Purposing A Hbase-BloomFilter (a Bloom Filter that optimized for densely continuing objects that pre-know the range)

Got this idea when last week discuss with Rahul G about bloom filters. Put up some draft here, feedback please!

Idea:
Regular bloom filter has been widely used in HBase and LSM, however in these cases, certain important knowledge of the key space is pre-known but not been take advantage of. Such as:
1, the MIN/MAX of the keys is pre-known.
2, the keys are usually densely and continuously sorted within a range.

Issues of regular bloom filter:
Current bloom filter makes no the assumptions of the distributions of the incoming objects. After hashing, the hashes is assumed to be evenly distributed among (-int, int).
Take most popular implementation for example (google/**guava**.bloomfilter), there are two processes involved: hashing and modulo, as illustrated below. (code link)

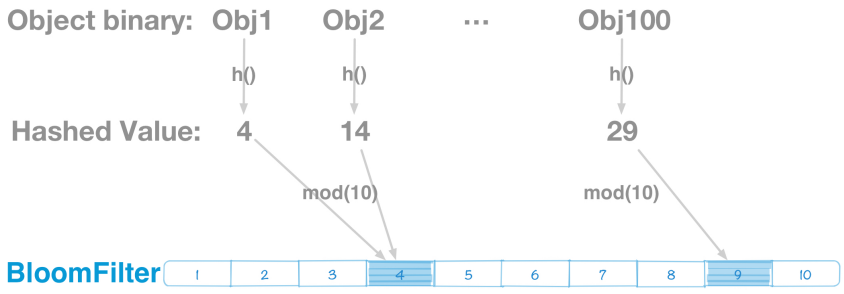


Fig 1 Two processes for most Bloomfilter implementation. Hashing (fnv or murmur being most popular chooses) and modulo. Modulo operation is usually necessary in order to make hashing range fit into the length of bloom Filter.

Issue is that, no matter which hashing algorithm it choose(fnv or murmur being most popular), a modularization process is usually necessary to fit the size of bloom filter (Such as the Line 61 of com.google.common.hash.BloomFilterStrategies.java code link). As illustrated below, each bloom filter buckets are not giving the even collision chance therefore yield a sub-optimum collision/space ratio.

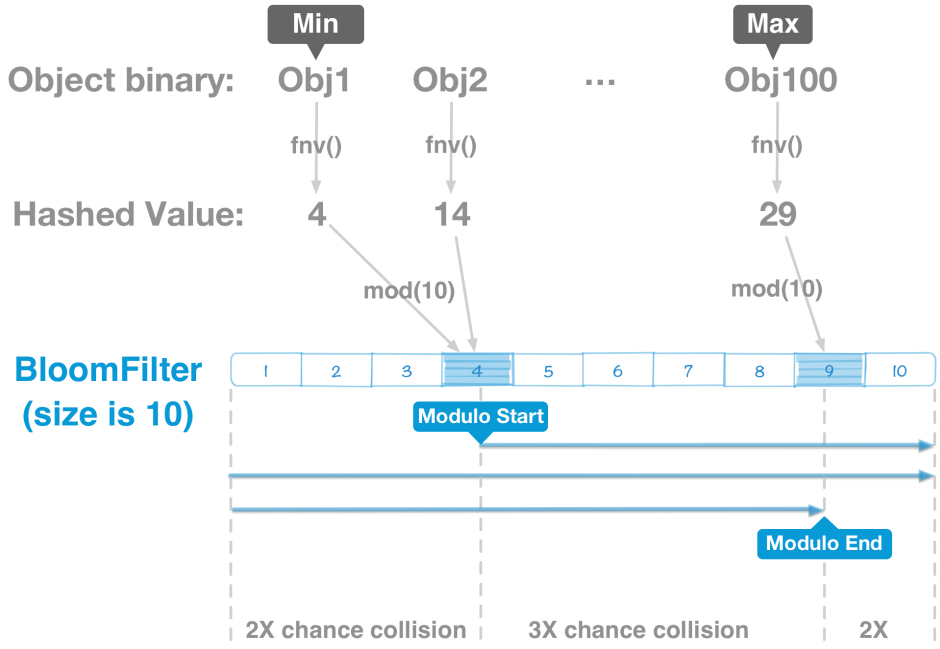
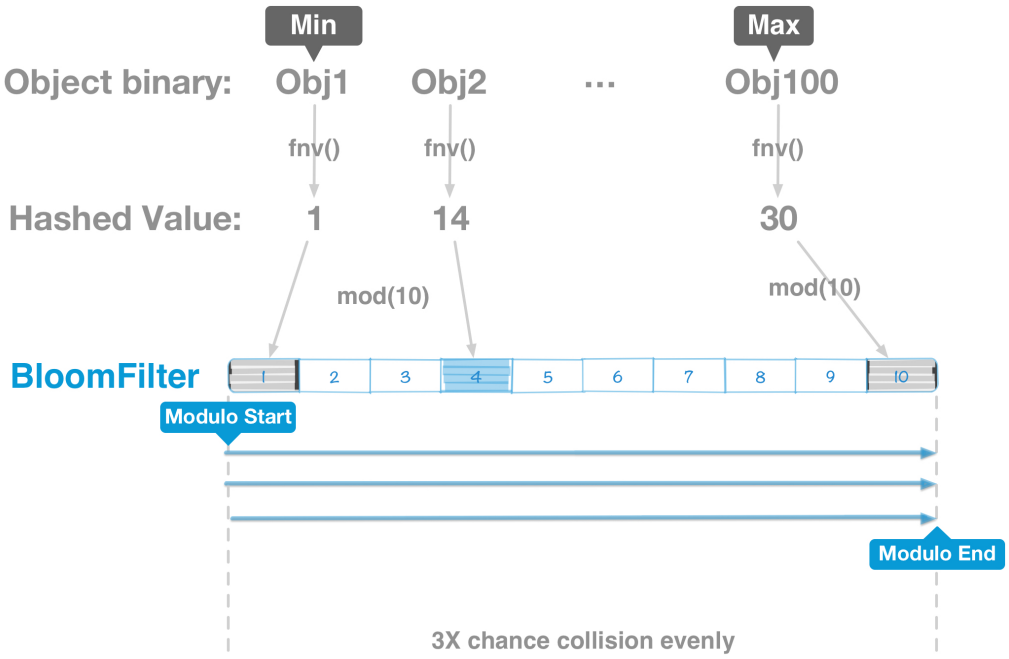


Fig 2. In this example, range of hashes is [4, 29]. When mod this down to [1,10], we can observe that the range [4, 9] will be "visit" 3 times while other ranges only 2 times. Therefore the items are denser between [4, 9] on the bloom filter. Reason of this is that the regular bloom filter don't know the min/max value of the incoming objects ahead of time, therefore can not prepare a good corresponding Hash/modulo process for it.

Improvement from HbaseBloomFilter(purposing)

During memstore flushing time, we know Min/Max of the key space a head of time, we can use this information to prepare a better hash and modulo process that avoid this issue. Illustrated below.



So How much does it improve?
From a experiment, from a naive implementation of this idea, (Code is here), result as below(each experiment set with the average of multiple experiments):

Bloom filter size is 100, fit in 100 values ranging from 1-150. Use identity based hash:
OldBloomFilter: Occupied Rate 65%, Collision rate:35%
NewBloomFilter: Occupied Rate 67%, Collision rate:33% (2% better)

Bloom filter size is 10K, fit in 100 values ranging from 1-10K. Use FNV hash:
OldBloomFilter: Occupied Rate 63.22%, Collision rate:36.78%
NewBloomFilter: Occupied Rate 62.72%, Collision rate:37.28% (0.5% worse)