Advanced Engineering Forum *Online: 2012-09-26*
ISSN: 2234-991X, Vols. 6-7, pp 790-795
doi:10.4028/www.scientific.net/AEF.6-7.790

The Research of Technology of Bloom Filter Realization in Hardware

Guo Tengfei [a], Mao Jianbiao[b] ,Sun Zhigang[c]

School of Computer, National University of Defense Technology, China

[a]gtf006010@163.com, [b]mjn198812@gmail.com, [c]sunzhigang@263.net

Keywords: Bloom filter;HBBF; NetMagic platform

Abstract. Bloom filter is a space-efficient data with a certain probability of false positive . We present a reusable hardware implementation framework, define a module interface to provide users with a customize module, and introduce the constraints of hardware resources in the analysis of false positive rate against the traditional Bloom filter hardware design and analysis of the Bloom filter false positives. Finally, we make verification and analysis of our design combined with the the NetMagic platform.

Introduction

A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries. It can quickly determine whether a given data is in the given set or not. It has heavily used in the database applications since the 1970's when it was invited[1]. Recently, it was used more widely with the rapid development of networking technology. It can be used in distributed cache, P2P and Overlay network, the source routing, message routing and measurement [2]. It also plays a positive role in the research of some emerging networking architecture, e.g., the NDN[9].

The research of extension for the standard Bloom filter is also widely development. Some variants came up as a result. [3] introduces the idea of counting Bloom filter in order to overcome the problem that one easily make a insertion into a Bloom filter but cannot perform deletion with it. In a counting Bloom filter, each and every entry in it is not a single bit but instead a small counter.

In a Bloom filter, each membership query consists hashing for a set of memory address and memory accessing at these locations. Dharmapurikar et al describes a technology based on Bloom filters for detecting predefined signatures in the packet payload[4]. In the system, parallel Bloom filters were used to speedup the hashing computing. But in that situation, it fields a expensive hardware overhead and higher requirement in the I/O system. Chen et al proposes a new design of Bloom filter in which every two memory addresses are squeezed into one I/O block in the main memory [5]. In this design, the average query delay was reduced by half but leads to a increment of false positive rate.

We can see that there is no reusable model which is more flexible for the applications of Bloom filters. We proposes a Bloom filter realization scheme based on hardware which is reusable and easily implemented to overcome the expensive overhead and lack of reusability. Users may easily define Bloom filters to meet their requirement with our framework design. In this paper, we (1) propose a hardware realization framework of Bloom filters named HBBF (Hardware Based Bloom Filter) and the procession in this model; (2) provide a method for parameters selection in the situation of limited hardware resources based on NetMagic[6,7,8]; (3) investigate the account of resources in the hardware realization of Bloom filter which impacts the false positive rate.The following section describes the basic theory of Bloom filter and analysis of the relationship between the parameters in the Bloom filter. Section 3 describes the scheme. Section 4 describes the analysis of resource requirement for this scheme. Finally, the last section summarizes the contribution of our design.

Basic idea

A Bloom filter representing a set $S=\{x_1,x_2,..,x_n\}$ usually consists an array of m bits(initially all set to 0) and k independent hash functions with output range $\{1...m\}$. For each item x in S, the bits in the array are set to 1. To determine whether an element y is a member of set S or not, we hash it k times with the same hash function set and check the corresponding bits in the array. If at least one of those bits is 0, this element cannot be in the set; otherwise, the element is considered to be a set member. But using Bloom filter may yield false positive due to the conflict in hash. False positive rate can be calculated in a straightforward approach assuming the hash functions' output are perfectly random. The probability of false positive is $(1-(1-1/m)^{kn})^k$ and note that when $k=(m/n)\cdot ln2$, we have a minimum false positive rate for the given m and n [2]. We can see that these are four key parameters in Bloom filter data structure, i.e., the number of members in the set, the number of members in the Bloom filter array, the number of hash functions and the false positive rate.

The Bloom filter we want is one that has better performance and lower false positive rate. In this paper, we utilize the idea of reuse and provide users with programmable hash function interface. Our hardware realization is simple, low complexity and high expansibility. But we do not consider the issue of speedup to the Bloom filter .Of course, you can scan [4][5] for speedup variants of the Bloom filter.

Our Design

The hardware realization logic of Bloom filter in our design is illustrated below.

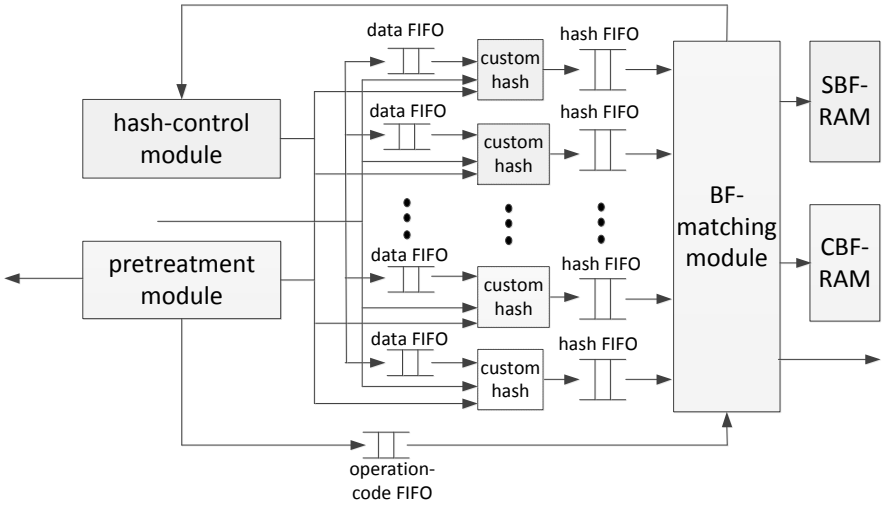


Figure 1 HBBF logical diagram

Module function description. Pretreatment module is designed to extract the key data in packets. First, It extracts the key data in the arrived packets, i.e., operating data and operating code . Then it sends the operating data to the data FIFOs and the operating codes to the operation-code FIFO. Hash-control module is designed to control the custom-hash module. It immediately sends the validate signals to all the custom-hash modules as long as it receives the feedback signal from BF-matching module. Keeping the signals for a certain time, it automatically revoked the signals,and wait for the next feedback signal. Custom-hash module is designed to calculate the memory addresses based on the data it received from pretreatment module. When the enable signal reached, each module reads the operating data from hash FIFO respectively. Then, it hash these value using the selected hash function according the mode selection signal. Finally, it sends the results to the data FIFO. The flexibility is that we provide two hash function modes, one is the default hash functions, and the other is the user defined hash functions. Our design, in this way, provides a hash function definition interface. BF-matching module is designed to process Bloom filter operations. It reads operating code to determine which operation should be performed. Then it reads each hash value from hash FIFO respectively. Following, it accesses the RAMs according to the algorithm described below . Finally, it output the result.

Operation algorithm description. There are three Bloom filter operations in the BF-matching module,that is, insertion, deletion and query. The algorithms are described below.

Algorithm 1 Insertion with Bloom filter
1：judge operation type
2：read hash values
3：**if** operating code is op_insert
4：access SBF-RAM in the positions the hash values indicated
5：add the values in those positions with 1
6：output the result insertion successful
7：**else** output the result insertion failed
8：**endif**

Algorithm 2 Query with Bloom filter
1：judge operation type
2：read hash values
3：**if** operating code is op_query
4：access SBF-RAM in the positions the hash values indicated
5：**if** all values in those positions are 1
6：output the result query successful
7：**else** output the result query failed
8：**endif**
9: **endif**

Algorithm 3 Deletion with Bloom filter
1：judge operation type
2：read hash values
3：**if** operating code is op_delete
4：access CBF-RAM in the positions the hash values indicated
5：**if** all values in those positions are more than 1
6：munis the value by 1
7：output the result deletion successful
8：**else** output the result deletion failed
9：**endif**
10: **endif**

Resource

In this section, we make a theoretical analysis for the resources requirement of HBBF hardware implementation at first. Then wo will analyze the actually requirement based on NetMagic platform. As a theoretical analysis, we show a formula to calculate the resources and the parameters are shown below.

Table1 Parameters for HBBF resource

n	number of elements in the set
m	number of members in array
k	number of hash functions
l_{data}	number of bits of operating data
l_{op}	number of bits of operating code
w_{data}	depth of data FIFO
w_s	depth of hash FIFO
w_{op}	depth of operating-code FIFO
c	counter size in CBF
r	resources requirement of HBBF

The resources requirement of HBBF can be calculated by these formulas using the parameters defined above.

$$r=m\cdot(1+c)+k\cdot(l_{data}\cdot w_{data}+\log_2 m\cdot w_s)+l_{op}\cdot w_{op} \quad (1)$$

$$k=(m/n)\cdot \ln 2 \quad (2)$$

$$f=(1-(1-1/m)^{kn})^k \quad (3)$$

Formula (1) is a straightforward approach to calculate the required resource. The total resource requirement *r* consists that array needs and that of FIFOs. Assuming a Bloom filter array has members of *m*, so the array of standard Bloom filter needs resources of *m* bits , and the counting Bloom filter needs $c \cdot m$ bits. As a result, the total array needs is $(1+c)\cdot m$. Resources data FIFOs need are $k \cdot l_{data} \cdot w_{data}$ bits. Hash FIFOs required $k \cdot log_2 m \cdot w_s$ bits and $l_{op} \cdot w_{op}$ bits are the operating code FIFO required. It is implied that the resources these FIFOs required can be computed by the depth of the FIFO multiplies the width of the data in that FIFO. Formula (2) is the relationship between the three parameters of Bloom filter in the ideal case. Formula (3) is the false positive rate analyzed in a traditional approach.

Now, we will analyze the resource requirement to implement this HBBF on the NetMagic platform. The goal that we realize HBBF on the NetMagic are making a verification to our HBBF design and setting up an experimental platform for the realization of NDN node for our future work in NDN. Therefore, we make the NDN experiment a factor when we select the parameters. of The main function of NDN node is look-up table based on names. In the NDN network, the account of data in the name space is massive in theory, but according to the theory of ziph distribution [10], the probability that we may find the name in a set is very small when the set has members of 16k. Therefore, we select the value of *n* from 16k,32k and 64k. Then, we can get a relationship below the four parameters according to formula (2) and (3) as figure 2 illustrated below:

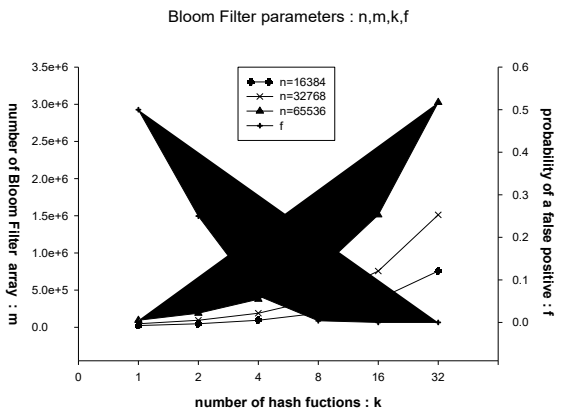
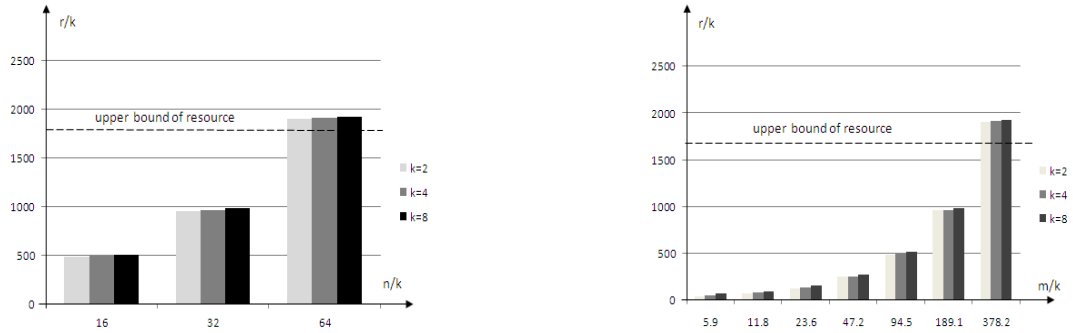


Figure 2 Relationship between four Bloom filter parameters

As selecting other parameters, we should make a trade-off .In order to improve the performance of the Bloom filter, we need to reduce *k*, i.e., the number of hash functions that will lead to a increment of the false positive probability. The three parameters *m*,*n* and *k*, in the ideal situation, are associate according to formula (2). According to the curve in figure 1, we choose 4 for the parameter *k* as well as 2 and 8 which both beside it as references. We assume that the operating data, i.e., l_{data} is 128 bits. We suggest that the operating code is 2 bits in order to meet 3 types of the Bloom filter operations. According to the practical experience, we make the depth of the FIFO as 1/4 as the width of the data in that FIFO, that is 32 bits. We use a 4 bits counter as analyzed in[3]. So, the account of resources we need to implement a Bloom filter as we make the limited hardware resources to realize the HBBF a factor to investigate is illustrated blow.



(a) (b)

Figure 3 (a)relationship between r and m (b)relationship between r and n

The dotted line represents the amount of resources the NetMagic platform can provide for users . We get a Bloom filter with our HBBF design which has around 200k members in the array and can represent a set of 64k members as well as the number of hash functions 8 which maybe a little large. chart data from the conclusion get, when set the number of elements must, as the number of resources demand hash function multiplied the number almost no too big change; When hash function when the number, the number of resources demand set number of elements with the doubling of approximate and the increase exponentially. It is reflected that our HBBF design is feasibility in practical application and Bloom filer is a space -efficiency data structure.

Conclusion

We have introduced the design of the reusable Bloom filter hardware model and provided the hash function design interface which was verified based on NetMagic platform . As we mentioned before, we will use this design in NDN networking environment to observe the effect in NDN name lookup in future work.

This work is supported by a grant from the Major State Basic Research Development Programs of China (973 Programs) (No.2009CB320503 and No.2012CB315906), and Chinese National Programs for High Technology Research and Development (863 Programs) (No.2011AA01A101)

References

[1] B.H.Bloom,"Space/time trade-offs in hash coding with allowable errors,"Communications of ACM, vol.13, no.7, pp. 422-426, July 1970

[2] Broder A, Mitzenmacher M. Network Applications of Bloom Filters: A Survey[J]. Internet Mathematics, 2004, 1(4): 485-509.

[3] Fan, L., Cao, P., Almeida, J., Broder, A.: Summary cache: A scalable wide-area web cachesharing protocol.IEEE/ACM Transactions on Networking (TON) 8 (2000) 281-293.

[4] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood,.Deep packet inspection using parallel bloom filters,. In Proceedings of High Performance Interconnects, 2003, pp. 44.51.

[5] Chen, Y., Kumar, A., Xu, J.: A New Design of Bloom Filter for Packet Inspection Speedup. In:IEEE Global Telecommunications Conference, 2007.

[6] Li Tao,et al.A Novel Packet Processing Model for Next-Generation Internet Experiment Platform-EasySwitch. The 8th China National Computer Conference (CNCC2011), Shenzhen, 2011.

[7] Taoli, Zhigang Sun, ChunboJia, Qi Su, Myungjin Lee. Using NetMagic to Observe Fine-Grained Per-Flow Latency Measurements .Proceedings of the ACM SIGCOMM 2011,Toronto, Canada, August, 2011:466-467

[8] http://www.netmagic.org

[9] Networking Named Content, Van Jacobson et al. CoNEXT'09, Rome, Italy, 2009

[10] Breslau, L., et al.: Web caching and zipf-like distributions: Evidence and implica-tions. In: In INFOCOM. pp. 126–134 (1999)