International Journal of Computer Applications and Technology (2278 – 8298)
Volume 1– Issue 1, 2012, 25-29

International Journal of Computer Applications and Technology
journal homepage: www.ijcat.com

BLOOM FILTERS & THEIR APPLICATIONS

Saibal K. Pal
Defence R & D Organization
SAG, Metcalfe House
Delhi, India

Puneet Sardana
Department of Computer Science
University of Delhi
Delhi, India

Abstract: A Bloom Filter (BF) is a data structure suitable for performing set membership queries very efficiently. A Standard Bloom Filter representing a set of n elements is generated by an array of m bits and uses k independent hash functions. Bloom Filters have some attractive properties including low storage requirement, fast membership checking and no false negatives. False positives are possible but their probability may be controlled and significantly lowered depending upon the application requirements. There are many variants of the standard Bloom Filter – counting BF, variable increment BF, compressed BF, scalable BF, generalized BF, stable BF and Bloomier Filter. Bloom Filters are increasingly finding applications in fast and approximate search, encrypted search in the cloud, routing and controlling of network traffic, network intrusion detection and differential database and file updating. This paper explores the typical properties of Bloom Filters, their variants and their suitability for use in present day applications.

Keywords: Bloom Filter, Variants, Set Membership, Hashing and Encrypted Search.

1. INTRODUCTION

A Bloom Filter is a space efficient probabilistic data structure which is used to represent a set and perform membership queries [1] i.e. to query whether an element is a member of the set or not. The Bloom Filter data structure was introduced by Burton H. Bloom [2] in 1970. A Bloom Filter occupies negligible space compared to the entire set. Space saving comes at the cost of false positives but this drawback does not affect the processing of information if the probability of an error is made sufficiently low. Bloom Filters typically find applications in situations that involve determining membership of an element for a sufficiently large set in small amount of time. Today, Bloom Filters are used in wide variety of applications including spell checking, network traffic routing and monitoring, database search, differential file updating, distributed network caches, and textual analysis. In this paper we will describe bloom filter, its variants and its applications in different areas of computer science.

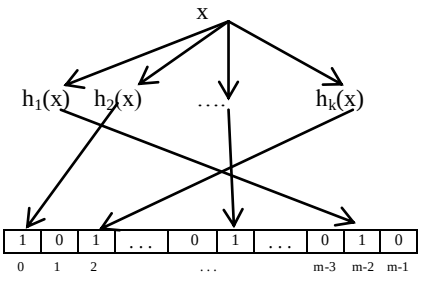

Fig. 1 Set Operation in a Standard Bloom Filter

1.1 Standard Bloom Filter

A Bloom Filter for representing a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements is described by an array of m bits, initially all set to 0. A Bloom Filter uses k independent hash functions $h_1, h_2, \ldots, h_k$ with range $\{1, \ldots, m\}$. Each hash function maps every item to some random number over the range $\{1,\ldots,m\}$. For each element $x \in S$, the bits $h_i(x)$ are set to 1 for $1 \leq i \leq k$.

To check if an item y is in S, we check whether all $h_i(y)$ are set to 1. If any of $h_i(y)$ is 0 then clearly y does not belong to S. If all $h_i(y)$ are set to 1 then y may belong to S.

Following are the properties of a Standard Bloom Filter:

• The amount of space needed to store the Bloom Filter [3] is very small as compared to the entire set.

• The time needed to check whether an element is present or not is independent of the number of elements present in the set.

• False negatives are not possible. False positives are possible but their probability can be significantly lowered.

• Bloom Filters can be easily halved in size allowing applications to shrink a Bloom Filter.

• Bloom Filters can also be used to approximate the intersection between two sets.

• If two Bloom Filters represent sets S1 and S2 with same number of bits and same number of hash functions then a Bloom Filter representing the union of these two sets can be obtained by taking OR of the two bit-vectors of the original Bloom filters [4].

The remaining part of the paper is organized as follows. Section 2 describes different variants of the bloom filter. Different applications of bloom filters in area of approximate / encrypted search, network security and database updating are

explained in Section 3. Concluding remarks are mentioned in Section 4.

2. VARIANTS OF BLOOM FILTER

In this section we explore and describe variants of Bloom Filter [5] built on the Standard Bloom Filter data structure.

2.1 Counting Bloom Filter

The Standard Bloom Filter works fine when the members of the set do not change over time. Addition of elements only requires hashing the additional item and setting the corresponding bit locations in the array. However, deletion is not possible in the Standard Bloom Filter since it will require setting 0's in the array to already set 1's that was result of hashing another item which is still a member of the set. To overcome this deficiency of Standard Bloom Filter, Fan et al. [6] introduced the idea of Counting Bloom Filter. In Counting Bloom Filter, bits of the array are replaced by a small counter. When an element is inserted, the corresponding counters are incremented; when the element is deleted, the corresponding counters are decremented. The value of the counter gives the number of items hashed to it. Since each counter size is limited, the n-bit counter will overflow if it reaches a value of 2^n. The figure below shows the structure and set operation in a Counting Bloom Filter. Analysis carried out by Fan et al shows that a 4-bit counter is adequate for most applications.

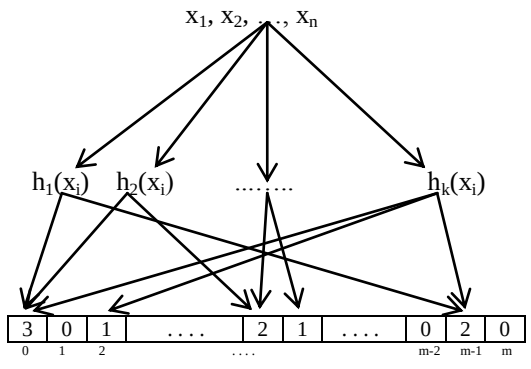

Fig. 2 Set Operation in a Counting Bloom Filter

2.2 Variable Increment Bloom Filter

The Variable Increment Counting Bloom Filter (VI – Bloom) [7] is a generalization of the Counting Bloom Filter that uses variable increments to update each entry. In this structure, a set of possible variable increments are defined. For each counter update by an element we hash the element into the variable increment set and use it to increment the counter. Similarly, to delete an element we decrement by its hashed value in the variable increment set. To determine if an element is part of the set, we check in each of its counters if its hashed value in the variable increment set could be part of the sum. If this be the case in at least one counter the element definitely does not belong to the set. Otherwise, the element may belong to the set with some probability of false positive. VI – Bloom can be used in Approximate Concurrent State Machine [8], Counter Braids [9] and Fingerprint-based schemes.

2.3 Compressed Bloom Filter

Using a larger but sparser Bloom Filter can yield the same false positive rate with a smaller number of transmitted bits. The resulting bloom filter is called a Compressed Bloom Filter [10]. By using Compressed Bloom Filters networking protocols reduce the number of bits broadcast, the false positive rate and the amount of computation per look up. Costs involved are larger computation time for compression and decompression.

2.4 Scalable Bloom Filter

A Scalable Bloom Filters consist of two or more Standard Bloom Filters, allowing arbitrary growth of the set being represented. When one Bloom Filter gets filled due to the limit on the fill ratio, a new filter is added. Querying an element involves testing the presence in each filter. Each successive bloom filter is created with a tighter maximum error probability on a geometric progression [11].

2.5 Generalized Bloom Filter

Generalized Bloom Filter [12] uses hash functions that can set as well as reset bits. In Generalized Bloom Filter, the initial value of the bits of the array is not restricted to zero anymore. For each element $x_i \in S$ bits corresponding to the positions $h_1(x_i)$, $h_2(x_i)$, … , $h_k(x_i)$ are set and the bits corresponding to the positions $g_1(x_i)$, $g_2(x_i)$,…, $g_k(x_i)$ are reset. In case of collision between function h_i and g_i the resulting bit is always reset. To check if element belongs to the set we check whether bits corresponding to h_i are all set and bits corresponding to g_i are all reset. If at least one bit is inverted then the element does not belong to the set with high probability. If no bit is inverted, then the element belong to set with high probability.

2.6 Bloomier Filters

Bloomier Filters [13] associate a value with each element that had been inserted thereby implementing an associative array. These structures achieve a small space overhead by accepting a small probability of false positives. In this type of Bloom Filter, a false positive is defined as returning a result when the key is not in the map. The map will never return the wrong value for a key that is in the map.

2.7 Stable Bloom Filter

This variant of Bloom Filter is particularly useful in data streaming applications. In these applications when more and more elements arrive, the number of 1's in the array of bloom filter will increase significantly, finally reaching the limit where every distinct element is reported as duplicate indicating that bloom filter can no longer be used. In Stable Bloom Filters [14], this state is avoided by eviction of some information. In this approach a random deletion operation is incorporated in the Bloom Filter so that it does not exceed its capacity.

3. APPLICATIONS OF BLOOM FILTER

3.1 General Applications

3.1.1 Spell Checkers

Bloom Filters are particularly useful in spell checking software. They are used to determine if the word is a valid word in its language. This is done by creating Bloom Filter of all possible words of that language and checking a candidate word against that Bloom Filter. Suggested corrections are generated by making all single substitutions in rejected words and then checking if these results are members of the set [15].

3.1.2 Longest Prefix Matching

Bloom Filters are used for longest prefix matching algorithms [16] as these are typically used for efficient exact match searches. The basic idea is to create a hash table consisting of prefixes of various lengths that have to be potentially matched against a given string with the goal of finding the longest possible match. By using Bloom Filter one can avoid

unnecessary look up into a hash table when the corresponding prefix does not exist in the table.

3.1.3 Refining Web Search Results

Bloom Filters are extremely useful in refining search results [17] returned by search engines. Most of the top search results returned by search engines contain similar contents. This technique involves removing or grouping all near-duplicate documents in the results presented to the user. Bloom Filter is also used for similarity detection of text documents. For finding similar documents, Bloom Filters are compared by using bit wise AND operation. In case the two documents share large number of 1's after applying bit-wise AND to their bloom filter, the documents are assumed to be similar.

3.2 Networking Applications

3.2.1 Routing

If the network is in the form of a rooted tree with nodes holding resources and a node receives a request for resource, it checks its unified list to ascertain if it has a way of routing that request to the resource [18]. False positives in this cause may forward the routing request to an incorrect path. In such a case backtracking of the tree is necessary. Another similar application needs to verify if the requested file has a replica nearby and in such cases the request may be routed efficiently along the shortest path [19, 20]. Each node in the network keeps an array of Bloom Filter for each adjacent edge i.e. The k^{th} Bloom Filter in the array keeps track of the files reachable via k hops from the node in the network.

Bloom Filters are also used for geographic routing system for mobile computers [21]. In this scheme each node contains a Bloom Filter representing the list of mobile hosts reachable through itself or through its three siblings at each level.

3.2.2 Loop Prevention

Normally, packets trapped in the network loop are detected using the IP Time-To-Live field but these are not of much help if the loops are small. A small Bloom Filter can be used which can be carried in the packet header and which keeps track of the set of nodes visited [22]. Each node has a mask that can be ORed into the Bloom Filter as its passes; if the filter does not change there is a loop.

3.2.3 IP Traceback and IP Multicast

Bloom filter is also used to trace the route that a packet traversed in a network [23]. Bloom Filters reduce the amount of information [24] that needs to be stored in order to summarize the set of packets seen. A router mistakenly identifying a packet as having been seen would be treated as a false positive.

Bloom filters are also used as alternative of interface lists that the router associates with multicast addresses to send packets through a multicast tree [25]. There can be Bloom Filter of addresses associated with each interface. When a packet with multicast addresses arrives on one interface, the Bloom Filters of all other interfaces are queried to check if packets with that address should be forwarded along that interface. This helps in significant space savings.

3.2.4 Network Traffic

Bloom filters are widely used to reduce network traffic. Bloom filters are used in caching proxy servers [26] on the World Wide Web (WWW). Bloom filters are used in Web caches to efficiently determine the existence of an object in cache. Use of web caches help to reduce the network traffic. Bloom filter are also used as cache digest. A cache digest contains information of all cache keys with lookup capability. By checking a neighbor cache, a cache can determine with certainty if a neighboring cache does not hold a given object [27]. This allows in reduction of the cache directory size while keeping the number of collisions low.

Bloom Filters find applications in network traffic measurement and detection of heavy flows inside a router [28]. The basic idea is to hash each packet entering into the Bloom Filter. A counter is associated with each location in the Bloom Filter that records the number of packet bytes that have traversed the router associated with that location. The counter is incremented by the number of bytes in the packet. If a minimum count associated with a packet is above some threshold, the corresponding flow is marked as heavy flow.

3.3 Applications in Security & Database Management

3.3.1 Intrusion Detection

Network Intrusion Detection and Prevention Systems (IDS/IPS) use string matching to scan Internet packets for malicious content. Bloom Filters are particularly useful for searching large number of strings efficiently. The basic idea is to find substrings (or commonly known as signatures) at high speed [29, 30]. A common approach is to separate signatures by length and use Bloom Filter for each length allowing parallel processing. If the Bloom Filter detects a match, a hash table is queried to determine if exact match has occurred. If the queried signature is exact match, the malicious content can be blocked and the network administrator is informed. Google Chrome uses Bloom Filters to make preliminary decision whether a particular web site is malicious or safe. Bloom filters are also used in virus scanning [31], worm detection [32], Denial of Service (DoS) prevention [33] and network forensics [34].

3.3.2 Encrypted Search

Bloom filters are extremely useful for searching in encrypted text. At the client end, user first creates the Bloom Filter of the document, encrypts the document using an encryption algorithm and then sends both the encrypted document as well as its corresponding Bloom Filter to the server. When the client needs to search the document, it sends keyword to the server and the server checks the document Bloom Filter for presence of the keyword. If presence of the keyword is established, the encrypted document is returned to the client which is decrypted with the key (used earlier to encrypt the document).

3.3.3 Database Applications

Bloom Filters have been frequently used for management of databases. Bloom Filters were used to estimate the size of joins in databases [35, 36] and to speed up semi-join operations. This is particularly useful in distributed databases. In these applications, one host sends the other host information in the form of Bloom Filter to reduce the overall communication load between two hosts.

Bloom filters can also be used to maintain differential files [37]. A differential file keeps track of all changes to a database that occurred during the day or within a specific time period. Instead of keeping a list of all records that are being changed, one can replace this list with Bloom Filters of the records that have been changed.

4. CONCLUSION

The significance of Bloom Filters has been highlighted in this paper. Variants of Standard Bloom Filter were explored, which have been modified according to the requirement of different applications. We have also explained various applications of Bloom Filters. This simple data structure is gaining significance particularly for applications related to searching of documents, databases and encrypted content on the cloud. Applications related to network traffic management, database management and cloud security are also being addressed using Bloom Filters. The Standard Bloom Filter may be modified according to the needs of the application so that more power can be derived from this data structure. In the future, we are interested in applications related to cloud security and encrypted search and would like to modify this data structure to make it more suitable for these applications.

5. REFERENCES

[1] Peter Brass, Advanced Data Structures, Cambridge University Press, 2008, pp. 402-405.

[2] Burton H. Bloom, Space/time trade-offs in Hash Coding with Allowable Errors, Communications of the ACM, Volume 13, Issue 7, 1970, http://portal.acm.org/citation.cfm?doid=362686.362692.

[3] Jing Chi, Research and Application on Bloom Filter, Applied Computing, Computer Science and Advanced Communication, First International Conference on Future Computer and Communication, FCC 2009, China, June 2009, pp. 30-33.

[4] Andrei Broder and Michael Mitzenmacher, Network Applications of Bloom Filters, Internet Mathematics Vol. I, pp. 492-505.

[5] Graham Cormode and Marina Thottan, Algorithms for Next Generation Networks, Springer, 2010, pp. 185-189.

[6] L. Fan, P. Cao, J. Almeida and A. Z. Broder, Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. IEEE/ACM Transactions on Networking, 2000.

[7] Ori Rottenstreich and Issac Keslassy, The Variable-Increment Counting Bloom Filter, Technion, Israel.

[8] F. Bonomi, M. Mitzenmacher, R. Panigraphy, S. Singh and G. Varghese, Beyond Bloom Filters: from Approximate Membership Checks to Approximate State Machines. SIGCOMM, 2006.

[9] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, Counter Braids: a Novel Counter Architecture for Per-slow measurement, SISMETRICS, 2008.

[10] Michael Mitzenmacher, Compressed Bloom Filters, IEEE/ACM, Transactions on Networking, 2002, pp. 604-612.

[11] Almeida, Paulo; Baquero, Carlos; Preguica, Nuno; Hutchison, David (2007), Scalable Bloom Filters, Information Processing Letters, pp.255–261.

[12] Rafael Laufer, Pedro B. Velloso, and Otto Carlos M. B. Duarte, A Generalized Bloom Filter to Secure Distributed Network Applications, Computer Networks, vol. 55, no. 8, pp. 1804-1819, June 2011.

[13] Chazelle, Bernard; Kilian, Joe; Rubinfeld, Ronitt; Tal, Ayellet, The Bloomier Filter: an Efficient Data Structure for Static Support Lookup Tables, Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 30–39, 2004.

[14] Deng, Fan; Rafiei, Davood , Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters, Proceedings of the ACM SIGMOD Conference, pp. 25–36, 2006.

[15] James K. Mullin and Daniel J. Margoliash, A tale of three spelling checkers, Software, Practice and Experience, pp. 625- 630, June 1990.

[16] S. Dharmapurikar, P. Krishnamurthy and D.E. Taylor. Longest prefix matching using Bloom Filters, IEEE/ACM Transactions on Networks, pp. 397-409, 2006.

[17] Navendu Jain, Mike Dahlin and Renu Tewari, Using Bloom Filters to Refine Web Search Results, Eighth International Workshop on the Web and Databases, 2005.

[18] S. Czerwinski, B. Y. Zhao, T. Hodes, A. D. Joseph and R. Katz. An Architecture for a Secure Service Discovery Service. In Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp. 24-35, ACM Press, 1999.

[19] S. C. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Volume 3, pp. 1248-1257, IEEE Computer Society, 2002.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. ACM SIGCOMM Computer Communication Review, Proceedings of the 2001 SIGCOMM Conference, 2001.

[21] P. Hsiao, Geographical Region Summary Service for Geographical Routing. Mobile Computing and Communications Review, 2001, pp. 25-39.

[22] A. Whitaker and D. Wetherall. Forwarding without Loops in Icarus. In Proceedings of the Fifth IEEE Conference on Open Architectures and Network Programming (OPENARCH), pp. 63-75, Los Alamitos, CA, IEEE Computer Society, 2002.

[23] R. P. Laufer, P. B. Velloso, D. d. O. Cunha, I. M. Moraes, M. D. D. Bicudo, M. D. D. Moreira, O. C. M. B. Duarte, Towards stateless single-packet IP traceback, Proceedings of the 32nd IEEE Conference on Local Computer Networks, IEEE Computer Society, Washington, DC, USA, 2007, pp. 548–555.

[24] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-Based IP Traceback. ACM SIGCOMM Computer Communication Review, Proceedings of the 2001 SIGCOMM Conference, (2001).

[25] B. Gronvall, Scalable Multicast Forwarding, Computer Communication Review 32, 2002.

[26] Jia Wang. A survey of web caching schemes for the internet. ACM SIG-COMM Computer Communication Review, 1999.

[27] James Blustein and Amal El- Maazawi, Bloom Filters – A Tutorial, Analysis, and Survey, 2002.

[28] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. ACM SIGCOMM Computer Communication Review, Proceedings of the 2002 SIGCOMM Conference, 2002, pp. 323-336.

[29] S. Dharmapurikar, P. Krishnamurthy, T.S. Sproull and J.W. Lockwood. Deep packet inspection using Parallel Bloom Filters, IEEE Micro, pp. 52-61, 2004.

[30] S. Dharmapurikar and J.W. Lockwood, Fast and Scalable Pattern Matching for Network Intrusion Detection Systems, IEEE Journal on Selected Areas in Communications, 2006.

[31] O. Erdogan and P. Cao, Hash-AV: Fast Virus Signature Scanning by Cache-resident Filters, in IEEE Globecom 2005, St Louis, MO, 2005.

[32] G. Gu, D. Dagon, X. Qin, M. I. Sharif, W. Lee, and G. F. Riley, Worm detection, early warning, and response based on local victim information, in In Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004), Tucson, Arizona, 2004.

[33] Y. Kim, W. Lau, M. C. Chuah, and H. J. Chao, Packetscore: statistical based overload control against distributed Denial-of-Service, in 23rd Annual IEEE Conference on Computer Communications (INFOCOM), Hong Kong, 2004.

[34] K. Shanmugasundaram, H. Bronnimann, and N. Memon, Payload attribution via hierarchical Bloom Filters, in 11th ACM Conference on Computer and Communications Security, Washington, DC, 2004.

[35] James K. Mullin. Optimal semijoins for distributed database systems. IEEE Transactions on Software Engineering, May 1990.

[36] James K. Mullin. Estimating the size of a relational join. Information Systems, 1993.

[37] L. L. Gremilion. Designing a Bloom Filter for Differential File Access. Communications of the ACM 25, 1982, pp. 600-604.