Using Bloom Filter Array (BFA) to Speed up the Lookup in Distributed Storage System

Myat Pwint Phyu
University of Computer Studies, Yangon

Ni Lar Thein
University of Computer Studies, Yangon

ABSTRACT

Today's storage systems have a major issue for the long-term storage of massive amounts of unstructured data. The reliability and availability of that fortune of data become important factors. So, distributed storage system is essential for many large-scale organizations. It is challenging that how to access the distributed data from a place. In this paper, a structure of the Bloom filter array (BFA) is proposed to get time and space efficiency in distributed storage system. The proposed structure that can efficiently lookup the queries will be discussed from the algorithm perspective and then evaluate BFA through simulations.

Keywords

Unstructured data, large-scale distributed storage, replication, availability, reliability, bloom filter

1. INTRODUCTION

Nowadays, organizations have struggled under heavy burden of its data. On the other hand, as storage system grows larger and more complex, many file systems have emerged focusing on specialized requirements such as data sharing, remote file access, distributed file access, parallel files access, high performance computing, archiving etc.

The boost of massive amount of data in storage system results in large bandwidth and high latencies demands to provide fault tolerant continuous access in distributed storage system. Many research communities have attempted to solve these availability and reliability issues by various ways. Nevertheless, how to lookup the massive amount of data is a common consideration in many different techniques.

For looking up data in P2P, the method based on the distributed hash table (DHT) is actively researched [9, 11, 12, 14]. In the method with DHT, the load of the network is smaller than that of pure P2P to which the query is flood. Moreover, the load of the server can be widely distributed in the method compared with hybrid P2P. But there is a problem of costing for maintenance.

On the other hand, the method of looking up data by using the Bloom filter is researched so far [1, 5, 8, 13]. The resource on the distributed systems can be efficiently looked up by forwarding the query using the Bloom filter.

In this paper, we will present a method to look up data in distributed storage system without the need of query forwarding. Array structure of the Bloom filter is used to decide whether a specific attribute value belongs to a given set.

The rest of this paper is organized as follows. Section 2 shows the related works of the system. Section 3 explains some theory backgrounds and Section 4 introduces the proposed system with the algorithm analysis. The performance evaluation through simulation is presented in Section 5.Then we conclude the paper with future work in Section 6.

2. RELATED WORK

There are many researches and proposed methods for solving the query of a specific content to the distributed systems. One of the methods to change the direction where the query is forwarded with the Bloom filter of the networked site is the attenuated Bloom filter [13]. In the attenuated Bloom filter, each node manages a table of the Bloom filters. The node can know the hops from the node to the target node in the network by exchanging the table of Bloom filters with the adjacent node. And it can forward the query to the adequate adjacent node. However, this method cannot fix the upper bound of the number of query forwarding. Chord algorithm [14] which is the one of the DHT is used for the method of the query forwarding in the Bloom filter [15]. And another research proposed the management method of the Bloom filter with B-tree structure for information retrieval [16]. It showed that the number of Bloom filters can be less than that of the Bloom filter with fixed ring structure and there is the speed improvement of the query processing of each node. But in the previous methods, there is the network limitation such as bandwidth and latency because the query must hop between nodes. In this paper, the proposed BFA method does not need to consider the barrier of the network.

Bloom filter, as a space efficient data structure, can support query (membership) operations with $O(1)$ time complexity since a query operation needs to probe constant-scale bits in Bloom filters. Standard Bloom filters [2] have inspired many extensions and variants, such as the compressed Bloom filters [10], the space-code Bloom filters [7], the spectral Bloom filters [17], distributed Bloom filter [19] and the beyond Bloom filters [3]. The counting Bloom filters [5] are used to support the deletion operation and represent a set that changes over time. Multi-Dimension Dynamic Bloom Filters (MDDBF) [1] supports representation and membership queries based on the multi-attribute dimension. A novel Parallel Bloom Filters (PBF) and an additional hash table [18] can maintain multiple attributes of items and verify the dependency of multiple attributes, thereby significantly decreasing false positive rates. Whenever space is a concern, a Bloom filter can be an excellent alternative to storing a complete explicit list.

3. BLOOM FILTER

Bloom Filters (BFs) provide space-efficient storage of sets at the cost of a probability of false positives on membership queries. A Bloom filter is traditionally implemented by a single array of M bits, where M is the filter size. On filter creation all bits are reset to zeroes. A filter is also parameterized by a constant k that defines the number of hash functions used to activate and test bits on the filter. Each hash

function should output one index in M. When inserting an element e on the filter, the bits in the k indexes $h_1(e), h_2(e), ..., h_k(e)$ are set.

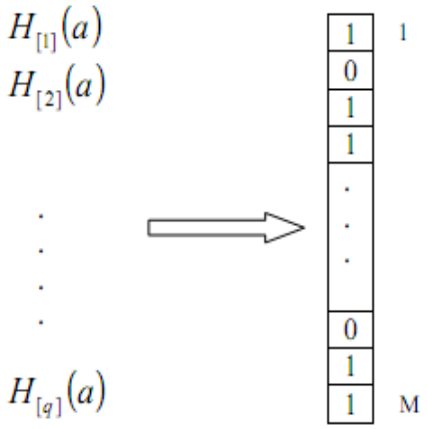

Fig 1: Standard Bloom filter structure

Bloom Filter is a bit array of M bits for representing a set $S = \{a_1, a_2, ..., a_n\}$ of n items. All bits in the array are initially set to 0. Then, a Bloom filter uses q independent hash functions $\{h_1, ..., h_q\}$ to map the set to the bit address space $[1, ..., M]$. For each item a, the bits of $h_i(a)$ are set to 1. To check whether an item a is a member of S, we need to check whether all $h_i(a)$ are set to 1. If not, a is not in the set S. If so, a is regarded as a member of S with a false positive probability, which suggests that set S contains an item a although it in fact does not. Generally, the false positive is acceptable if it is sufficiently small. The time complexity of a standard bloom filter is a fixed constant $O(1)$, completely independent of the number of items in the set. Use of Bloom filters have a strong space advantage over other data structures for representing sets, such as self-balancing binary search trees, tries, hash tables or simple arrays or linked listed of the entries. Most of these require storing at least the data item themselves, which can require anywhere from a small number of bits to arbitrary number of bits.

4. THE PROPOSED SYSTEM

As an ongoing research, a system to replicate data in large-scale distributed storage system is being developed for space efficiency and reliability guarantee. A client will operate by sending the fingerprints of a file to a randomly selected storage node as an agent server. In the server, the fingerprints are checked to see if it is in the cache. If so, there will be existing segments. If it is not, a Bloom filter is checked to determine whether the fingerprint is likely to exist in the storage volume. If it is a hit, the corresponding list of fingerprints is loaded into the cache. And then return response of which segments have already existed and their associated reliability levels and then the system is going to do the rest of the flow. The detail system flow is omitted in this paper and the performance of the Bloom filter lookup is only emphasized.

A simple data structure called bloom filter is used to get a fast lookup on each node. When a request comes, Bloom filter array (BFA) starts to return the hit/ miss response. Each node has a bloom filter maintaining fingerprints of locally stored chunks and the bloom filters of other nodes as an array. Figure 2 shows the structural representation of Bloom filter array.

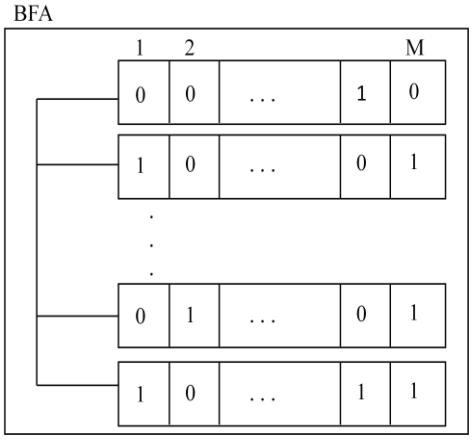

Fig 2: Structural representation of BFA

4.1 BFA Query

```
Lookup (x)
1: Fingerprint(x);
2: for B: 1 to n do
3:     c ← 0;
4:     j ← 1;
5:     while j ≤ k do
6:         i ← A[j];
7:         if B_i == 1 then
8:             c++;
9:         end if
10:        j ← j + 1;
11:    end while
12:    if (c==j) then
13:        return true;
14:    end if
15: end for

Fingerprint(x)
1: for j: 1 to k do
2:     A[j] ← h_j(x);
3: end
```

Fig 3: Lookup algorithm in BFA

As shown in Figure 3, a query to a Bloom filter array is encoded as a fingerprint with k independent hash functions. In the BFA, there are n Bloom filters and the bit positions of the queried fingerprint are compared with those of each Bloom filter. If all matches are non-zeros, it can be said that the query is presented in the Bloom filter array.

4.2. Algorithm Analysis

The efficiency of a Bloom filter depends on some key parameters. The more the number of hash functions (k) and the size of filter (M) are used, the more the computation and the space and the lower the false positive rate will be got. Moreover, if the Bloom filter array is arranged by the access frequency, the operations on the BFA can be done in $O(n)$ as a worse case. Although there are many variant kinds of

bloom filter [10, 7, 16, 17, 3, 6] to get better performance, the most basic one is used for simplicity.

5. PERFORMANCE EVALUATION

In this section, the BFA is simulated by using the large amount of random data and the performance is measured in terms of query accuracy and the memory overhead.

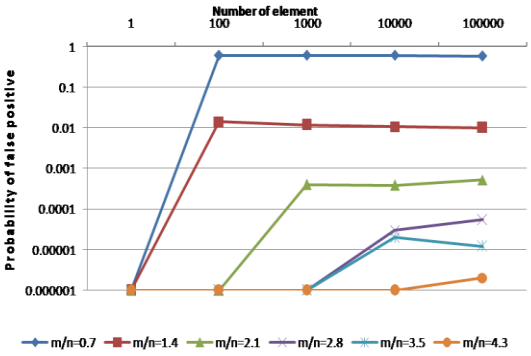

Fig 4: False positive probability of BFA using 10 hash functions

The false positive performance depends on the bit per elements ratio (m/n) and the optimal number of hash functions is determined by $k = (m/n)\ln 2$. Figure 4 shows that the accuracy of BFA is proportionally high depending on the bit per element ratio and the space used by PBA is linear to the number of items to get a specified false positive probability.

6. Conclusion and Future Work

In this paper, the efficient way to retrieve required data from distributed storage system is presented. The Bloom filter array approach is introduced to improve lookup efficiency. The proposed system is evaluated with various workloads and showed that the system can save time and space utilization in the distributed storage system using BFA. In this paper, the simulation using Bloom filter array to retrieve the specific facts in distributed storage system is only demonstrated at this time. The system is still in progress and the complete system integrating with this BFA approach will be presented at future.

7. REFERENCES

[1] A. Broder, M. Mitzenmacher, Network Applications of Bloom Filters: A Survey, Internet Mathematics, 2002, pp.636-646.

[2] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, vol. 13, 1970.

[3] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, Beyond Bloom filters: From approximate membership checks to approximate state machines, SIGCOMM, 2006.

[4] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An Improved Construction for Counting Bloom Filters," in 14th Annual European Symposium on Algorithms, LNCS 4168, 2006, pp. 684–695.

[5] 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 (MobiCom '99), pp. 24—35. New York: ACM Press, 1999.

[6] D. Guo, J. Wu, H. Chen, and X. Luo, Theory and network application of dynamic Bloom filters, INFOCOM, 2006.

[7] A. Kumar, J. Xu, and E. W. Zegura, Efficient and scalable query routing for unstructured peer-to-peer networks, INFOCOM, 2005.

[8] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao, OceanStore: An Architecture for Global-Scale Persistent Storage, ACM SIGPLAN Notices 35:11 (2000), 190—201.

[9] P. Maymounkov and D. M. Kademlia, A Peer-to-peer Information Systems Based on the XOR Metric, In Proceedings of the IPTPS 2002, Boston, March 2002.

[10] M. Mitzenmacher, Compressed Bloom filters, IEEE/ACM Trans. on Networking, vol. 10, no. 5, pp. 604–612, 2002.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, A Scalable Content Addressable Network, In Proceedings of the ACM SIGCOMM 2001 Technical Conference, San Diego, CA, USA, August 2001.

[12] A. Rowstron and P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pages 329-350, November 2001.

[13] 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. Los Alamitos, CA: IEEE Computer Society, 2002.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, In Proceedings of the ACM SIGCOMM 2001, San Diego, CA, USA, August 2001.

[15] K. Sato, N. Matsumoto and N. Yoshida, Multi Keyword Search for DHT P2P Networks, IPSJ/IEICE Forum on Information Technology (FIT)2006, (2006).

[16] F.Sato, Evaluation of the Structured Bloom Filter, In Proceedings of CISIS '10 of the 2010 International Conference on Complex Intelligent and Software Intensive Systems, pp. 313-320.

[17] C. Saar and M. Yossi, Spectral Bloom filters, SIGMOD, 2003.

[18] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing, Proc. ACM SIGCOMM, 2005.

[19] Y. Zhang, D. Li, L. Chen, and X. Lu, Collaborative Search in Large-scale Unstructured Peer-to-Peer Networks, ICPP, 2007.