ISSN:0975-9646

A Design for a BloomFilter Variant Based Cloud Database Validation System

Arun Swaminathan[1], Udit Mehta[2], Puneet Kohli[3], Mihil Ramaiya[4]

Department of Computer Engineering and Information Technology, College of Engineering Pune
Shivajinagar, Pune, Maharashtra, India.

Abstract— **As databases become increasingly ubiquitous, one of the relevant problems involves maintaining data in a consistent and error free manner across remote systems. Considering the magnitude of data contained in the diverse existing types of databases, the bottleneck occurs at transmission bandwidth of the network during the synchronized exchange of database information during updates. As opposed to a complete cell by cell comparison for all tuples required for database validation - In this paper, the proposed design achieves validation of a source database with a remote target in constant space complexity by passing a data structure, Bloom Filter as a message. Any discrepancy is reported in a verbose manner and the false probability rate based on the bloom filter parameters can be tuned to achieve an optimal balance between spurious hits and transmitted message size. The analysis of three bloom filter variants namely Scalable, Cuckoo and Standard Bloom filter is carried out to test feasibility in the context of the proposed design by measuring their computation time and the size of the bloom filter array produced that will be transmitted across the network.**

Keywords— ***Bloomfilter, Databases, RDBMS, Validation, Compression***

I. Introduction

Databases have become ubiquitous for both commercial and noncommercial applications. Depending on the usage scenario, they have varied sizes ranging from a few megabytes to petabytes and may be RDBMS or NoSQL in nature. One of the prerequisites of most applications involve this data being maintained in an error-free and consistent manner and in synchronization across multiple remote systems. This is achieved through an arduous, well-timed exchange of database information and updates across the network with the bottleneck occurring at the transmission bandwidth of the network.

Bloom filters are a data structure having a bit array and a set of hash functions associated for populating the array. It can posit whether an element is a part of the set or not with a certain probability of occurrence of false positives. Typical validation would involve two systems, a source and target machine - with the target verifying the presence or alteration of a tuple with the desired facility of verbose error reporting. If any discrepancy is detected during the analysis of the Bloom Filter sent as a message, it may initiate a cell by cell transfer or comparison.

Significant savings in terms of bits transmitted across the network can be obtained by passing the data structure Bloom Filter as a message among systems involving validation. Processing during compression or decompression as well as lookup computation time are secondary metrics in this particular application as compared to the transmission size and false positive rate.

This paper aims to analyze the performance of three variants of Bloom Filters and the tuning of their parameters for remote database validation using cloud based message passing to determine which delivers the least false probability ratio while minimizing the size of the compressed network message.

II. Literature Review

A. Bloom Filter

Bloom filter originated in 1970s and since has been widely used as a space-efficient alternative for fast matching in IP routing, in distributed databases for element membership query problems and other network applications.

A compact set representation is provided by standard bloom filters. They support insert and lookup operations. The False Positive(FP) rate of a Bloom Filter can be tuned. The query result would either be “definitely not” or “probably yes”. The trade-off is between the efficiency and the space that the bloom filter requires.

There are k-hash functions. All bits are initially “0”. In order to hash an item, k-hash functions insert the item in k-positions bit by bit, thereby setting k-bits to 1. For lookup, k bits corresponding to the hash functions are checked. If all the k-bits are “1”, then the item may be present and the query returns true. However, this may be a false positive but false negatives are absent in Bloom Filters.

B. Bloom Filter Variants

Bloom-1 or Fast Bloom Filter has a slightly higher false positive rate given a memory size but reduces the overhead for query operations [1]. The new Scalable Bloom Filter provides false positive rate as low as 21.3% of the dynamic Bloom filter presented before and the querying CPU time increasing logarithmically rather than linearly [2].

Large data sets can successfully be validated using a series of Bloom filters (where each BF validates one subset of the database) whose error rate can be controlled by enlarging the capacity of the DBA. These bloom filters provide high performance as they can be accessed in parallel [3]. A bloom filter variant for fast and scalable applications such as secure broadcast in wireless networks involves the use of Tiger hash function that has greater resistance to collisions due to nonlinear avalanche bit generation. Compact mapping is done by using LSFR counter arrays and whole design can be reconfigured by FGPA implementation [4].

One-Hashing Bloom Filter uses one base hash function and modulo operations involving a consecutive prime numbers set to have properties of distinct independent hash functions while minimising the computation overhead involved in hashing [5].A Bloom Filter variant called TinySet has greater space efficiency along with partial support for removals for false positive rates less than 2.8 % [6]. Another approach involves a Bloom Filter based tree that works by assigning bloom filters to a fraction of interior nodes for searching a data element in tree structured-data and can prune out subtrees from the dataset [7]. A tree structured Bloom Filter has '*i*' hash functions ('*i*' being the depth of the tree), constant time complexity and uses 1 Dimensional Bit array for efficient storage of bits [8].Cuckoo filters store the fingerprints of bits in hash tables due to which items can dynamically be added or removed. Moreover, it uses substantially low space (when FP rate is less than 3%) and provides better lookup performance [9]. A bloom filter variant called Partitioned BF (Par-BF) involves a trade-off balance between low-overhead and fast performance using a group of formulas to tune essential parameters for dynamic data sets. It supports fast matching in parallel and outperforms Scalable Bloom Filter and Dynamic Bloom Filter(DBF) with a memory overhead less than DBF through it's garbage collection policy [10].

The metrics for performance measurement for a Bloom Filter include memory requirement, false positive rate and overhead for queries. Bloom filters are valid in applications where false positives can be tolerated but false negatives are unacceptable [11].

III. Methodology

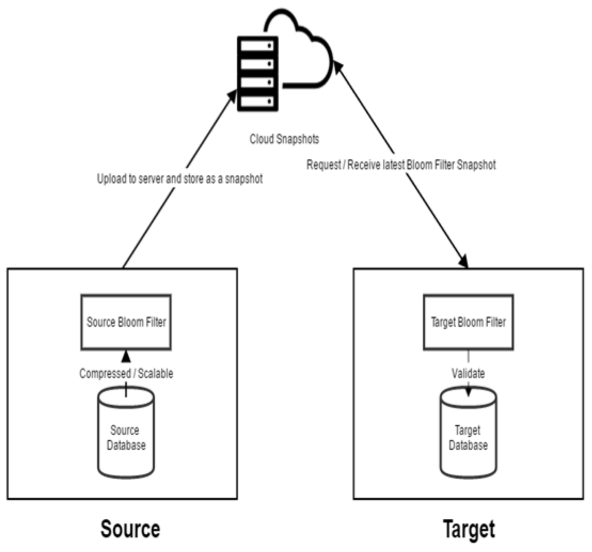

Fig 1. Overall system diagram for a cloud-based validation system using bloom filter

1. Create a Bloom Filter from the Source Database. This is done by using each row as a single entry into the bloom filter. The efficiency of generating this bloom filter can be quickened by parallelizing the entry into the bloom filter by splitting the total number of rows among threads where each performs hashing over it's allotted dataset.
2. This bloom filter is sent to the cloud server over the web interface using the HTTP interface. The size of the bloom filter determines the amount of bandwidth utilized in this transaction and can be reduced by using a bloom filter variant that uses compression on it's internal computed data structure that shall be passed as a message.
3. While implementing care, the above two steps has to be maintained that there should be no false negativity introduced, and the false positive rate shouldn't be too high as there is a tradeoff involved in the data structure size and false positive probability rate.

$$p = \left(1 - e^{-\frac{kn}{m}}\right)^k$$

p – False positive probability
k – Number of hash functions
n – Size of the input
m – Size of the bloom filter

4. The cloud server maintains snapshots of every bloom filter sent to it, for target databases to validate themselves at different points of time. Also an updates to a particular bloom filter snapshot can be stored as a new snapshot or the same bloom filter can be rewritten, to save server memory. If historical database validation is desired, a versioning system of these snapshots can be maintained on the server.
5. The target database can send an HTTP request to the cloud server requesting a bloom filter (snapshot) corresponding to a particular source database instance at a given time.
6. The target bloom filter can validate itself against the received bloom filter and it can be found out which rows of the target are not maintained with integrity or are missing in the source, due to various reasons such as corruption of primary key.

IV. Results

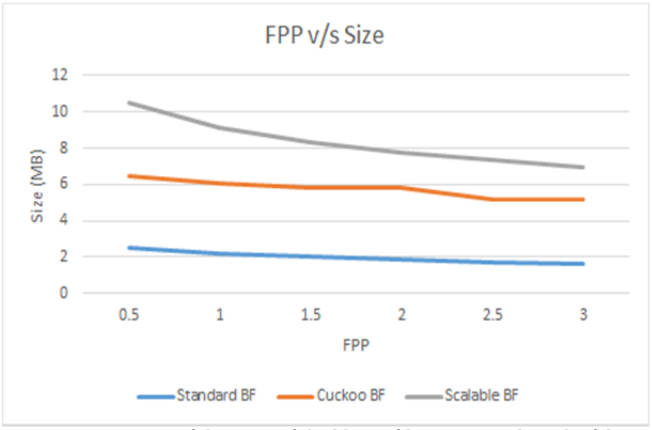

Fig 2. Comparison of the sizes of the bloom filter generated v/s the false positive probability

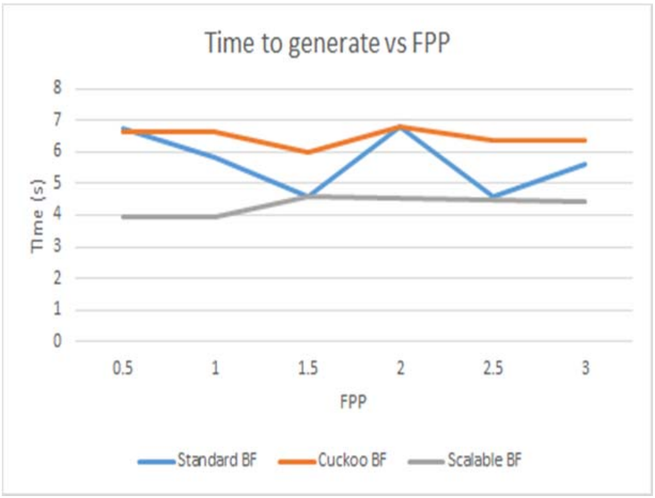

Fig 3. Comparison of the time taken to generate the bloom filter v/s the false positive probability

We compared three different variants of bloom filters (standard, cuckoo and scalable) on the basis of their false positive probability, size of bloom filter data structure computed and the time taken to generate the bloom filter. These parameters decide the effectiveness of using the bloom filter in the aforementioned architecture and application.

We have fixed the number of rows to 1 million records in the source/target database for each test, and the test is run on a machine with 8GB RAM, Intel i7 4th Generation, 2.4 GHz X8.

For each test, we have varied the false positive probability, to find the minimum possible size of the bloom filter bit array that is generated per variant and the time taken to generate this bit array and for encapsulating it to be sent as a network message. The above graphs measure space of the network message and time taken for generating it as a function of the false positive probability rate that is achieved.

In the first graph we can see that standard bloom filter has the lowest file size generated in comparison to the other two variants. This goes in accordance with the theory that there is an overhead with the cuckoo and scalable filters when we create the data structure.

In the second graph, we can see from our experimentation that the scalable filter is created in the shortest time. Hence in time critical application it would be more beneficial to go with a scalable bloom filter.

From both the experiments we can see that the cuckoo filter has both moderate file size and generation time.

Thus by looking at the above two graphs, we can conclude that when size is a major consideration and not time, we should stick to the standard bloom filter, whereas the scalable can be chosen when time is critical. Cuckoo filter has an intermediate performance so it can be chosen when both factors are important.

V. CONCLUSION

The design elucidated in the paper would use Bloom Filter variants to remotely validate large databases in cloud environments. Through the results, it may be inferred that for this scenario, the standard bloom filter is best applied in networks with low bandwidth whereas a scalable bloom filter can be employed when the metric is quick creation time.

This system can be extended to maintain a cloud based library of snapshots for target databases to validate against previous states of a source database. Thus, significant savings in terms of transmission bandwidth can be achieved.

Since the false probability rate, hash functions and other parameters can be tuned, this design with constant space complexity will find versatile applications in remote database validations.

REFERENCES

[1] P. Reviriego, K. Christensen and J. A. Maestro, "A Comment on “Fast Bloom Filters and Their Generalization”," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 303-304, Jan. 1 2016.

[2] K. Xie, Y. Min, D. Zhang, J. Wen and G. Xie, "A Scalable Bloom Filter for Membership Queries," *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, Washington, DC, 2007, pp. 543-547.
doi: 10.1109/GLOCOM.2007.107

[3] J. Wei, H. Jiang, K. Zhou and D. Feng, "DBA: A Dynamic Bloom Filter Array for Scalable Membership Representation of Variable Large Data Sets," *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, Singapore, **2011**, pp. 466-468.

[4] K. Saravanan and A. Senthilkumar, "Investigation on bloom filter and implementation of 3k combined parallel tiger bloom filter design," *Electronics and Communication Systems (ICECS), 2014 International Conference on*, Coimbatore, 2014, pp. 1-7.

[5] J. Lu, "One-hashing bloom filter," *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, Portland, OR, 2015, pp. 289-298.doi: 10.1109/IWQoS.2015.7404748

[6] Gil Einziger Roy Friedman “TinySet - An Access Efficient Self Adjusting Bloom Filter Construction” CS-2015-03 – 2015, Technion, Computer Science Department- Technical Report

[7] M. Wang and Y. Zhu, "Bloom Filter Tree for Fast Search in Tree-Structured Data," *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, 2015, pp. 18-23.

[8] Yongquan Fu and Ernst Biersack. 2015. Tree-structured Bloom Filters for Joint Optimization of False Positive Probability and Transmission Bandwidth. *SIGMETRICS Perform. Eval. Rev.* 43, 1 (June 2015), 437-438

[9] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies* (CoNEXT '14)

[10] Yi Liu, Xiongzi Ge, David H. C. Du, and Xiaoxia Huang. 2014. Par-BF: a parallel partitioned Bloom filter for dynamic data sets. In *Proceedings of the 2014 International Workshop on Data Intensive Scalable Computing Systems* (DISCS '14). IEEE Press, Piscataway, NJ, USA, 1-8.

[11] O. Rottenstreich and I. Keslassy, "The Bloom Paradox: When Not to Use a Bloom Filter," in *IEEE/ACM Transactions on Networking*, vol. 23, no. 3, pp. 703-716, June 2015.