Don’t Thrash: How to Cache Your Hash on Flash

Bender, Farach-Colton, Johnson, Kraner, Kuszmaul, Medjedovic, Montes, Shetty, Spillane, Zadok

Presented by Áron Ricardo Perez-Lopez

Approximate Sets

- Two operations: INSERT and MAY-CONTAIN (also DELETE in some variants)
- Can have false positives
- No false negatives
- Adjustable error rate: ϵ
- Much more space-efficient than sets based on hash tables
- Classical example: Bloom filter (1970)

Bloom Filter (BF)

- A bit-array B of length m
- k hash functions from U (the set of all elements) to $[0,m)$
- INSERT(B, x): set all bits $B[h_k(x)]$ to 1
- MAY-CONTAIN(B, x): return true if all bits $B[h_k(x)]$ are 1
- Yields an error rate of $\epsilon = \left(1 - e^{-nk/m}\right)^k$
- For optimal k and 1 byte per element, $\epsilon = 1.56\%$
- Can't delete (counting variant can with a 4x space overhead)

BF Variants for (Flash) Disk

Elevator Bloom Filter (EBF)

- Buffers insertions in memory.

Buffered Bloom Filter (BBF)

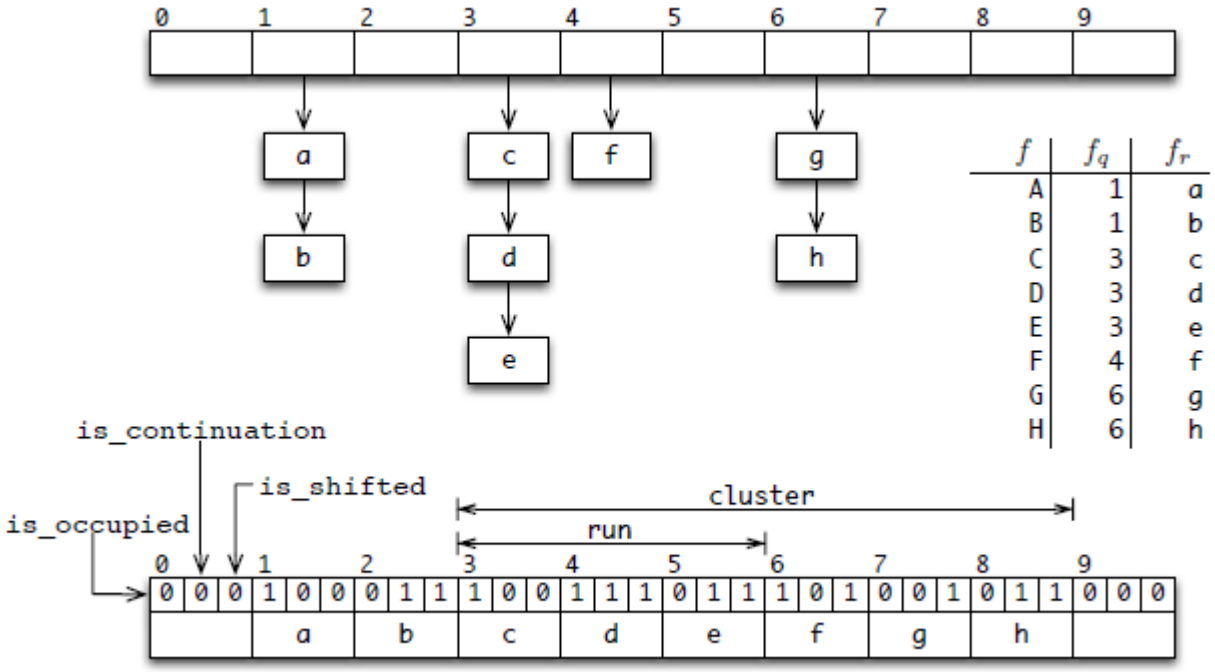
- Filter is split into blocks.
- Separate hash function is used to determine the block for a value.
- Uses in-memory buffers.

Forest Bloom Filter (FBF):

- Multiple layers of filters exponentially increasing in size.
- Uses two-level block structure (with two extra hash functions)
- Uses in-memory buffers.

The Quotient Filter (QF)

- Essentially a compact representation of a multiset
- Uses a single hash function, base 2 **integer division**
- Key is quotient (high bits) and value is remainder (low bits)
- Stores set elements consecutively with three extra bits
- Supports DELETE



QF variants

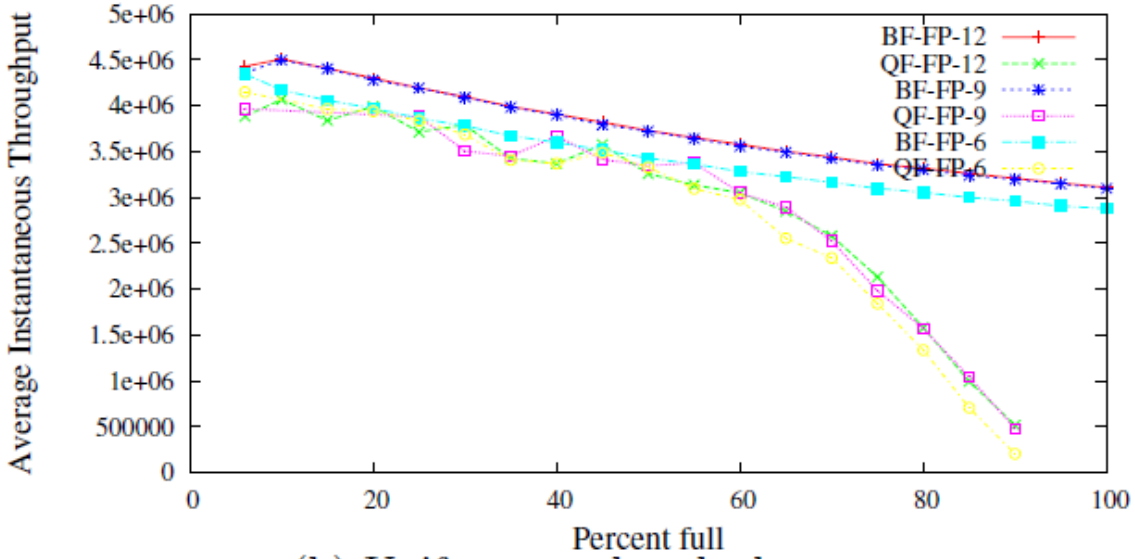
Buffered Quotient Filter (BQF)

- Uses two QFs, one in memory and one on disk.
- In-memory QF is flushed to disk when full.

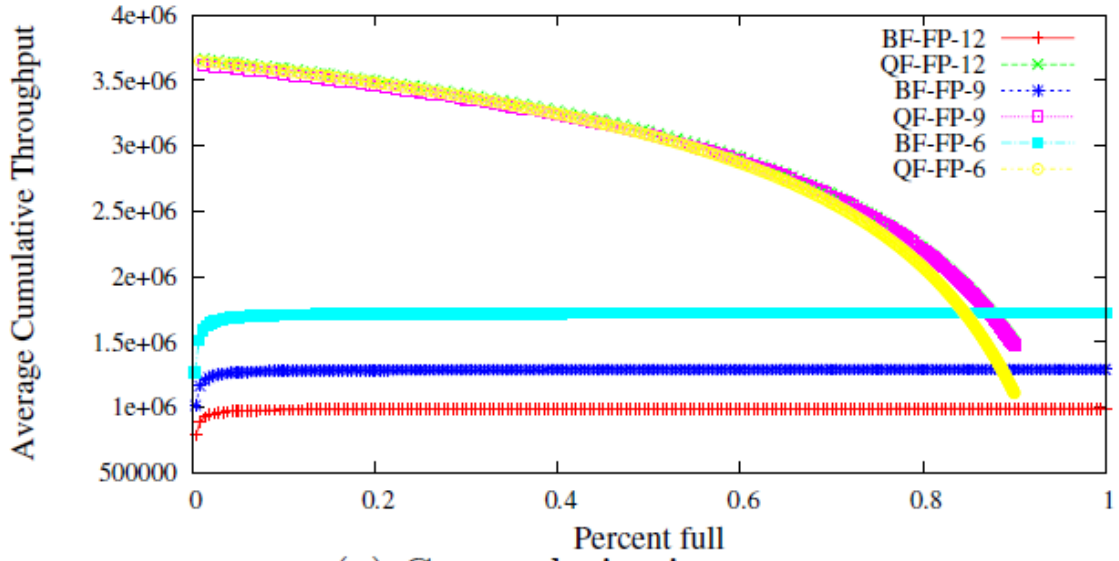
Cascade Filter (CF)

- Uses many QFs of exponentially increasing size.
- When in-memory one is full, all filters are merged into smallest level that can hold all values.
- Faster insertion but slower lookup than BQF.

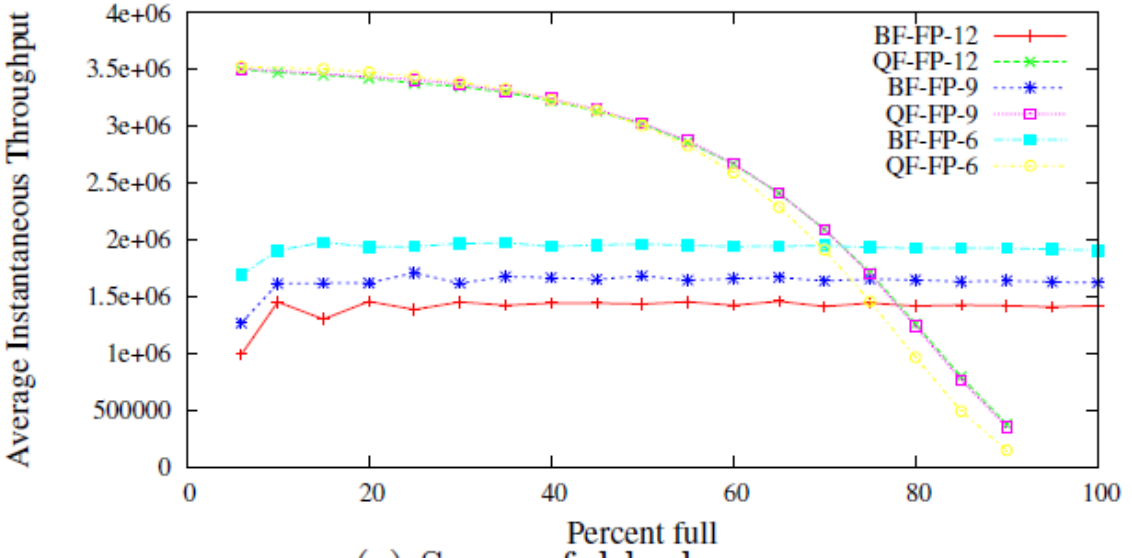
Performance in RAM



(b) Uniform random lookups

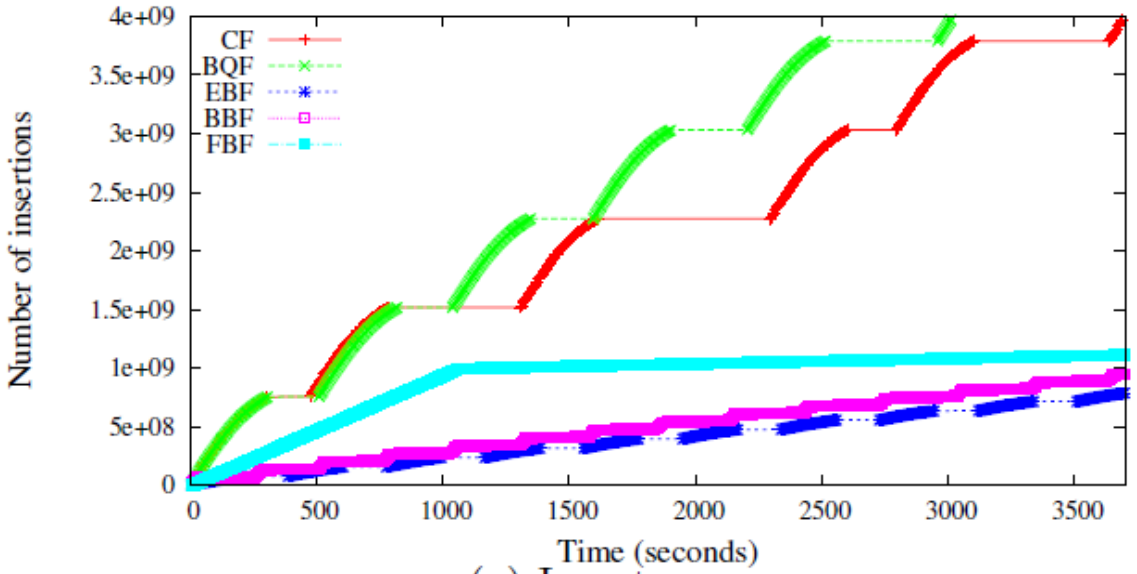


(a) Cummulative inserts

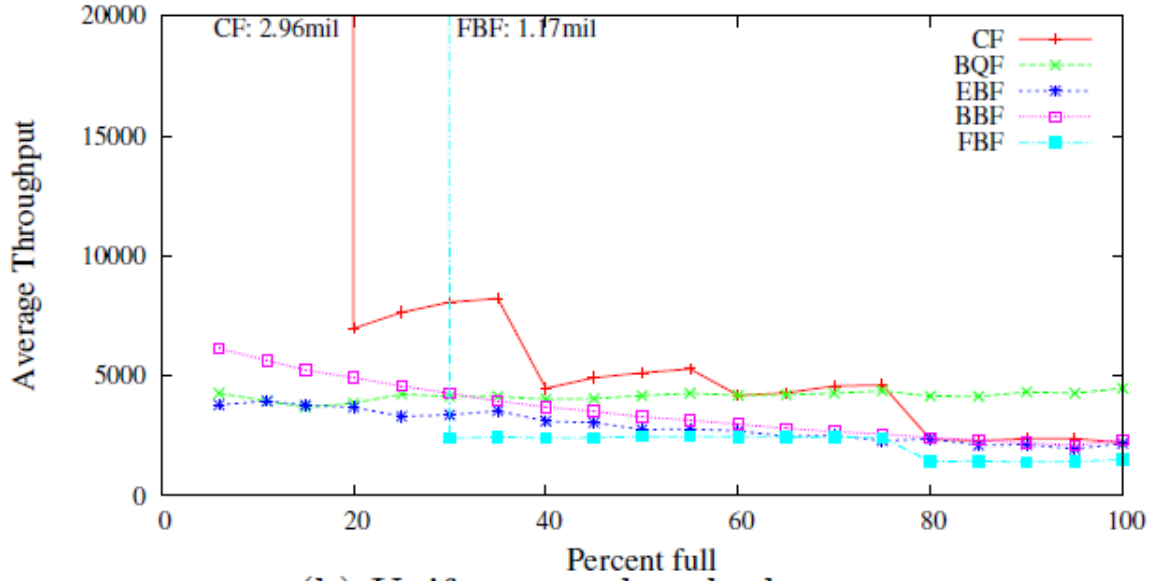


(c) Successful lookups

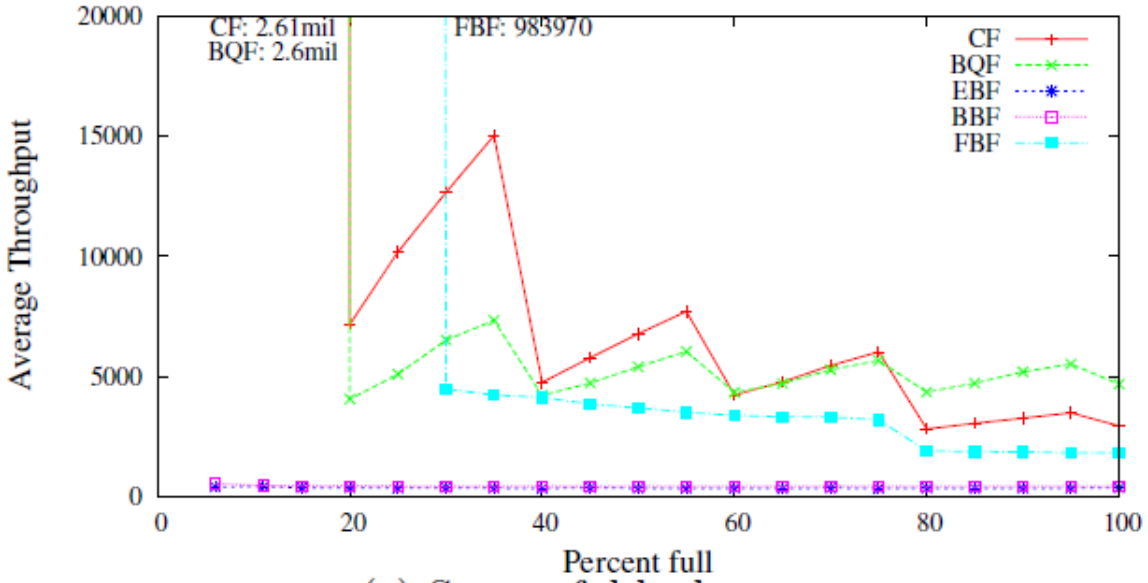
Performance on disk (small disk-to-RAM ratio)



(a) Inserts

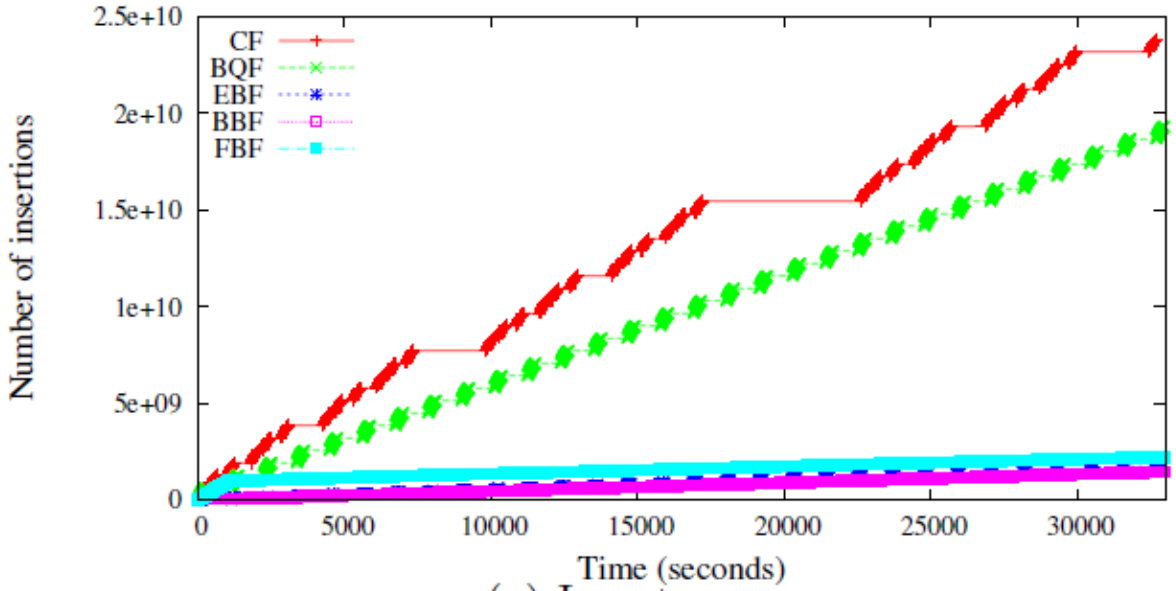


(b) Uniform random lookups

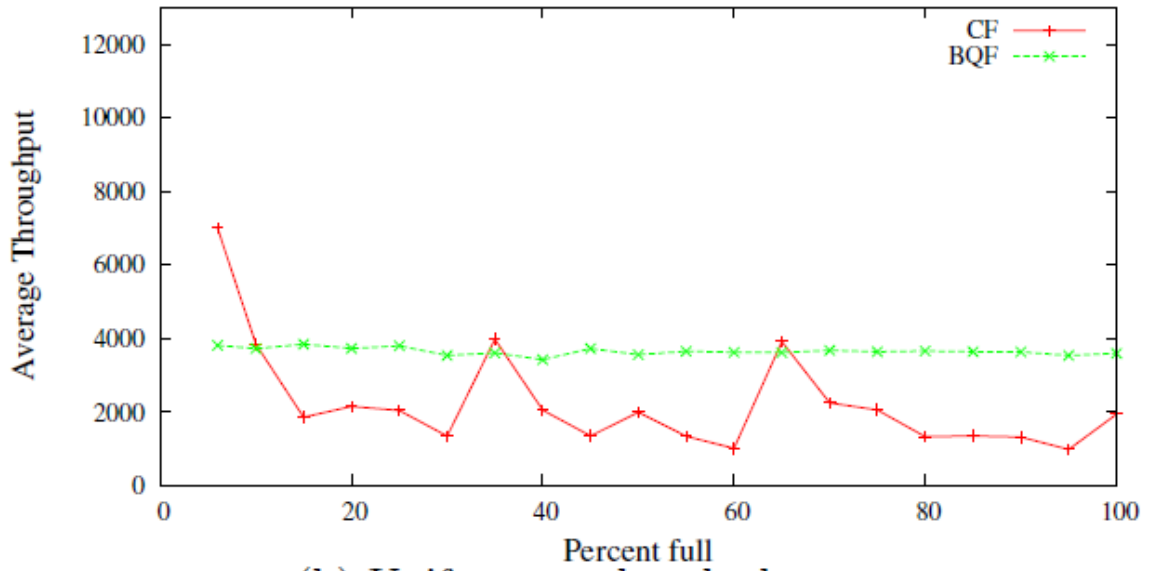


(c) Successful lookups

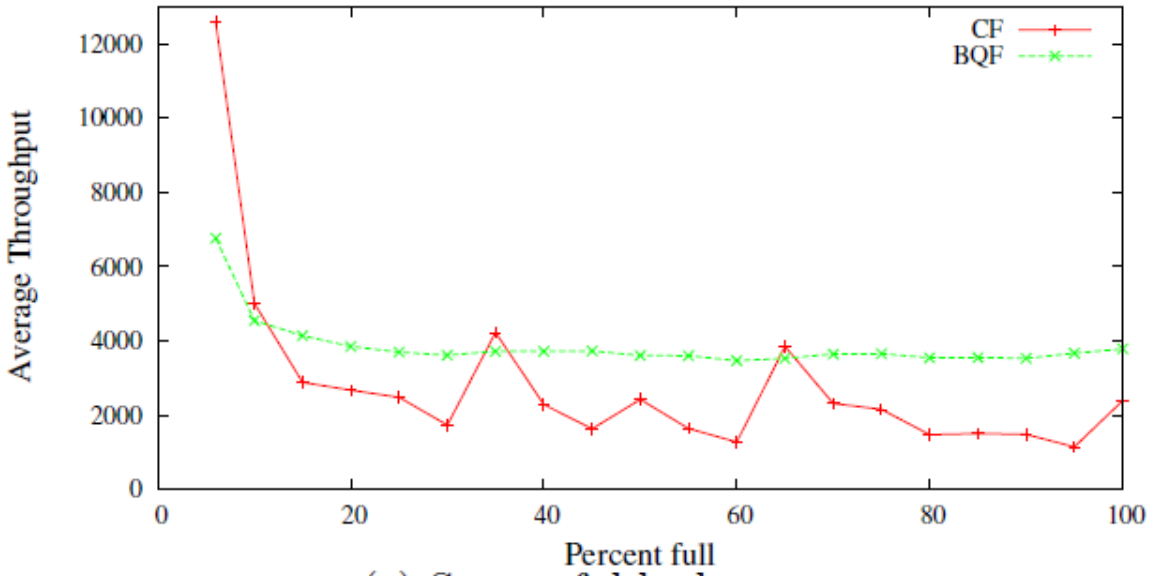
Performance on disk (large disk-to-RAM ratio)



(a) Inserts



(b) Uniform random lookups



(c) Successful lookups

Discussion Questions

- Are the achieved speedups convincing?
- Are insertions or lookups more frequent for real-world filters?
- What do you think about not reporting lookup throughput for the QF variants because “they were not able to complete the large experiment?”