Project Proposal - Analyzing performance of Cassandra with Cuckoo Filter

Aman Khalid aman.khalid@slu.edu
001086158

1. Introduction

Cassandra is a distributed data-store designed at Facebook Lakshman and Malik (2010) to support high concurrency at large workloads, it was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency. It offers high availability, replication and a flexible NoSQL schema.

Every Cassandra datacenter organizes its nodes in a cluster Foundation (2016), these nodes are responsible for storing data. The nodes communicate with each other using gossip protocol to support fault tolerance. The nodes of a cluster are arranged in a distributed hash ring.

Data is replicated many times a ring. Every node writes the incoming data to the in-memory memtables in append-only fashion, when the size of the memtable reaches threshold, it is flushed to the immutable SSTables. For efficient lookup Cassandra uses a probabilistic data structure called bloom-filters.

The Bloom filter represents the values in the SSTables. It is a quick way to check weather a key exists in the SSTable or not. It is much faster than linear lookup or binary search, and is much more practical at scale at which Cassandra operates. But as with everything else there is no free lunch, depending on the configuration of the filter it is possible to get *false positives* when the size of data increases.

Because Bloom filter uses non-cryptographic hash functions to represent the data, it is possible to have collisions and therefore it can say that an entry is in the SSTable even when its not there, however its impossible to get false negatives.

The bloom filter has many applications Broder et al. (2002) in networking and the reason for its popularity is space efficiency and quick membership queries. There are variations of Bloom filter for different requirements like scalability Paulo Sérgio Almeida and Hutchison (2007) or support for deletions Varghese (2006). However there is better way to implement lookups for large databases, without compromising on space complexity or rate of false positives via **cuckoo filters**. It hasn't been used much outside academia but has been proven to work better than bloom filter and its variants Fan et al. (2014).

2. Objective

The cuckoo filter is also a probabilistic data structure that supports set membership testing like bloom filter cuckoo filter improves upon the bloom filter in that it offers deletion, limited counting, and a bounded false positive probability and has a similar space complexity.

Moreover, the cuckoo filter guarantees low false-positive rates even when 95 percent of space has been utilized. The filters store fingerprints of the values being inserted thus being memory efficient. It does that by using cuckoo hashes, it is a technique for implementing a hash table. As opposed to most other hash tables, it achieves constant time worst-case complexity for lookups.

Cuckoo hashes maintains two lookup tables or "buckets" that assigned to two different hash functions, therefore only two lookups are required, which makes its worst-case time complexity $O(1)$

2.1 Expected Results

A node in cassandra has downtime when its in the process of flusing the in-memory memtable to the disk, having too many flushes can result in low availability. Since it is now possible to have larger data without compromising false positive rates We can have less frequent memtable flushes, or have larger memtables.

Algorithm 1 The lookup process of a cuckoo filter

```
function Lookup($\mathbf{x}$)
    $f = fingerprint(\mathbf{x})$
    $i_1 = hash(\mathbf{x})$
    $i_1 = i_2 \oplus hash(\mathbf{x})$
    if $f$ in $bucket[i_1]$ or $bucket[i_2]$ then
        return $true$
    return $false$
```

By using cuckoo filters we can have better lookup time despite having large SSTables. And at that scale it might even be possible to get constant amortized time which is a significant improvement over bloom filters

2.2 Implementation

The project will be done in two phase the first phase would be the implementation of cuckoo filters for Cassandra open source project. At present Cassandra stores its bloom filters in-memory, and gives users the option to set the desired false positive rates, the user can choose to have lower false positives at the expense of memory DataStax (2020b).

The second phase would be to setup the test-bed and analyze the performance on three different compaction strategies DataStax (2020a). Compaction is a technique that Cassandra uses to merge SSTables in the background. Depending on the configuration being used, it is triggered to eliminate duplicates, save memory. The bloom filters are reset whenever this happens.

References

Andrei Broder, Michael Mitzenmacher, and Andrei Broder I Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

DataStax. Configuring compaction `https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/operations/opsConfigureCompaction.html?hl=compaction`. 2020a.

DataStax. Tuning bloom filters `https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/operations/opsTuningBloomFilters.html?hl=bloom`. 2020b.

Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page 75–88, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450332798. doi: 10.1145/2674005.2674994. URL `https://doi.org/10.1145/2674005.2674994`.

The Apache Software Foundation. Dynamo `http://cassandra.apache.org/doc/latest/architecture/dynamo.html`. 2016.

Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, 2010. ISSN 0163-5980. URL `https://doi.org/10.1145/1773912.1773922`.

Nuno Preguiça Paulo Sérgio Almeida, Carlos Baquero and David Hutchison. Scalable bloom filters. 101(6):255–261, 2007. ISSN 0020-0190. URL `http://doi.org/10.1016/j.ipl.2006.10.007`.

Flavio BonomiMichael MitzenmacherRina PanigrahySushil SinghGeorge Varghese. An improved construction for counting bloom filters. In *Algorithms – ESA 2006*, pages 684–695. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-38876-0.