Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 5, May 2015, pg.1094 – 1100

RESEARCH ARTICLE

An FPGA Implementation of Hashed Key-Value Store Using Bloom Filter

Sireesha Y[1], M. Roopa[2]

[1]Department of Electronics and Communication, DSCE Bangalore

[2]Department of Electronics and Communication, DSCE Bangalore

[1] sireeshay23@gmail.com

[2] surajroopa@gmail.com

Abstract: In distributed applications of database management lengthy memory access greatly reduces the performance. To overcome the limitation network processors utilize different techniques. Key-Value store is the key intrinsic component for many network applications just as online retailing and cloud computing. Such storage usually supports for operation on key value pairs. So, it can be stored in memory and be used for data-retrieval hierarchy, speed responses for queries. Till now, existing methods are deterministic. Providing such accuracy and storage comes at a cost of CPU time and memory. On contrary, in this paper we propose a new architecture for Key-Value pairs storage which is more compact. Its design is based on probabilistic space efficient data structure called bloom filter to test for membership in a set. In this project, we open up the concept of bloom filter to support key value storage operation with reduced false positive rate. The size of bloom filter need not be significantly large to minimize false positives. The bloom filter is widely used in various applications because of its storage efficiency and operational ability. Bloom filter is suitable for local functions such as logging, storage, filtering and reduces unnecessary memory transactions. Storing the inputs in the hashes as calculated and verifying their presence using bloom filter is coded in VHDL and simulated using Xilinx ISE, Implemented on Virtex 5. Simulation results shows false positive of proposed methods converges to zero and hash collisions of bloom filter has significantly reduced to zero and number of elements is inserted is increased. Finally we increase the accuracy of existence of elements by slicing bloom filter. Hence the overall false positive rate becomes zero.

Key-words: Distributed Systems, Bloom Filters, Key-Value store, False Positive.

1. INTRODUCTION

Given that, if there are millions or even billions of data, developing efficient storing, updating and querying the database become very crucial and necessary. The amount of data handled by internet users is growing fast like a wind fire in recent decades. Accordingly the amount of traffic has also been extremely increased. In order to handle such large amount of data safely, efficiently and filtering out redundant data would be useful in various applications.

For faster data access, Key-Value stores are used and it contains large number of KV pairs, data is indexed using a unique key to allow its retrieval easily. Key-Value store supports simple operations such as data lookup and KV pair insertion. Key-Value store is implemented through an index format mapping between key and values. Key-Value stores has replaced traditional databases in popular data management (index+ storage) solution for large volume of data, with many applications like social networks, web multiplayer gaming, Storage data deduplication [1]. By storing almost data in main memory, the memory access while processing would be difficult.

Many new algorithms have been proposed for distributed systems key-value stores based on bloom filters. Bloom filter is used to filter out inputs not included in a set. Bloom filters helps to reduces unnecessary accesses to much larger, much slower and power hungry addressable memories. They are used in network intrusion detection systems[2], Network devices utilizing

© 2015, IJCSMC All Rights Reserved

bloom filter on FPGA's shows proper balance between performance and flexibility which can make FPGA's more preferable than software and ASIC's.

In early systems, where memory was limited and valuable resource, the space saving of bloom filter offered significant performance advantages. Bloom filters are consistently implemented on on-chip memory for fast processing, it is capable of providing the desired answer to membership test much faster and save power because accessing bloom filter is much fastest that is of 25µs only compared to accessing memories. Google big table [3] uses bloom filter to find unique data and avoids latent secondary access. Other bloom filters applications in databases are Cassandra [4]Aamazon dynamo[5], Memcache D[6] .

Our goal is to develop enhanced version of bloom filter, so it will provide key value operations in faster rate with reduced latency .Specifically supporting these operations: configure, update, delete and query.

2. BLOOM FILTER

A bloom filter is space efficient ingenious data structure for probabilistic representation of array or a set for supporting the membership queries and allows to tell if an object is present more effectively. It reduces the energy and delay of membership tests by maintaining a precise and compact representation of large set to be searched. The basic operations includes adding of elements to set and querying element memberships. The accuracy and efficiency of bloom filter is dependent on the size of filter and number of elements inserted in it. Adding of elements to bloom filter increases its probability of reporting false positives by query operations.

Bloom filter is programmed with predefined set of rules which consists of N number of elements .This is consummate by hashing each element of the set with the k hash functions, the output of which functions as indices (address) for m-bit array and the bits at corresponding indices are referred as 'set'. After programming, review whether the piece of data is a member of set (referred as querying) that is to be hashed. If the corresponding bits of a bloom array are set accordingly, then that piece of data is certainly a part of the predefined set, if any of the corresponding bits are not set, then that data is definitely not a part of the set as shown in fig 1. The easiness of bloom filter with its excellent performance makes it a standard data search which is of great use to many network applications [7].Cassandra uses bloom filter to save IO to perform a key lookup: each database of Cassandra has a bloom filter associated with Cassandra checks, before doing any disk access. It is space efficient but at cost of false positives.

3. HASH FUNCTIONS

A natural way of representing a set in an encrypted form is by hashing technique. Hash representation of a normal set helps in querying at a faster rate and it is a new lookup technique [8]. A hash function is generally used to map digital data of arbitrary size into fixed size, with a few differences in rapid input data producing with large differences in output data. Hash coding techniques are used in saving large amount of space; allow efficient and effective membership lookups. They are used generally in computer software to detect location of data for fast database lookups and redundant data [9]. Hash functions are primarily used in hash tables to enable quicker location of data given in its search key. Usually, the hash function is used to map the search key to index, so that it gives the place in hash table in which the corresponding data is stored as shown in figure 2. Hash collisions impend if two data elements are mapped to same address [10]. These collisions are to be minimized. The hash functions used in bloom filter should be independent and uniformly distributed. For Example, Cassandra uses murmur hashes.

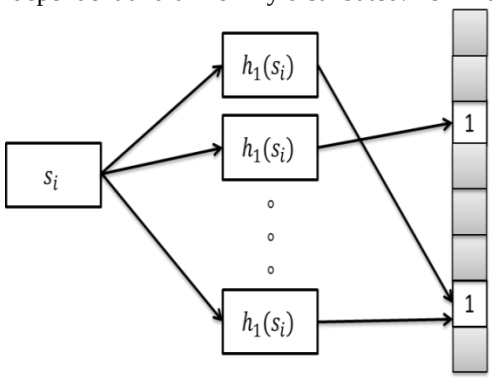


Figure. 1. Configuring of Bloom Filter, The s_i element is used as the input for the hash functions and the bits indexed by the $h_1, h_2, \cdots, h_k$ functions are set to 1

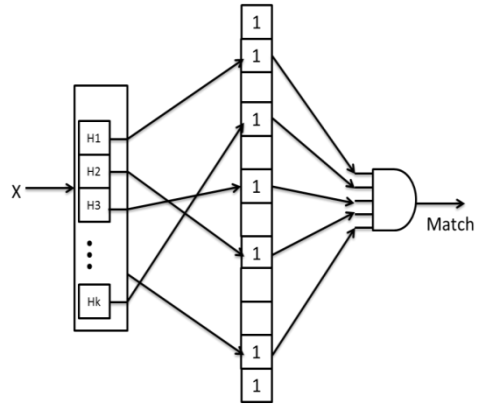


Figure. 2. Querying of Bloom Filter

4. RELATED WORK

In 1970, Burton Bloom described the compact probabilistic data structures, which are used to represent words in dictionary and these structures are used by early UNIX spell checkers. It has many applications in networking areas like IP address look up, Packet classification [11], Traffic measurement, Peer-Peer systems [12], Intrusion Detection and Web caching [13].

False positives:

An intrinsic probability of bloom filter is it may be used to produce a positive even if an input is not present in the set and it is referred to as false positive. These are generated because hashing used in programming does not guarantee one to one mapping between an element and a group of indices. Index is mapped by multiple elements. So, generally false positives gives a negative impact on systems performance.

When inserting an element into the filter, and the probability that a certain bit is not set by a hash function

$$1 - \frac{1}{m} \qquad (1)$$

Now there are k hash functions, and after inserting n elements to filter, the probability that a given bit is still zero is

$$\left(- \frac{1}{m} \right)^{kn} \qquad (2)$$

And consequently that the probability that the bit is one

$$\left(1 - \frac{1}{m}\right)^{kn} \qquad (3)$$

The probability of bloom filter claiming as presence when the element is not part of set is given by

$$\left(1 - \left(\quad (4) \quad \right)^{kn}\right)^{k} \approx \left(1 - e^{-kn/m}\right)^{k}$$

Where m= number of bits in bloom filter, k= number of optimal hash functions

The false-positive probability decreases as the size of the bloom filter, m increases as shown in equation (4). The probability increases with n as more elements are added. Now we want to minimize the probability of false positives, by minimizing $\left(1 - e^{-kn/m}\right)^{k}$ with respect to k. This is accomplished by taking the derivative and equating to zero, which gives optimal of k as

$$\left(\frac{1}{2}\right)^{k} = 0.6185^{m/n} \qquad (5)$$

The false positive rate after optimizing (5) is

$$P = \frac{1}{2}^{k} \qquad (6)$$

The number of hash functions is given as

$$k = \log_2(1/P) \qquad (8)$$

The number of elements inserted

$$N = \frac{M(ln2)^2}{|lnP|} \qquad (9)$$

A conventional bloom filter is referred to a simple input array implemented by single array of M bits, where M is the filter size. The filter is also parameterized by k hash functions. While inserting any element into the set, each of the k hash functions will generate a hash value that can be used as array index. That is for an element e, k indexes $h_1(e)$, $h_2(e)$…, $h_k(e)$ are generated and corresponding bits are set as shown in figure 3. When querying for an element, to get information as regards the presence of an element in a set, it will be sufficient to verify that all the bits in the mapped hash functions are one. In this method M bits are partitioned into k slices with m =M/k bits each and assign a hash function to each these slices. Every element is represented by k slices obtaining robustness. The high query latencies are involved with access down the storage hierarchy or across the network. Each false positive cause all KV pairs search "in vain" [14].

In order to maintain false positive rate as constant, it should be fixed in prior and this factor decides the number of elements to be inserted. This bloom filter is highly compact and it needs minimum 10 bits to store an element and false positive rate of 1% is obtained [15]. Henceforth, where only limited on chip RAM is available these bloom filters can be particularly used. The insertion of elements will be very less.

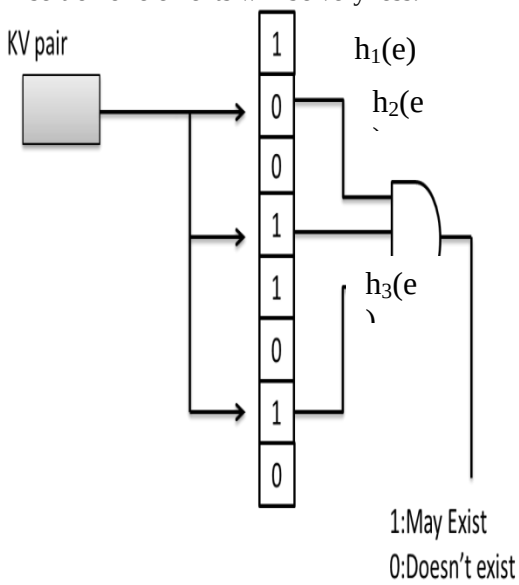


Figure. 3. Example of checking bits in Bloom Filter

5. PROPOSED ARCHITECTURE

In this section, We introduce the design of enhanced hashed key value store with bloom filter. Initially we check with problem formulation and the architecture overview and finally the analysis of results and the comparison.

I. PROBLEM FORMULATION

Assume that we have n key value pairs (k_i ,v_i) where i € [0, n-1]. The keys and values can be numbers or binary strings, our aim is to develop key value store to support the following operations for stored key value pairs

1) Insert (key, value) insert key value
2) Update (key, new_value) update a value for a key
3) Query (key, value) Query the value for a given key

II. ARCHITECTURE OVERVIEW

This architecture reduces the rate of false positives while maintaining the simplicity and storage efficiency of bloom filter and it focuses on data configuration and querying, that is instead of checking the complete memory storage and causing delay for checking existence of data, we check only preconfigured bloom filter. It consists of three main components Block RAM [16], Hash Generator and Bloom Filter

© 2015, IJCSMC All Rights Reserved

a. *Input controller*

The input is given serially to input controller buffer such that it stores in a parallel order according to the given reset. It is persistent storage area and the data can be updated according to user applications by reset. The control signal provides the decision to configure/update the given data or to query for the result.The Block RAM consists of series of input values given by input controller. This block RAM is obtained by IP coregen of Xilinx ISE.

b. Hash Generator

For hardware implementations a separate class of universal hash functions are proposed by Ramakrishna et.al [17]. Interesting class of hash functions are the Ex-OR based hash functions, which compute each set index bit as the Exclusive-OR of the subset of address bits as shown in equation (11). We have used these class of hash functions which are based on random values and Ex-OR operations. Ex-OR hashing is generally preferred because it avoids excessive hardware for multiplication, division and also long critical paths required for carry chains and sums. In configuring operation, hash generator computes addresses for storage and in querying the existence of must be data verified as shown in figure 4. Usually hash collisions are reduced by taking the bit length of random values more than input bit length. Accuracy of hash collisions is verified in MATLAB by taking varying bit lengths for random values and computing Ex_OR and calculating accuracy. Considering the trend in computational power versus memory access time, the practical bottleneck is the amount of (slow) memory accesses rather than computational time.

$$h_i(X) = d_{i1} \cdot x_1 \odot d_{i2} \cdot x_2 \odot \cdots\cdots \odot d_{ib} \cdot x_b \qquad (11)$$

Where d=random values $\{d_1, d_2, d_3 ..... d_b\}$, $x=\{x_1, x_2, x_3 ..... x_b\}$

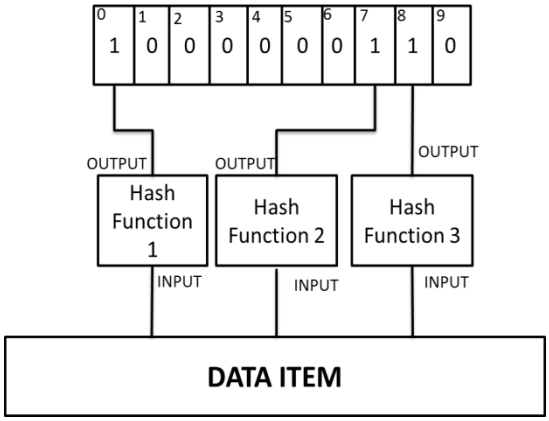


Figure. 4. Indices generated by Hash Generator for Inputs data and configuring Bloom Filter

c. Bloom Filter

Bloom filter reduces unnecessary accesses to main memory thereby increasing performance. The reset, configure and query operations are performed as per control signals given by De-MUX according to user constraints. Initially, bloom filter array is allocated and initialized to zeros, the hashing results are used to index input bit array and each indexed bit is set. After the insertions memory queries can be conducted and verify whether the element exists as member of set as shown in fig 5.As the bloom filter does not store the element themselves, it reduces the disk IO to look them up. Its lookups are independent and can be parallelized. Time needed to query is constant and independent of number of data inserted. Bloom filter may return false positives but this can be set very low.

© 2015, IJCSMC All Rights Reserved

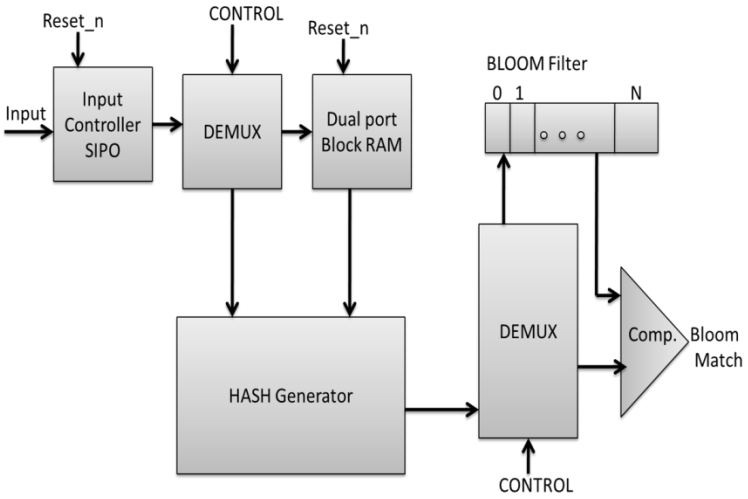


Figure 5. Proposed Architecture of Bloom

6. EXPERIMENTAL RESULTS

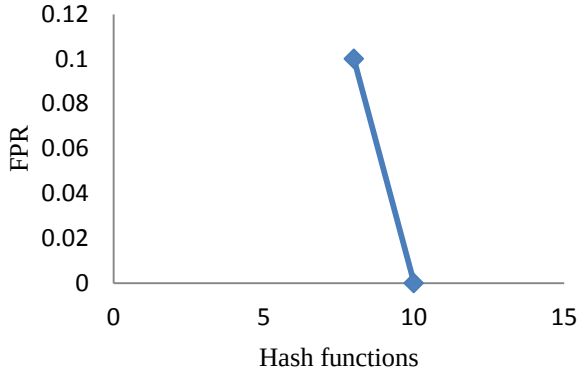
Analysis of performance can be made by results showing that FPGA uses its parallel operations such as bloom filter calculations, hash values calculation and hash table lookup so that memory overhead is reduced and making it favourable for applications in large number of databases. The design is coded using VHDL and synthesized using Xilinx ISE and targeted for Virtex FPGA. The implementation constraints are given as

Table 1. Comparison of Ex-OR hashed bloom filter with conventional bloom filter

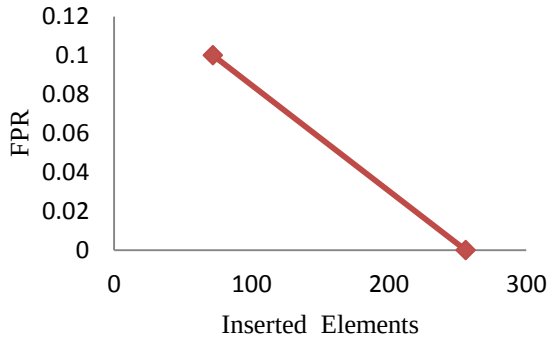
Parameter	**Conventional Bloom Filter**	**Ex-OR Hashed Bloom Filter**
M	1 KB	1KB
N	72	256
K	10	8
P	0.1%	~~0
Block RAM's	6	2
LUT's	2990	167

In this section we compare the results of proposed architecture and conventional methods. By taking the bloom filter size as fixed, the number of hash functions is reduced and the data elements inserted is more comparatively as shown in graphs [1][2].and the false positive rate has reached to zero. Therefore by efficiently maintaining same size the false positivity is decreased and the hash collisions are mitigated due to perfect Ex-or hashing as shown the block RAM's and LUT's usage is reduced as shown in table.

© 2015, IJCSMC All Rights Reserved



Graph 1. Hash functions vs False positive rate



Graph 2. No of elements inserted vs False positive rate

7. CONCLUSION AND FUTURE WORK

In this paper, by using classic bloom filter as base design, we extend it to approximate key value storage system by analyzing its performance in storing large datasets. Key value stores has emerged as popular storage and querying billions of Key Value pairs by providing database optimization. The proposed architecture has better flexibility by obtaining negligibly smaller number of false positives while memory usage is maintained the same. Improving false positive probability provides positive impacts to variable fields which have bloom filter applications. Ex-OR hashed bloom filter is superior to normal standard bloom filter in utilization of given area, in other words by inserting more elements in a given area and by obtaining hash collision reduction rate as by eliminating redundancies. Proposed architecture provides faster key lookups as order of 25µs with lower response time and higher throughput of operations for improved responsiveness and improved accuracy of hash collisions with approximately zero false positive rate. Our future work would be using different hashing techniques for bloom filters and measurement power consumption.

REFERENCES

[1] G.Lu, Y. Nam, and D.Du, "Bloomstore:Bloom filter based memory efficient key-value store for indexing of datadeduplication on Flash"Mass storages systems and Technologies, vol 15,2012

[2] Arun. M and Prof. PS Periasamy, "Design and Implementation of string Matching system for Network Intrusion Detection Using FPGA based Bloom Filter",Journal of Computer applications Vol. 1, No.3.Dec 2008

[3]Fay Chang,Jeffray Dean,Sanjay Ghemawat, Robert E. Gruber. "Bigtable:A Distributed Storage system for Structured data"Google.inc

[4]GuiseppeDecandia, Denis Hashtorum,MadanJampani, AvinashLakshman,Werner vogles"Dynamo: Amazon's highlyAvalaible key value store" Amazon.com

[5] Cassandra, http://en.wikipedia.org/wiki/Apache_Cassandra

[6]S.Chalamalasetti,K.Lim,M. Margala, An FPGA Memcache d appliance, "International symposiyum on field programmable gate arrays"2008

[7]SisiXIong,Yanyam Yao, Qing Cao and Tiam He "Kbf:A Bloom filter for KV Store with application on approximate state machines"2014

[8] IlhanKaya,TaskinKocak, "A Low Power Lookup Technique for Multi-Hashing Network Applications", IEEE Computer Society, Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI'06)

[9] Carter, J. L. and Wegman, M., "Universal classes of hash functions", *Journal of Computer and System Sciences*, vol. 18, pp. 143-154, 1978.

[10] Silberschatz, Korth and Sudarshan "Indexing and Hashing" Database System Concepts, 6th Ed.

[11]SasuTarkoma,Christian Esteve Rothenberg and EemilLagerspetez, "Theory and Practice of Bloom Filters For distributed Systems"

[12]ChoonHoong Ding, SaranaNutanong, and RajkumarBuyya,"Peer-to-Peer Networks for Content Sharing"Grid Computing and Distributed Systems Laboratory

[13] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload Analysis Of A Large-scale Key-value Store," In Proceedings of the ACM SIGMETRICS, 2012