBF Design

**Challenge 2**: How to **determine** the most appropriate number of filter units for each SSTable and how to **realize** dynamical adjustment?

**Adjust: in-memory multi-queue**

Multiple LRU queues to realize dynamical adjustment

- **Upgrade**: each time when a segment is accessed, move to the MRU side.
- **Downgrade**: search an “expired” segments from \( Q_n \) to \( Q_1 \) and move it to the next lower-level queue if \( E[Extra\_IO] \) can be reduced by releasing one filter unit.
ElasticBF Design

**Challenge 3**: Writes in mixed workloads may reset the hotness information (as compaction creates new SSTables)

**Hotness inheritance**
Estimate the hotness of new segments after compaction

1. Find out involved old segments
2. Estimate using the mean of the hotness of old segments
3. Enable an appropriate number of filter units based on the estimated hotness
Performance Evaluation

- We implement ElasticBF in various KV stores: LevelDB/RocksDB/PebblesDB

- **Experiment setting**
  - **Machine**
    
    | CPU                        | Mem/Disk                  | OS                               |
    |---------------------------|---------------------------|----------------------------------|
    | Dell PowerEdge R730       | 64GB RAM                  | Ubuntu 16.04 OS with             |
    | 12-cores Intel Xeon CPU E5| 500GB SSD and 1TB         | Linux 4.15 kernel                |
    | 2650 v4 with 2.20GHz      | 7200RPM HDD               |                                  |

  - **Micro-benchmarks: workloads generated by YCSB-C**

<table>
<thead>
<tr>
<th>Size of KV pair</th>
<th>Size of database</th>
<th>Request Distribution</th>
<th>Zipfian skew</th>
<th>Zero lookup/Non-zero lookup</th>
<th># of Get Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>100/400 GB</td>
<td>zipfian/uniform</td>
<td>0.99/1.1/1.2</td>
<td>1:1</td>
<td>10 million</td>
</tr>
</tbody>
</table>

- YCSB benchmarks (six core workloads)
How much improvement does ElasticBF achieve?

- Compare read performance w/ and w/o ElasticBF (10M GET requests)

ElasticBF increases the read throughput to >2x and reduces the latency by >50%, and also reduces the # of I/Os by ~60%.
Micro-benchmarks

- Mixed workload
  - Still remarkable improvement

- PUT and SCAN performance
  - Negligible impact
ElasticBF improves read throughput under read-dominant workloads (B: 95% read, C: 100% read, D: 95% read)
Comparison with Monkey

- Monkey: coarse-grained scheme (even BF allocation in each level) w/o dynamical adjustment

Micro-benchmark: 10M GET to 100GB KV store

ElasticBF further increases the throughput to $1.39 \times - 2.20 \times$
Comparison with Monkey

- Monkey: coarse-grained scheme (even BF allocation in each level) w/o dynamical adjustment

YCSB benchmark: 10M GET to 100GB KV store

ElasticBF further increases the throughput up to \(~2\times\) under read-dominant workloads with high skewness.
Impact of System Configurations

- Impact of:
  - Hard disk
  - Zero lookup ratio
  - Block cache size
  - KV pair size
  - Database size
  - Segment size
  - Filter unit size

- Please refer to our paper for detailed results
Conclusion

- LSM-tree based KV stores suffer from read amplification problem
  - Bloom filters reduce extra I/Os and improve read performance
  - Uniform Bloom filter design either suffers from high false positive rate or incurs large memory overhead

- We develop ElasticBF
  - An elastic scheme to dynamically adjust Bloom filters, so it improves read performance with limited memory
  - Orthogonal to the optimizations of the LSM-tree structure, so it can be deployed in various existing KV stores
Thanks for your attention!

For any questions, please feel free to contact
Prof. Yongkun Li@USTC

ykli@ustc.edu.cn
http://staff.ustc.edu.cn/~ykli/