Addressing the Space Overhead of Vector Quotient Filter

Chaeyoung Hwang Yongjin Kim Junhan Lee Youjip Won
Department of Electrical Engineering, KAIST, Korea

***Abstract*—Filters, also known as Approximate Membership Query data structures, show whether an item is present in a dataset. Filters are widely used in storage systems to avoid unnecessary I/O operations. In distributed database systems, effective memory management can be achieved by utilizing filters to reduce the I/O requests of non-volatile memory. Vector Quotient Filter is a filter that achieves high performance in uniform datasets. However, it has limitations in terms of space and performance when applied to skewed datasets. It does not count duplicate items and gives them equal space, leading to large space overhead. We examine how Vector Quotient Filter can overcome this space overhead issue in skewed datasets by applying a counting algorithm.**

***Index Terms*—Filter, Memory Usage, Skewed Dataset**

I. Introduction

Filters are probabilistic data structures that examine whether an item is in a dataset. They support insertion, lookup, and some also support removal. Bloom Filter [1], Quotient Filter [2], Cuckoo Filter [3], Morton Filter [4], Vector Quotient Filter [5] are well known filters. Filters are utilized in storage systems to lessen I/O operations [6]. It is usually stored in volatile memory and returns whether an item is in a dataset. In this way, the system avoids I/O operations of non-volatile memory for items that are not present in the dataset. Filters may return false positive answers on an item. This may cause unnecessary I/O operations on absent items, which degrades performance. Therefore, it is important to design filters with a low false positive rate and less space usage.

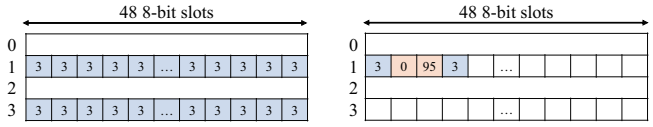
Among the well-known filters, Bloom Filter supports insertion and lookup but does not support removal. Quotient Filter, Cuckoo Filter, Morton Filter, and Vector Quotient Filter support insertion, lookup, and removal. However, all these filters lack support for counting duplicate items. Bloom Filter doesn't keep track of duplicate items. Quotient Filter, Cuckoo Filter, Morton Filter, and Vector Quotient Filter use equal amounts of space for each duplicate of items. Having a large number of duplicates can result in space overhead, leading to performance degradation. Since most real datasets follow a Zipfian distribution, keeping track of duplicate items can help conserve space.

To overcome this weakness, filters that track duplicates with a counting method have been proposed. Counting Bloom Filter [7] replaces each bit in the Bloom Filter with a counter. However, it uses additional space than the Bloom Filter by a factor equal to the size of the counter. Counting Quotient Filter [8] uses a counting algorithm that counts duplicates in a dataset. However, it has the same weakness as the Quotient Filter, which is severe performance degradation with high load factors. In this work, we examine the space overhead of the Vector Quotient Filter and propose a way to overcome it.

II. Model

Vector Quotient Filter [5] consists of multiple blocks, containing b logical buckets and s slots each storing an r-bit fingerprint. When inserting an item, it uses a hash function to generate a 64-bit hash value. The hash value is divided into an upper log$b$-bit bucket index and a lower r-bit fingerprint. Vector Quotient Filter finds two candidate blocks that have the corresponding bucket and inserts the r-bit fingerprint into the emptier block. The block also has a (b+s)-bit metadata area that indicates the number of slots occupied within the block. In our implementation, we use an 8-bit fingerprint with 48 slots and 80 logical buckets in a block. Each block fits in a 512-bit cache line. By using this block structure with AVX-512 vector instruction, Vector Quotient Filter optimizes the performance of insert, lookup, and removal operations in $O(1)$ time.

Vector Quotient Filter has a limitation in storing more than 48 fingerprints in a single block. When insertions to fully occupied blocks occur, it does not make additional space, unlike other filters which use linear probing [2] or cuckoo hashing [3]. Furthermore, as Vector Quotient Filter allows duplicate fingerprints to be stored in separate slots, the insertion of duplicates increases the likelihood of insertion failure. Since real datasets follow a skewed distribution, specifically the Zipfian distribution, Vector Quotient Filter is more vulnerable to experiencing insertion failures.



(a) Vector Quotient Filter (b) Counting Vector Quotient Filter

Fig. 1: 96 Insertions of 3 to Vector Quotient Filter and Counting Vector Quotient Filter

Vector Quotient Filter can overcome this space overhead by using the counting algorithm proposed in the Counting Quotient Filter [8]. Fig. 1 shows the 48 slots of Vector Quotient Filter and Counting Vector Quotient Filter, a Vector Quotient Filter with a counting algorithm implemented. In the worst case, when a duplicate item is inserted 96 times, Vector Quotient Filter assigns slots to each duplicate item, making the two candidate blocks full. Additional insertion of item 3 will cause an insertion failure. However, in Counting Vector

TABLE I: Size of filters, 2^{17} slots

Filter	Size(KB)
Quotient Filter	176
Cuckoo Filter	256
Morton Filter	178.1
Vector Quotient Filter	170.7
Counting Vector Quotient Filter	170.7

Quotient Filter, it uses a counter to count the duplicate items. In Fig. 1(b), the orange slots are used as counters for item 3. Compared to Vector Quotient Filter which uses all 96 slots for 96 duplicate items, Counting Vector Quotient Filter can use only 4 slots.

III. Evaluation

We compare the space usage and performance of Quotient Filter, Cuckoo Filter, Morton Filter, Vector Quotient Filter, and Counting Vector Quotient Filter. All filters are configured to have 2^{17} slots, each storing 8-bit fingerprints, to reside in a 1.3MB L2 cache. All the experiments are done on AWS EC2 "m6i.large" instance, which uses 2 vCPUs, 8GB of DRAM, and Intel Ice Lake CPU(Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90GHz) running Ubuntu 20.04.6 LTS. EBS type "gp2" volume is used as the secondary storage.

A. Space Overhead

We first examine the size of filters that have the same number of slots and fingerprint size. From Table I, we observe that Vector Quotient Filter and Counting Vector Quotient Filter achieve the smallest filter size, 170.7KB.

Table II shows the maximum allowed number of keys of Vector Quotient Filter and Counting Vector Quotient Filter on different Zipfian constants when they both have 2^{17} slots. Vector Quotient Filter experiences a significant decrease in number of keys that can be inserted from a Zipf constant of 0.8. On the other hand, since Counting Vector Quotient Filter has a counting algorithm, it can contain more keys on high Zipf constants. This leads to space overhead in skewed datasets. As Vector Quotient Filter and Counting Vector Quotient Filter both use 170.7KB, the bits per key used decreases as the filter can accept more keys. In Zipf Constant 0.99, while Counting Vector Quotient Filter uses 8 bits per key, Vector Quotient Filter uses 554 bits per key.

B. Performance

Insert throughputs of items having a uniform distribution are measured by varying the load factor in Fig. 2. As the Counting Vector Quotient Filter uses an additional algorithm to count items, it has a lower throughput than the Vector Quotient Filter. However, in most cases, Counting Vector Quotient Filter has a higher or comparable throughput among other filters. It has

TABLE II: Maximum allowed number of keys

Zipfian Constant	VQF	CVQF
0	123,207	123,207
0.5	125,113	125,197
0.8	40,822	127,139
0.9	8,525	137,625
0.99	2,522	173,015

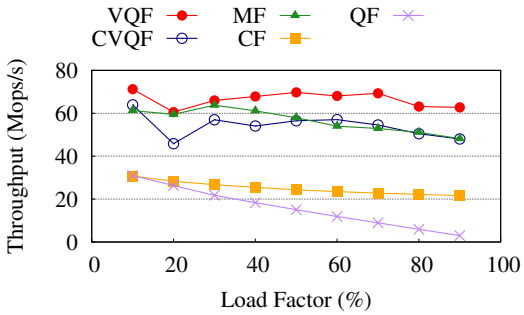


Fig. 2: Insert throughput: Quotient Filter(QF), Cuckoo Filter(CF), Morton Filter(MF), Vector Quotient Filter(VQF), and Counting Vector Quotient Filter(CVQF)

a higher throughput than Quotient Filter and Cuckoo Filter. When the load factor is low, it has a lower throughput than the Morton Filter. However, it has a similar insert throughput when the load factor is high. In all load factors, it has a low throughput than the Vector Quotient Filter due to the additional counting algorithm.

IV. Conclusion

This work examines the feasibility of incorporating a counting mechanism to Vector Quotient Filter to overcome the space overhead on skewed datasets. Our evaluation shows that Counting Vector Quotient Filter uses the space efficiently on skewed datasets, and has acceptable insert performance. For future work, we plan to implement Counting Vector Quotient Filter on a database system and observe the effects on memory usage on various workloads with skewed datasets.

Acknowledgment

This work was supported by NRF of Korea (No. NRF2020R1A2C3008525) and SNU-SK hynix Inc. Solution Research Center (S3RC) (No. MOUS002S).

References

[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[2] M. A. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R. P. Spillane, and E. Zadok, "Don't thrash: how to cache your hash on flash," in *Proceedings of 3rd Workshop on Hot Topics in Storage and File Systems (HotStorage, 11)*, 2011.

[3] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 75–88, 2014.

[4] A. D. Breslow and N. S. Jayasena, "Morton filters: faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity," *Proceedings of the VLDB Endowment*, vol. 11, no. 9, pp. 1041–1055, 2018.

[5] P. Pandey, A. Conway, J. Durie, M. A. Bender, M. Farach-Colton, and R. Johnson, "Vector quotient filters: Overcoming the time/space trade-off in filter design," in *Proceedings of the 2021 International Conference on Management of Data*, pp. 1386–1399, 2021.

[6] B. Debnath, S. Sengupta, J. Li, D. J. Lilja, and D. H. Du, "Bloomflash: Bloom filter on flash-based storage," in *2011 31st International Conference on Distributed Computing Systems*, pp. 635–644, IEEE, 2011.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM transactions on networking*, vol. 8, no. 3, pp. 281–293, 2000.

[8] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "A general-purpose counting filter: Making every bit count," in *Proceedings of the 2017 ACM international conference on Management of Data*, pp. 775–787, 2017.