USING BLOOM FILTER FOR BIG DATASET

A probabilistic data structure for membership test.

Problem Statement

Given a dataset of 100 Billions size with below four columns

- User_id - 50 M Cardinality
- Merchant_id : 10 K Cardinality
- Total_spend- Amount Spend
- Day: contains last 5 years date in “YYYY-MM-DD”

Design an system/api to test membership by any combination of query containing {user_id, merchant_id, day}. If no item in the dataset matches the query than return “Yes” else return “No”.

“Yes” means 100 % sure the record is not in the dataset , “No” means the record maybe locate at the datasets.

Choice Of Data Structure

The First choice came to mind is Hash Table Data Structure. However, as the problem statement has probabilistic nature on membership test, I can use bloom filter considering its efficient time and space usage.

Time Complexity: If we are using a bloom filter with m bits and *k* hash function, insertion and search will both take $O(k)$ time.

Space Complexity: For bloom filter with m bits, it will be $O(m)$.

Approach:

- Implemented Bloom Filter Data Structure; using `MD5' hashing algorithm, any other hashing can be used such as “SHA-256”, “Murmur” etc.

BloomFilter(int numRow, float fpProb); // numRow = total number of records; fpProb = False positive probability

Public Methods :

- *add(T item) // add an item of type T*
- *contains(T Item) // check if item exists*
- *clear() // for testing, it clears bloom filter*
- *getBloomFilterSize() // BitSet size in bytes*

Some Metrics:

Insertion Speed	**Inserted 1000000 elements in 2.076962 Seconds.**	**481472 elements/second**
Query Speed	**Queried 1000000 elements in 0.596466000 Seconds.**	1.67654e+06 elements/secon**d**

BigDataStore

For given problem, I implemented BigDataStore containing 3 bloom filters for day, merchant_id and user_id in BloomDataStore.java.

bf_day-> bloom filter for day, bf_user-> bloom filter for user, bf_merchant->bloom filter for merchant
BitSet Size of user bf is 479252928
BitSet Size of merchant bf is 95872
BitSet Size of day bf is 17600

For Insertion of records:

1. Fetch the record
2. validate record, if not valid return
3. Parse day, merchant_id and user_id; add to respective bloom filter atomically

For Search Query: The idea is simple to start with smaller size bloom filter i.e day and then go to merchant and than finally look in user's

1. Parse query
2. Validate query, not valid throw "invalid query" error
3. If day in query, look up in day bloom filter, if not contains return "Yes" else set 'isDayAvailable= true'
4. If merchant_id in query, look up in merchant's bloom filter, if not contains return "Yes" else set isMerchantAvailable= true
5. If user_id in query, look up in user's bloom filter, if not contains return "Yes" else set isUserAvailable= true
6. If any of isDayAvailable= true OR isMerchantAvailable= true OR sUserAvailable= true; than return "NO"
7. Else return "Yes"

Testing

Tested using a public DataSet : Downloaded BlackFriday.csv from kaggle and converted to my use case using python script “prep_dataset.py”

Testing Program: BigDataStoreTest.java
Insert records: BlackFriday.txt
Query records: test_BlackFriday.txt

To Compile and test all together : **./run.sh**

Time taken to Insert all 537577 records is ~5 Seconds.
Time taken to query all 537577 the records is ~4 Seconds.

The above performance number is on local computer, which will surely degrade over network. Sequential Insertion and Query Time seems to be performing well, which can be further improved by using batching in insertion and query api.

Scalability: Parallelism And Distribution

Currently, I am checking membership in bloom filter sequential starting from smaller one i.e day to merchant's to larger one user's bloom filter.

However, possibility is to implement parallelism and I can look in all the three filter's parallely which will reduce the query time significantly.

All my experiment done in standalone machine. However, this api can easily be extended for distributed mode.

Possible Extension: Have a Writer box and couple of reader box. Deploy this service across all the boxes. Any Insertion of records will be served from writer box and query from reader box. In background both will sync the state and keep the state eventual consistent.

Fault Tolerance And Failover

For standalone:

Keep rolling snapshot, on crash reload from file system.

For Distributed:

In replica mode with frequent snapshot and load balancer.

Thanks!

Any Questions?

Source Code available in https://github.com/ckmishra/bloom_filter_poc