MINOR PROJECT-1

SYNOPSIS

ON

MALICIOUS URL CHECKER USING BLOOM FILTER

Submitted By

SPARSH ANURAG	**500061280**
SURAJ	**500060081**

Under the guidance of

DR. LALIT KANE

ASSOCIATE PROFESSOR

Department of **__________Virtualization_____________**,

Cloud Computing & Virtualization Technology

Department of Virtualization

School of Computer Science

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

Dehradun-248007

Aug-Nov, 2019

ABSTRACT

In today’s world, with the advent of the Internet and mobile devices with limited resources and with the growing requirements of information storage and data transfer, cloud computing has become an important aspect but cloud computing also requires physical infrastructures, somewhere down the lane. This exponential sub purge of data leads to high demand for fast data processing that leads to a high computational requirement which is usually not available at the user’s end. Bloom filter is a data structure designed to tell rapidly and memory-efficiently whether an element is present in a set.[1] Bloom Filters provide space-efficient storage of sets at the cost of a probability of false positives on membership queries. The size of filter must be defined based on the no. of elements to store and desired false positive probability, being impossible to store extra elements without increasing the false positive probability. This typically leads to a conservative assumption regarding maximum set size, possible by orders of magnitude, and a consequent space waste. In this project, we will be using hashing algorithms like Murmur and FNV to implement a bloom filter.

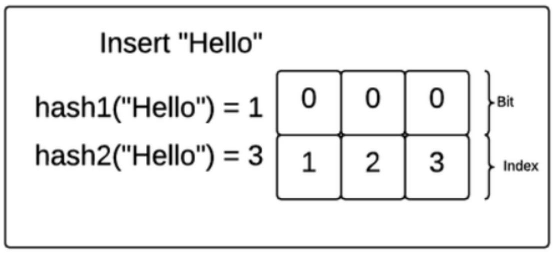
Keywords: Data Structures, Bloom Filters, Hashing Algorithms, Murmur, Fowler-Noll-Vo hash

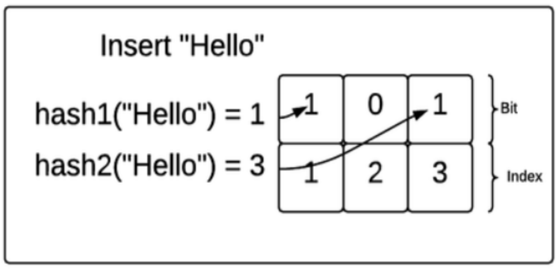
INTRODUCTION

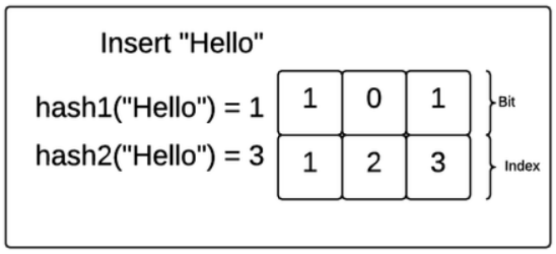
A bloom filter is a probabilistic data structure that is based on hashing. It is extremely space efficient and is typically used to add elements to a set and test if an element is in a set. Though, the elements themselves are not added to a set. Instead a hash of the elements is added to the set. When testing if an element is in the bloom filter, false positives are possible are possible. It will either say that an element is definitely not in the set or that it is possible the element is in the set. A bloom filter is very much like a hash table that it will use a hash function to map a key to a bucket. However, it will not store that key in that bucket, it'll simply mark it as filled. So, many keys might map to the same filled bucket, creating false positives.

Bloom filters [2] provide space-efficient storage of sets at the cost of a probability of false positive on membership queries. Insertion and membership testing in Bloom filters implies an amount of randomization, since elements are transformed using one-way hash functions. Testing for the presence of elements that have actually been inserted in the filter will always give a positive result there are no false negatives. On the contrary, there is always some probability of false positives: elements that have not been inserted into the filter can erroneously pass the membership test.

An empty bloom filter is a bit array of m bits, all of which are initially set to zero. A bit array is an extremely space-efficient data structure that has each position set to either 0 or 1. A bloom filter also includes a set of k hash function with which we hash incoming values. These hash functions must all have a range of 0 to m-1. If these hash functions match an incoming value with an index in the bit array, the bloom filter will make sure the bit at that position in the array is 1.







In this example, "Hello" was hashed to 1 by the first hash function and 3 by the second hash function. So, the bloom filter made sure the bits at index 1 and 3 were flipped to 1.

When a query happens in a bloom filter, we once again hash the key with all of our hash functions. We then check all of the output bits to make sure they are all 1. If any of them is a 0, we know for sure that the key we are searching for is not in our list. If they are all 1, we know that it might be.

Murmur3 Hash Algorithm

Murmur hashing is a non-cryptographic hash function which is used for Hash based look-ups, it uses 3 basic operations as a whole.

1. Multiplication
2. Rotate
3. XOR

It uses multiple constants which are just there to make it good hash function by passing 2 basic tests.

1. Avalanche Test
2. Chi-Squared Test

It relies on:

A seed, to start with. It is often customizable but it has to be carefully set because a different set because a different seed will lead to a different hash for the same key, some fixed constants, determined empirically.

FNV Hash Algorithm

FNV or Flower-Noll-Vo is a non-cryptographic hash function which are designed to be fast while maintaining a low collision rate. The FNV speed allows one to quickly hash lots of data while maintaining a reasonable low collision rate. The FNV speed allows to quickly hash lots of data while maintain a reasonable collision rate. The high dispersion of FNV hashes makes them well suited for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc.

In this algorithm we calculate the hash value for the data using multiplication and XOR operations. Firstly, we take FNV offset basis in hash value and then we multiply the value obtained in the first value with FNV prime and further take the value obtained and do XOR with data to be hashed to get final hash value.

PROBLEM STATEMENT

Millions of people search the Internet, government databases, private databases and blockchains everyday for medical advice, financial updates, weather reports, maps and more. Likewise, millions of people want or need fast searches.

A Solution to this problem i.e. to speed up many searches is the use of indexes, Indexes, like those at back of textbooks, provides the locations of all the search terms but the drawback is they require a large amount of storage. An effective method to speed up many searches, with less storage requirements is the use of “Bloom filters”

LITERATURE REVIEW

In recent years, Bloom filters have received increased attention, and they received increased attention, and they are now being used in a large number of systems, including peer-to-peer systems, web caches, database systems and others.

- In [3] the authors introduce the idea of a counting Bloom filter, allowing elements to be removed from the set represented by the Bloom filter, Spectral Bloom Filters [4] use a similar approach to store multi-sets [5] proposes a multi-segment Bloom Filter that allows efficient access when this data structure is stored on disk; a similar approach[6] is used in a network routing algorithm. Completely Bloom Filter is passed as a message, by using larger but sparser filters that led to smaller compressed sizes.
- In [7] the authors introduced the idea of using Bloom Filter as a Spam Detector. A spam detector prevents sending spam emails to the inbox by keeping a dictionary of acceptable emails ids. When an email arrives, the spam detector checks if the email id belong to the accepted mail id dictionary, and forwards the email to the inbox if the search result is positive.
- Akamai's Content Distribution Network. "Akamai's web servers use Bloom filters to prevent one-hit-wonders from being stored in its caches. One-hit-wonders are web objects requested by users just once, something that Akamai found applied to nearly three-quarter of their caching infrastructure. Using a Bloom filter to detect the second request for a web object and caching that object only on its second request prevents one-hit wonders from entering the disk cache, significantly reducing the workload and increasing disk cache hit rates."
- There have been several prior researches on the personalized search engine. A comprehensive survey on personalized search can be found in [8]. One approach of personalized search is to re-rank search results by checking content similarity between web pages and user interest model[9].

OBJECTIVE

A bloom filter is a space-efficient data structures that lets us quickly check whether or not an item is in set. The tradeoff for that space efficiency is that it it’s probabilistic. A bloom filter doesn’t store the elements themselves, this is the crucial point. We don’t use bloom filter to test if an element is present, we use it to present whether it’s certainly not present, since it guarantees no false negatives. It is highly space-efficient and fast.

METHODOLOGY

Algorithms

Bloom Filter

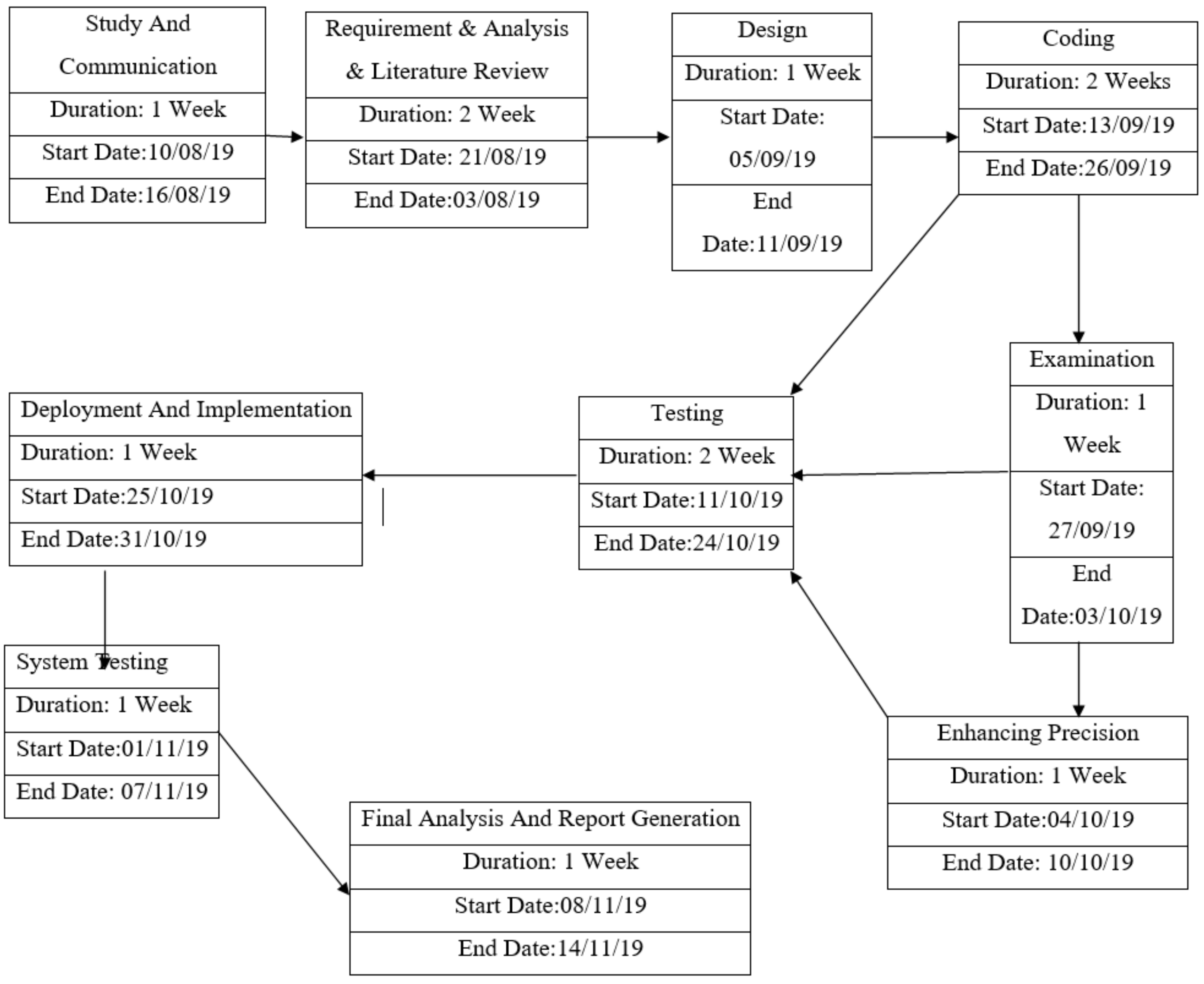
1. Bloom filter is just a bitmap. Initially all the bits are set to 0.
2. For Insertion:
    - Hash the string.
    - Mod the result by the length of our bitmap.
    - Set that bit to 1.
3. For Lookups:
    - Hash it.
    - Mod the result.
    - Check if the corresponding bit is 0 or 1.

SYSTEM REQUIREMENTS

- **Hardware Interface:**
  - 64 bits processor architecture supported by windows.
  - Minimum RAM requirement for proper functioning is 4 GB.
  - Required input as well as output devices.

- **Software Interface:**
  - C Compiler (GCC).
  - Socket Programming Library in C.

SCHEDULE (PERT CHART)



REFERENCES

[1] A Scalable Bloom Filter for Membership Queries, Kun Zie, Yinghua Min

[2] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communication.

[3] L. Fan, P. Cao, J. Almeida, A. Z. Broder, Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM Trans. Network

[4] S. Cohen, Y. Matias, Spectral bloom filters, in: Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD ’03), ACM Press, New York, NY, USA,

[5] U. Manber, S.Wu, An algorithm for approximate membership checking with application to password security, Inf. Process. Lett. 50 (4) (1994)

[6] S. Dharmapurikar, P. Krishnamurthy, D. E. Taylor, Longest prefix matching using bloom filters, in: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM ’03), ACM Press, New York, NY, USA, 2003,

[7] Barna Saha, Algorithms for data science.

[8] F. Liu, C. Yu, S. Member, W. Meng, “Personalized Web Search for Improving Retrieval Effectivness,” IEEE Transactions on Knowledge and Date Engineering, IEEE Press,2004,p. 28

[9] Z. Dou, R. Song, J.Wen, “Evaluating the Effectiveness of Personalized Web Search, ” IEEE Transactions on Knowledge and Date Engineering, IEEE Press, 2009, pp. 1178-1190.