ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 3 , Issue 4 , April 2014

Parallel and Pipeline Pattern Matching Strategy forLow Power Applications

Kiruthika.T[1], Brindha.P[2]

PG Student, Department of ECE, Velalar College of Engineering and Technology, Erode, Tamilnadu, India[1]

Assistant Professor, Department of ECE, Velalar College of Engineering and Technology, Erode, Tamilnadu, India[2]

***Abstract*:** One broadly used method for representing membership of a set of items is the simple space-efficient randomized data structure known as Bloom filters. Generally the regular Bloom filter suffers in terms of power consumption and FPR (False Positive Rate). To overcome this we proposed two methods. The pipelined Bloom filter architecture for k-stages has been proposed to attain the significant power saving. The second method is the parallel Bloom filter that reduces the FPR. Further a novel B_h-sequence scheme is introduced in this pooled pipelined and parallel Bloom filter architecture to reduce the FPR. Through this method around 10%-20% of the power saving can be achieved. Bloom filters are used in network security applications such as web caches, resource routing, network monitoring.

Keywords: False Positive Rate (FPR), Multi-dimension Dynamic Bloom Filter (MDBF), Parallel Bloom Filter (PBF), Counting Bloom Filter (CBF).

I. INTRODUCTION

Now a day, there is an adequate amount of software programs are installed to guard the computer systems. By using NIDS method the malicious contents [1] such as internet worms and viruses were identified in network packets. Network intrusion detection system (NIDS) [11] scans the header of the internet packets to seem for the presence of the predefined IP address.Generally in VLSI signal processing, several outputs are computed in parallel in a clock period for parallel processing. In pipelining, it processes a single module in a clock period. There are two main advantages of using pooled architecture: high speed and low power.

A Bloom filter is an inventive randomized data structure for giving information to representing a set in order to corroborate approximate membership queries. It was discovered by Burton Bloom in 1970's [2] for large-scale network applications such as shared web caches, query routing, network monitoring, resource routing and traffic management. Bloom filters are extensively used in networking applications and to identify malicious content [1] in high speed networks. During deep packet inspection [12], this checks the payload of the packets against a set of known virus.

The bloom filter may offer better performance, if the false positive [10] does not cause major troubles. Anywhere a list or set is used, and space is a concern, a bloom filter should be considered. While use a bloom filter, consider the potential effects of false positives. Generally Dynamic bloom filters are introduced to represent dynamic sets, as well as static sets.DBF can regulate the false positive probability at a low level. Standard and dynamic bloom filters just mainly focus on the representation of single attributes instead of representing [3] multi attribute. One of the new technique Multi-dimension dynamic bloom filter (MDBF) is introduced to represent the multi attribute items. By the use of RBF (Retouched Bloom filter) the overall error rate is maintained low [7] It is expressed as a group of false positive rate and false negative rate. In RBF, the error rate [8] is made equivalent to the false positive rate of the consequent bloom filters.

In order to reduce the power consumption of bloom filters, the pipelining technique is engaged. The embracing new type of bloom filter is termed as "Pipelined Bloom Filter". Bloom filters indicate the set of 'n' patterns in a m-bit array vector. Before programming, the elements in this array are set to '0'and each signature is hashed k times by the autonomous hash functions. Each hash function locates homogeneously to a random number and that indicates a bit location in the m-bit long lookup vector, which is set to '1'.

In query stage, bloom filters computes k many hash values for an input string 'y' by utilizing the same hash functions, used in programming operation. If all the hashes locate to the bit location that are set to '1' (match), then the query string is in the set [5]. If any of the hashes locates to the bit location that is set to '0' (mismatch), then the query string is definitely not in the set.

A bloom filter not at all produce false negatives, if it decides input is a nonmember, but it may produce false positives.

The false positive probability f is estimated by,

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 3 , Issue 4 , April 2014

$$f = \left(1\text{-}e^{\frac{-nk}{m}}\right)^k \quad (1)$$

Where, n is the number of patterns programmed into the bloom filter, k is the number of hash functions used to realize the bloom filter and m is the length of the lookup vector.The choice of m>n, to diminish the false positive probability. For a fixed value of m/n, k must be large to minimize the false positive probability. The number of hash functions that minimizes the FPR is,

$$K = \left(\frac{m}{n}\right) \ln 2 \quad (2)$$

The power consumption of the regular bloom filter is a summation of the power consumption of the each of the hash functions, P_{Hi}, P_L, P_{AND}.

$$P_{BFregular} = \sum_{i=1}^{k} (P_{Hi} + P_L) + P_{AND} \quad (3)$$

Here, P_{AND} is ignored. Since it is minimal compared to the power used by the hash functions and also presume that the lookup power over a m-bit vector is in the order of steady for each index designed by any of the hash functions. As hash functions with the identical number of input bits will be implemented with the similar number of components and will consume approximately the same amount of power. So we can write the power consumption of a regular bloom filter as follows,

$$P_{BF\ regular} = \sum_{i=1}^{k} (P_{Hi} + P_L) = k.(P_H + P_L) \quad (4)$$

However there is a critical challenge, to the representation and queries for items that are having multidimensional attributes.So we proposed one of the new technique MDDBFs (Multi dimension dynamic bloom filters) to represent items with multiple attributes. The probability of false positives may increase, if the MDDBF approach [4] lacks a way to verify the dependency of multiple attributes of items. Nevertheless, the MDDBF approach lacks a way to confirm the dependency of multiple properties of an item, which may increase the probability of false positives. Through by the parallel bloom filter with a hash table, this supports the representation of items with multiple attributes. By using parallel-pipelined bloom filter design [6], multiple strings can be queried and that can reduce power consumption along with improving the throughput. In pipelined design, query string is estimated at one of the pipeline stages, remaining stages are in 'idle'. This technique delivers a greater amount of reduction in power consumption. But it suffers computation latency. By using multiple hash functions, multiple query strings are concurrently evaluated in parallel pipelined bloom filter design.

II. PIPELINED BLOOM FILTER

Basically, a pipeline bloom filter consists of several groups of hash functions that are utilized in different stages. While the number of hash functions required to diminish the false positive probability of a bloom filter is large, it is superior, in terms of power, to implement these hash functions in a pipelined style. We call this new type of bloom filters pipelined bloom filter [7].

Here hash functions are arranged in pipelined manner, to reduce the power consumption. Essentially it consists of two groups of hash functions.

1. First stage, forever computes the hash values.
2. The second stages merely compute the hash values, if there is any match between the input and the patterns.

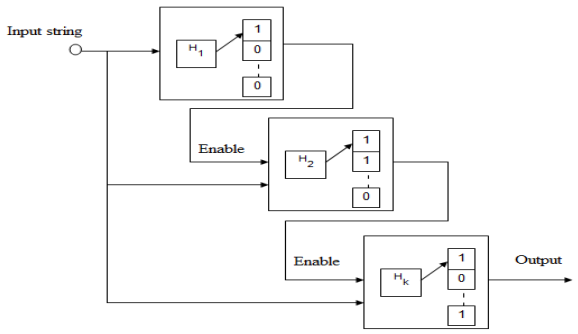


Fig.1 Fully pipelined bloom filter

The merits of using pipelined bloom filter techniques is that, if the first stage identifies a match, there is no need to use the second stage to decide whether input string is a part of the signature set. This is possible only because the Bloom Filter is free from False negative rate. The shortcoming with this is power consumption.

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 3 , Issue 4 , April 2014

Fully pipelined bloom filter design is the remedy for this power consumption problem. The building of fully pipelined bloom filter is displayed in Fig 1. This architecture has the same number of hash functions as the regular bloom filter. Therefore the false positive probability is also same. In the inquiry stage, the initial hash functionh1, is fed by a new inquiry sequence every cycle.

An inquiry string has progressed to the next stage only when the prior hash function produces a match. Now every hash module consists of hash function and a m/k bit lookup array. The inquiry string progress to the next stage, if previous hash functions fail to match the signature. When the inquiry string proceeds to the next stages, the design increases the latency.

Analysis

Each hash function coefficients are randomly selected in the range of 1 to m. The probability that the bit is unset, after all the signatures are programmed by using k-many independent hash functions are α.

$$\alpha = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{\frac{-kn}{m}} \quad (for\ large\ m) \qquad (5)$$

Here (1-1/m) is the probability that the bit is unset behind a single hash value computation with a single signature.

The probability that any one of the bit is set is,

$$(1 - \alpha) \approx 1 - e^{\frac{-kn}{m}} \qquad (6)$$

In order for the first stage to generate a match, the bits indexed by all r of the free hit and miss hash functions should be set.

So, 'P' is the match possibility of the initial stage that is indicated as,

$$P = \prod_{i=1}^{r} (1 - \alpha) = (1 - \alpha)^r \qquad (7)$$

So, '1-P' is the inequality possibility of the initial phase,

$$1 - p = 1 - \left(1 - e^{\frac{-kn}{m}}\right)^r \qquad (8)$$

By means of a probability of (1-p) the initial stage of the hash functions in the pipelined bloom filter will cause a mismatch. Or else, the initial stage produces a match, then the next stage is used to evaluate the input with the signature required.Therefore the power consumption of a pipelined bloom filter is given by,

$$P_{BFpipeline} = P_{1st\text{-}stage} + P\{match\} \times P_{2nd\text{-}stage}$$

$$P_{BFpipeline} = \sum_{i=1}^{r} (P_{Hi} + P_L) + p \times \sum_{j=r+1}^{k} (P_{Hj} + P_L) + P_{AND} \qquad (9)$$

Again,

P_{AND} can be neglected. The power consumption of a pipelined bloom filter is given by,

$$P_{BF\ Pipeline} = \sum_{i=1}^{r} (P_{Hi} + P_L) + \left(1 - e^{\frac{-kn}{m}}\right)^r \times \sum_{j=r+1}^{k} (P_{Hj} + P_L)$$

$$= r.(P_H + P_L) + \left(1 - e^{\frac{-kn}{m}}\right)^r (k-r)(P_H + P_L) \qquad (10)$$

The power saving ratio, PSR, in a single Bloom filter by deploying pipelining technique can be calculated as

$$PSR = \frac{(P_{regular} - P_{pipelined})}{(P_{regular})} \qquad (11)$$

The average power saving ratio, PSR, is specified by

$$PSR = \frac{\left(k \times A - \left[r + \left(1 - e^{\frac{-kn}{m}}\right)^r \times (k - r)\right] \times A\right)}{k \times A} \qquad (12)$$

Where, A = (P_H + P_L) which is the power consumption of a particular hash function with a only lookup operation.

Finally average power saving ratio PSR is given by,

$$PSR = \frac{k - r + (r - k)\left(1 - e^{\frac{-kn}{m}}\right)^r}{k} \qquad (13)$$

III. PARALLEL BLOOM FILTER

An intuitive approach to representing multiattribute items can concatenate various attributes into a single-attribute array to

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 3 , Issue 4 , April 2014

be stored in a standard bloom filter (SBF). Nevertheless, such approach may offer delay inquiry replies to users if multiple attributes have dissimilar formats. In fact, it takes a extended time to get the hashed result for a single but long attribute array.However, standard form in an SBF is fundamentally a compressed representation, limiting its rich inquiry services.

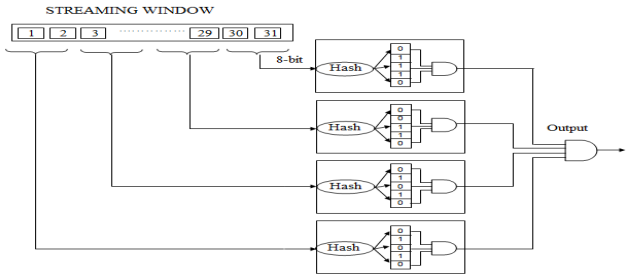
In real-world applications, many inquiry requests cannot provide exact and absolute descriptions of queried items, which bound the usage of SBFs for queries of partial attributes. In this paper we present an approach to the space well-organized representation of multiattribute items. The future approach utilize data structures to carry out rapid but exact membership queries and achieve space savings. So we describe data structures in three phases:

1. A Parallel Bloom filter (PBF) structures,
2. PBF with a Hash Table (PBF-HT) and
3. PBF with a Bloom filter (PBF-BF).

Fig 2.Shows the Architecture of Parallel bloom filter.This structure takes each 8-bit value from streaming window for each individual hash module and produce the corresponding output. Finally that outputs are ANDed and get the final single output. False positive rate is highly minimized while using this parallel bloom filter method, but hardware is slightly increases.

IV. COMBINED PIPELINED AND PARALLEL BLOOM FILTER

Combined Pipelined and parallel bloom filter is designed to satisfy the High speed and Low power requirements. A general bloom filter consists of multiple hash functions and a lookup array. The lookup array which is m-bit wide and to estimate a query string operation, specific k bits in the lookup array are inspected [9].If all bits locate to 1, the query string is member of the set. There is a possible that a non-member query string may be evaluated as a member of the signatures, which is false positive rate (FPR).If number of hash functions increases, the FPR(False positive rate) is reduced. It introduces significant amount of power consumption. By using combined parallel and pipelined bloom filter design[6], multiple query strings to be filtered in parallel and can reduce power consumption along with improving the throughout.



In pipelined design, query string is estimated in one of the pipelined stages, remaining stages are in 'idle'. This technique having greater amount of reduction in power consumption.By using multiple hash functions, multiple query strings are concurrently evaluated in parallel pipelined bloom filter design and compared to regular bloom filter greater development in throughput.

V. B_h-SEQUENCE METHOD

B_h-sequence

B_h-sequence [10] is a set of integers and are used in network applications.

Definition($B_h$sequence): Let (A, +) be an abelian set. Let $D = \{v_1, v_2, ..., v_l\} \subseteq$ A be a sequence of elements of A. then D is a B_h series over A if all the sums $vi_1 + vi_2 + ... + vi_h$ with $1 \le i_1 \le .... \le i_h \le l$ are distinct.

Example 1: Let A= Z and D = $\{v_1, v_2, v_3, v_4\}$ = {1, 4, 8, 9}; $\subseteq$ A. We can see that all the 15 sums of 4 elements are distinct: 1+1+1=3, 1+1+4=6, 1+1+8=10, 1+1+9=11, 1+4+4=9, 1+4+8=13, 1+4+9=14, 4+4+4=12, 4+4+8=16, 4+4+9=17, 1+8+8=17, 1+8+9=18, 8+8+8=24, 8+8+9=25, 9+9+9=27. Here 4+4+9 = 17 = 1+8+8, these two sums are producing same value. So this is not a B_h-sequence. Therefore, D is a not a B_3 sequence. If we choose D={1,6,8}, then check the possibilities for B_2-sequence, 1+1=2, 1+6=7, 1+8=9, 6+6=12, 6+8=14, 8+8=16. Here the entire sums produce different values. Therefore D is a B_2 sequence.

False Positive Rate of B_h-sequence method

We now present the false positive rate of B_h Bloom. As generally assumed in the reference [7] we assume that the hash functions plot items to random numbers equally spread over their given range.

Theorem: The false positive rate of B_h Bloom is given by,

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 3 , Issue 4 , April 2014

$$FPR = \left(1-\sum_{j=0}^{h}\binom{nk}{j}\left(\frac{l-1}{lm}\right)\left(1-\frac{1}{m}\right)^{nk-j}\right)^{k} \quad (14)$$

Proof:

Let X indicate the number of elements hashed into an entry. The probability of the event X=j is given by,

$$\Pr(X=j) = \binom{nk}{j}\left(\frac{1}{m}\right)^{j}\left(1-\frac{1}{m}\right)^{nk-j} \quad (15)$$

A particular value of the B_h sequence is not used, When accurately X=j elements are hashed into an entry.

$$FPR = \left(1-\sum_{j=0}^{h}\Pr(X=j)\left(1-\frac{1}{l}\right)^{j}\right)^{k} \quad (16)$$

This novel B_h-sequence method is introduced in pipelining technique to reduce the FPR.

VI. EXPERIMENTAL RESULTS

TABLE 1

FALSE POSITIVE PROBABILITY

PARAMETERS		THEORETICAL		EXPERIMENTAL	
m/n value	k value	Counting Bloom Filter	CBF With B_h Sequence	Counting Bloom Filter	CBF With B_h Sequence
35	24.260	0.0432	0.0401	0.0324	0.0524
40	27.725	0.0356	0.0445	0.0421	0.0324
45	31.19	0.0389	0.0489	0.0258	0.0324
50	35	0.0265	0.0078	0.0115	0.0421
55	38.123	0.0043	0.0056	0.0025	0.0152
60	41.58	0.0023	0.0003	0.0019	0.0013
			0.0003		0.0011
			0.0002		
			0.0058		
			0.0000		
			0.0023		
			0.0000		
			0.0024		
			0.0000		
			0.0018		
			0.0000		
			0.0031		

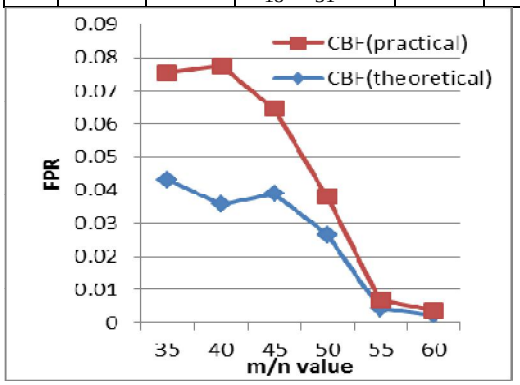


Fig .3 Comparison of false positive probability for CBF

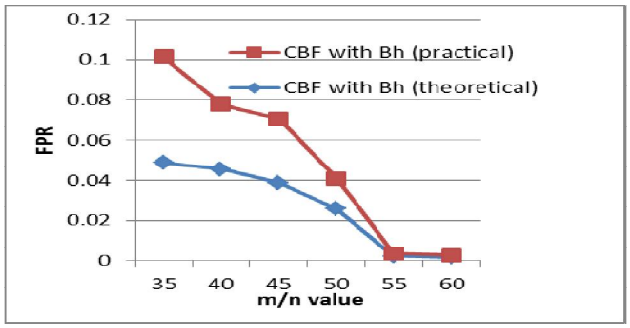


Fig.4 Comparison of false positive probability for CBF with B_h-sequence

TABLE 2

POWER SAVING RATIO

m/n value	r value	k value	PSR for Pipelined Bloom Filter	PSR for Parallel Bloom Filter

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 3 , Issue 4 , April 2014

20	10	13.86	0.312	0.489
40	20	27.72	0.289	0.324
60	30	41.58	0.232	0.314
80	40	55.45	0.202	0.295
100	50	69.31	0.182	0.243

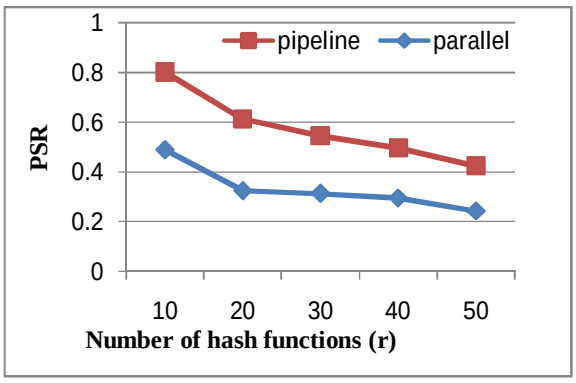


Fig .5Power saving ratio of pipelined and parallel bloom filter

From table 1, shows the false positive rate of counting Bloom filter and CBF with B_h-sequence are depending on m/n and k values. The choice of m should be greater than n, to diminish the false positive probability. For a fixed value of m/n, k must be larger to minimize the false positive probability. For example, substituting m/n=35 in (2) equation, the corresponding k=24.260 is calculated. Substituting k value into (1) equation, the corresponding false positive probability is estimated for counting Bloom filter. From this analysis, bloom filter with B_h-sequence will produce a less false positive probability compared with normal counting bloom filter. From this analysis, by using these methods false positive probability is highly reduced. From table 2 the PSR of pipelined BF is higher than the parallel BF.

VII. CONCLUSION

In this paper, we proposed a pipelined Bloom filter method to achieve greater power saving.To attain the accurate signature detection multi hash functions have to be included. The number of hash functions required to minimize the false positive probability of a bloom filter is large, thus power consumption is more. To reduce this, the hash functions are implemented in a pipelined manner, in which hash functions are made as stages and detection of signature is based on the previous stage output.Through this method 10%-20% of the power saving can be achieved.The use of parallel Bloom filter that diminish the FPR.Compared with traditional bloom filter, the use of Pipelined bloom filter with B_h-sequence method there is a small reduction in FPR. But for parallel Bloom filter with B_h-sequence method significant reduction in FPR. The Power saving ratio of pipelined bloom filter is high compared with parallel bloom filter.

ACKNOWLEDGEMENT

The authors acknowledge the contributions of the students, faculty of Velalar College of Engineering and Technology for helping in the design of test circuitry, and for tool support. The authors also thank the anonymous reviewers for their thoughtful comments that helped to improve this paper**.** The authors would like to thank the anonymous reviewers for their constructive critique from which
this paper greatly benefited.

REFERENCES

[1] Kaya.I and Kocak.T (2006), "Low-power Bloom filter architecture for deep packet inspection," *IEEE Commun. Lett.*, vol. 10, no. 3, pp. 210-212.

[2] Broder.A and Mitzenmacher.M (2005), "Network applications of Bloom filters: a survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485-509.

[3] Chen.H, Guo.D, Luo.X and Wu.J, (2006), "Theory and Network Application of Dynamic Bloom Filters," *Proc. IEEE INFOCOM*.

[4] Bin Xiao and Yu Hua (2010),"Using Parallel Bloom Filters for Multiattribute Representation on Network Services," *IEEE transactions on parallel and distributed systems*, vol. 21, No.1.

[5] Kocak.Tand Paynter.M (2008), "Fully pipelined Bloom filter architecture," *IEEE Communications letters*, vol.12, No.11.

[6] Deokhokim,Doohwan oh (2012) "Design of power-efficient parallel pipelined bloom filter, " *Electronic letters* ,vol.48, No. 7.

[7] Baynat.B, Donnet.B and Friedman.T (2006), "Retouched Bloom Filters: Allowing Networked Applications to Trade Off Selected False Positives Against False Negatives," *Proc. Int'l Conf. Emerging Networking Experiments and Technologies (CoNEXT).*

[8] Bloom.B (1970), "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no.7, pp. 422-426.

[9] Hill.M, Sanchez.D, Sankaralingam.K and Yen.L (2007), "Implementing signatures for transactional memory," *in Proc. 40th IEEE/ACM Int'lSymp. on Micro architecture, Chicago, IL.*

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 3 , Issue 4 , April 2014

[10] OriRottenstreich and Isaac Keslassy (2013), "The variable increment counting bloom filter", *IEEE /ACM transactions on networking.*

[11] Attig.M, Dharmapurikar.S and Lockwood.J (2012) "Design and Implementation of a String Matching System for Network Intrusion Detection using FPGA based Bloom Filters", *IEEE transactions on VLSI systems*,vol.6.

[12] Bloom.B (1970),"Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422-426.

[13] Dharmapurikar.S, Krishnamurthy.P, Lockwood.J.W and Sproull.T.S (2004), "Deep packet inspection using parallel Bloom filters," *IEEE Micro*, vol. 24, no. 1, pp. 52-61.

[14] Dharmapurikar.S, Krishnamurthy.P and Taylor.D.E(2003), "Longest Prefix Matching Using Bloom Filters," *Proc. ACM SIGCOMM.*

[15] Saar.C and Yossi.M, (2003), "Spectral Bloom Filters," *Proc. ACM SIGMOD.*