Bloom Filter based on unassociated Hash Methods

Thesis Submitted in Partial Fulfillment of Requirements for the Award of the Degree

Of

Master of Technology in

INFORMATION SYSTEM

SUBMITTED

BY

SWATI SINGH

(2K15/ISY/20)

UNDER THE GUIDANCE OF

Dr. RAJNI JINDAL

(Head of Department)

Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

BAWANA ROAD, DELHI-110042

(2015-2017)

Certificate

This is to certify that Ms. **SWATI SINGH (2K15/ISY/20)** has carried out the major project titled **"Bloom Filter based on unassociated Hash Methods "** as a partial requirement for the award of **Master of Technology** degree in **Information System** by **Delhi Technological University, Delhi.**

The Major project is a bonafide piece of work carried out and completed under my supervision and guidance during the academic session 2015-2017. The Matter contained in this thesis has not been submitted elsewhere for the award of any other degree

(Project Guide)
Dr. Rajni Jindal
Head of the Department
Computer Science & Engineering
Delhi Technological University

Declaration

I hereby declare that this M.Tech thesis entitled **"Bloom Filter based on unassociated Hash Method"**, was carried out by me for the degree of Masters of Technology under the guidance and supervision of **Dr. Rajni Jindal, Head of Department, Computer Science and Engineering, Delhi Technological University , Delhi**, India. The interpretations put forth are based on my reading and understanding of the original texts and they are not published anywhere in the form of books, monographs or articles. The other books, articles and websites, which I have made use of are acknowledged at the respective place in the text. For the present thesis, which I am submitting to the University, no degree or diploma or distinction has been conferred on me before, either in this or in any other University.

SWATI SINGH

Roll No.: 2K15/ISY/20

M.Tech (Information System)

Department of Information Technology

Delhi Technological University, Delhi

Acknowledgement

I express my gratitude to my major project guide **Dr. Rajni Jindal , Head of Department, Computer Science and Engineering** at **Delhi Technological University, Delhi** for the valuable support and guidance she provided in making this major project. It is my pleasure to record my sincere thanks to my respected guide for her constructive criticism, interminable encouragement and valuable insight without which the project would not have shaped as it has.

I humbly extend my word of gratitude to Dr. Kapil Sharma, Head of Department (Information Technology) and all the other faculty members and staff of department for providing their valuable help, time and facilities at the need of hour.

SWATI SINGH

Roll No.: 2K15/ISY/20

M.Tech (Information System)

Department of Information Technology

Delhi Technological University, Delhi

Abstract

Blossom filters are basic randomized information structures that are greatly valuable practically speaking. Truth be told, they are useful to the point that any noteworthy lessening in the time required to play out a Bloom channel operation instantly means a considerable speedup for some functional applications. Sadly, Bloom filters are simple to the point that they don't leave much space for advancement. At the point when space is an issue, a Bloom channel might be an astounding contrasting option to keeping an express rundown. The disadvantage of utilizing a Bloom channel is that it permits false positives. Their impact must be painstakingly considered for every particular application to decide if the effect of false positives is adequate.

In this research work it is shown that the false positive performance of a standard Bloom filter implementation strongly relies on the selection of hash functions, even if these hash functions are considered good. The three hashing methods implemented for the ideal bloom filter are compared on the grounds of their relative time consumption, false positive and number of collisions.

Table of Content

Certificate i

Declaration ii

Acknowledgement iii

Abstract iv

Table of Contents v

List of Figures vii

Chapter 1 INTRODUCTION 1

1.1 Bloom Filter 1

1.2 Working 2

1.3 Application 2

1.4 Space and Time Advantages 4

1.5 Advantage over normal query optimization 5

Chapter 2 LITERATURE SURVEY 6

2.1 Approximate Membership Query 6

2.2 AMQ Problem 7

Chapter 3 PROBLE STATEMENT 9

3.1 Problems in Bloom Fliter 9

3.1.1 Bloom Filter Size 10

3.1.2 Developing and Membership Existence in Bloom Filter 10

3.1.3 Can't give the embedded items 10

3.1.4 Evacuating a component 10

3.1.5 Executions in Different Languages 11

Chapter 4 ERRORS 12

4.1 False Positive 12

4.1.1 Probability of False Positive 13
4.2 False Negative 14
4.2.1 Probability of False Negative 15
4.3 Hash Collision 16
Chapter 5 HASH FUNCTIONS 18
5.1 Hash Function 18
5.2 Practical Hash Functions used for Bloom filter 18
5.2.1 Cryptographic Hash Functions 18
5.2.3 Universal Hash Functions 19
5.2.2 Non – Cryptographic Hash Functions 19
5.3 The ideal number of hash functions 19
Chapter 6 IMPLEMENTATION 21
6.1 Locality Sensitivity Hashing 21
6.1.1 Definition 21
6.2 MD5 Algorithm 23
6.3 SHA1 Algorithm 24
Chapter 7 RESULT & ANALYSIS 26
Chapter 8 CONCLUSIONAND FUTURE SCOPE 31
8.1 Conclusion 31
8.2 Future Scope 31
REFERNCES 32

List Of Figures

Fig.1 A typical hash transform 1

Fig.2 A simple bloom filter 2

Fig.3 Memory access with bloom filter 5

Fig.4 Collision of two keys in hashing $h(k_i)=h(k_j)$ 16

Fig.5 Geometry description of correct and false answers 22

Fig.6 MD5 Procedure 24

Fig.7 SHA1 Procedure 25

Fig.8 Difference in time consumed with p=0.05 and BF size = 1470802 27

Fig.9 Difference in time consumed with p=0.95 and BF size = 25183 27

Fig.10 Difference in no. of collisions with p=0.05 and BF size = 1470802 28

Fig.11 Difference in no. of collisions with p=0.95 and BF size = 210704 28

Fig. 12 Comparison of three hash function with similar fp = 0.05 29

Fig. 13 Comparison of three hash function with similar fp = 0.95 29

Fig.14 Comparison in no of collisions with p=0.05 30

Fig.15 Comparison in no of collisions with p=0.05 30

Chapter 1

INTRODUCTION

1.1 Bloom Filter

Probabilistic information structure that depends on hashing is called a Bloom Filter. It is to a great degree space proficient and is normally used to add components to a set and test if a component is in a set. However, the components themselves are not added to a set. Rather a hash of the components is added to the set. It is strategy for speaking to a set $\alpha = \{ \alpha_1 \alpha_2 \alpha_3 \dots \alpha_i \}$ of i components (additionally called keys)to brace membership queries, comprised of j bit array and z independent hash functions.

Rather than putting away the key-esteem sets, as a normal hash table would, a Bloom Filter will give you just a single snippet of data true or false in light of the nearness of a key in the hash table. This unwinding enables the filter to be spoken to with a substantially compact bit of memory, rather than putting away each value , a bloom filter is just a variety of bits showing the nearness of that key in the filter.

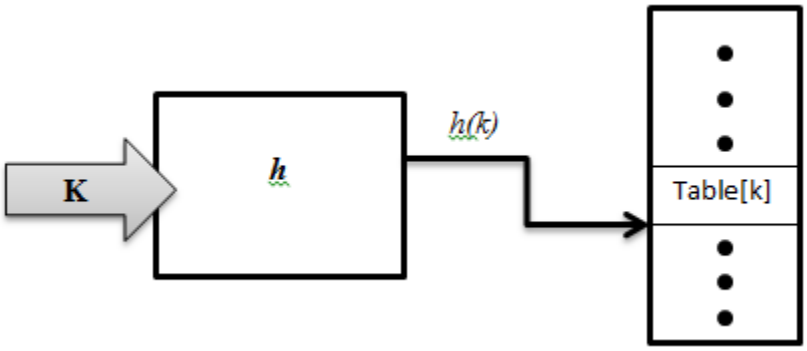


Fig.1 A typical hash transform

1.2 Working of bloom Filter

- Initially, a bit array is awarded and instated to every one of the zeroes. Every component of the given set is then processed through various self – reliant and random hash functions.
- Then each indexed bit is set to one which are indexed into bit array from the hash results.
- After the inclusion, membership queries can be led by processing the component being referred to and checking whether the demonstrated bits are set.
- On the off chance that no less than one piece is zero, at that point the component does not have a place with the given set without a doubt. Otherwise, the component is thought to be a convincing individual from the set.

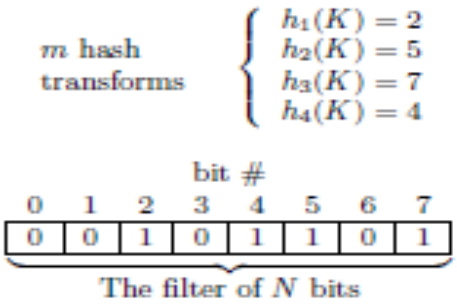


Fig.2 A simple bloom filter

There is a probability, however, that this suspicion isn't right and that the component is, truth be told, a false positive. Such an occasion happens when all showed bits of a non-part component are already set by other authentic components. Further addition of more components are that are embedded to filter, as a result increments the probability of false-positive and to outcome of this filter turns out to be more soaked (i.e., the portion of bits which are stationed as 1 increments until the point that there is no other way to recognize the components which have a place with the filter). In spite of this disadvantage, when the false positive approximation or probability is lay low ,Bloom filter are very effective.

1.3 Applications of bloom filter

i. The most widely recognized use for bloom filter is presumably trying to check whether a component exists on disk before executing out any i/o. One take a chance searching for something which is absent from there but rather one will never skip testing a section in light of the fact that the bloom filter said it wasn't there and it was. This ought to minimize I/O for queries significantly finished expansive informational indexes.

ii. Another decent utilize is cheap unique count. On the off chance that you have to know the surmised number of exceptional things you've seen (say, in a stream), you can utilize a bloom filter to test in the event that you've seen that component some time recently. Increase the check by one if the bloom filter says it's not in the set. One will get false positives (under counting), yet it's less expensive than keeping the whole set in memory. It's decent to have the capacity to state "we've seen at any rate this numerous exceptional things".

iii. Facebook utilizes bloom filter for typeahead look, to get companions and companions of companions to a client wrote inquiry. For each companion association bloom filter is just 16 bits (an edge in the facebook social chart) and known as "world's smallest bloom".

iv. Transactional Memory (TM) has as of late connected Bloom Filters to distinguish memory access clashes among strings. TM is a creating simultaneous programming style which plans to mitigate the difficulties of programming with locks (serialization or deadlock/races). The software engineer's assignment is to compose ("basic") transactional code, and the hidden TM framework tracks memory access and averts information races/deadlock.

v. Monitor the pages that a given client has gone by without really having the capacity to list the connections they have gone by. This encourages us address the protection worries that individuals may have about having our module in their framework. The bloom filter just empowers us to test if a url has been gone to yet not specify the urls they have gone to.

vi. Apache HBase utilizes blossom channel to support in incrementing the speed of reading by relocating with pointless circle careful examination of HFile squares that never contain a particular row or column.

Since HBase inside stores a rationale push by particular key-esteem sets for every segment , client can fabricate blossom channels by either row or row + column relying upon the real question designs.

1.4 Space and Time Advantages

Over other data structures bloom filters have a strong space advantage for representing sets. The majority of these require putting away at any rate the information things themselves, which can require anyplace from few bits, for little whole numbers, to a self-assertive number of bits, for example, for strings. In any case, Bloom filters never store the information things by any means, and a different arrangement must be accommodated the real stockpiling. Connected structures when there is a speak about pointers acquire an extra direct storage problem. The Bloom filter with one percent mistake and with an ideal estimation of k, interestingly, needs just around 9.6 bits for every component, paying little mind to the extent of the components. This preferred standpoint comes somewhat from its minimization, acquired from exhibits, and incompletely subject to its nature of being probabilistic. The one percent rate of false-positive can be diminished with a factor of 10 by including just around 4.8 bits for each component.

A Bloom filter with j bits and w hashing capacities, O(w) will be both addition and participation testing. That is, each time a component is inserted to the set or check set participation, there is a simple requirement to run the component through the w hash capacities and insert it to the set or check those bits.

The space favorable circumstances are more hard to total up; it relies upon the error rate to endure. It likewise relies upon the potential scope of the components to be embedded; on the off chance that it is extremely constrained, a deterministic piece vector can improve. On the off chance that you can't even ballpark appraise the quantity of components to be embedded, there might be in an ideal situation with a hash table or an adaptable Bloom filter.

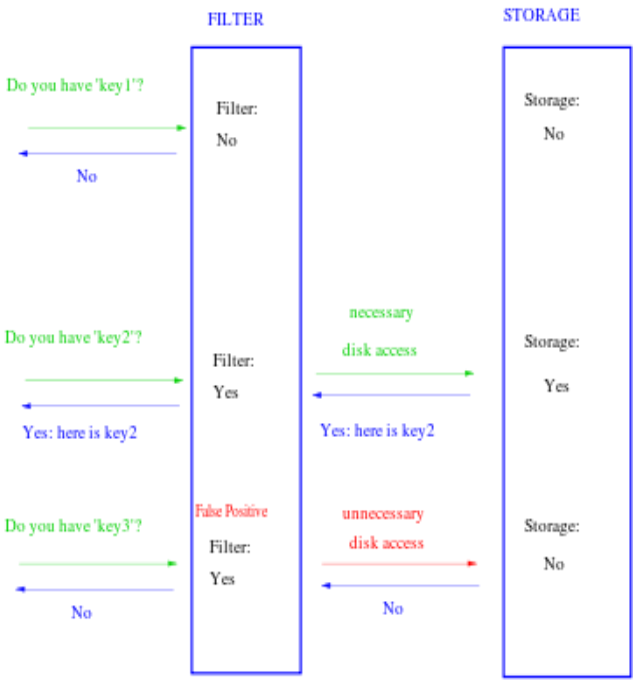


Fig.3 Memory access with bloom filter

1.6 Advantages over technique of normal query optimizing

- Efficiency in Space - the compact bloom filter is direct with the extent to the set and never rely upon from where S receives its esteems.
- Swift development - development is a quick and not much tedious procedure, because it needs a solitary sweep about the information.
- Effective testing of membership - To inspect the participation of a component in S needs as it were processing the k hash capacities (where k is normally consistent) and utilizing to k bits.
- Functioning - Its exactness relies upon its size. Bloom filter size chooses probability of FP so it ought to be ideal decision for putting away information. On the off chance that it is not as much as required then FP probability may be increment or in the event that it is more than ideal at that point seeking won't get influenced.

Chapter 2

Literature Survey

In the foregoing work locality sensitivity bloom filter has been proposed to productively bolster quick AMQ without trading off query execution. LSBF is an area-productive model utilizing bit-wise vector. Its working involve hashing an unit to a bucket which is a paired piece, out of which a bit vector can show the presence of nearly accurate unit, though this working is similar to an normal locality sensitive hashing. The outline depends on the perception that Bloom filter can delineate things into a generally brief storage room with the guide of variation hash functions. In this manner, it is doable to supplant autonomous and uniform hash works in Bloom filter with locality sensitive hash functions while keeping up thing vicinity (because of LSH charateristics) and accumulation room productivity in bit vector (as a result of Bloom filter characteristics). Through hypothetical investigation and broad analyses on appropriated framework usage, it demonstrates the effectiveness of LSBF to deal with AMQ as far as brisk inquiry reaction, high query accuracy, low I/O cost, and space overhead.

A thorough hypothetical examination FP, FN approximation of LSBF, and storage overhead was given. To check LSBF in genuine implementation, the proposed LSBF was executed and examined for its accomplishment in real applications, follows speaking to operations on document frameworks and high-dimensional natural data.

2.1 Approximate Membership Query

The goal of AMQ is to decide if a query q which is given , is estimated to an informational index S. In particular, suppose S be a collection of data points in a lattiude which actually is d-dimensional space U represented in the form (U,d) , and S⊂U. Given a consistent attribute R, if ∃ p Є S then the q which is a query point generally recognized as imprecise partner and the pair distance will be accurate for ‖p,e‖ ≤ R. Moreover, set S’ which is superset of set S vitally consist of the points approximate to set S. The precise constituent of S’ is from the approximate constituent of set S [4].

Rather than exact-matching query of membership, i.e., "e Є S?", it is more appealing and intriguing to help Approximate Membership Query (AMQ), i.e., "e →S ? . By changing "e Є S ? " to "e → S ?," there is no requirement of testing the nearness of direct e yet its closeness toward any part among set S in the stated measure fields.

2.2 AMQ Problem

‖*‖ is used to represent the distance computed amongst the 2 elements in a d- dimensional space, the representation of AMQ problem is as :

Problem 1 (Approximate Membership Query).

If ∃ p Є S, ‖ p,e‖ ≤R for a given attribute R then a queried point e will be examine as the near to accurate constitute of some data information set S.

Problem 2 (c-Approximate Membership Query).

If ∃ p Є S, ‖ p,e ‖ ≤ cR and c ≤ 1 for any given attributes point of query e ,R and c then content set S acquire e as the c-approximate member.

When c is set to 1 the AMQ issue is unique instance of c-Approximate Membership Query issue.

AMQ False Positive : on the off chance that the query gets positive answer while truth be told ∀ p Є S, ‖ p,q‖ ≤R for any attribute R, to a data set S the queried item q will be a false positive.

AMQ False Negative : on the off chance that the query gets negative answer while truth be told ∃ p Є S, ‖ p,q‖ ≤R for any attribute R, to a data set S the queried item q will be a false negative.

Likewise if cR will be restored in place of R the false negative and positive of c-approximate membership query is eloquanted.

Preprocessing:

i. Select *D* functions g_c , c = 1,…D, by setting $g_c = (h_1, j, h_2, c,…h_k, c)$, where $h_1, c,…h_k, c$ are selected randomly out of family of LSH H[20].

ii. Compute *D* tables of hashes, for every c = 1,…D, the c^{th} h table of hash constitute the information set points being hashed utilizing the function g_c .

Algorithm for query:

1. Repeat , every c = 1, 2,…D

i. Extract information points out of pail g_c (e) amongst c^{th} table of hash.

ii. Considering every single accessed point, calculate the stretch within e and itself, record the point whether it's an accurate reply (cR-near neighbor for Strategy1, and R-near neighbor for Strategy2).

iii. Simultaneously when number of recorded information points approaches limit D, leave the process.

Chapter 3

Problem Statement

In traditional hash coding, a hash territory is sorted out into cells, and an iterative pseudorandom computational process is utilized to create, from the given arrangement of messages, hash locations of exhaust cells into which the messages are then put away. Messages are tried by a comparable procedure of iteratively producing hash locations of cells. The substance of these cells are then contrasted and the test messages. A match shows the test message is an individual from the set; an unfilled cell demonstrates the inverse. The user is thought to be comfortable with this and comparable ordinary hash-coding techniques.

The peruser should take note of that the new strategies are most certainly not planned as other options to ordinary hash-coding strategies in any application range in which hash coding is right now utilized (e.g. image table administration [1]). Or maybe, they are planned to make it conceivable to misuse the advantages of hash-coding methods in specific regions in which customary blunder free strategies experience the ill effects of a requirement for hash regions too huge to be center inhabitant and thus found to be infeasible. With a specific end goal to increase considerable decreases in hash zone measure, without presenting over the top reject times, the mistake free execution related with ordinary strategies is yielded. In application zones where blunder free execution is a need, these new strategies are definitely not appropriate.

3.1 Problems in Bloom Filter

Other than the undeniable false positive potential, the sprout channel can just report yes or no. It can't propose choices for things that may be near being spelled effectively. A blossom channel has no memory of which bits were set by which things so a yes or no answer is as well as can be expected get with even a yes answer not being right in a few conditions. The following segment shows a Trie information structure that won't report false positives and can be utilized to discover choices for mistakenly spelled words.

3.1.1 Bloom Filter Size

The span of the Bloom Filters should be known from the earlier in light of the quantity of things that need to embed. This is not very good on the off chance that there is no knowledge of or can't inexact the quantity of things. A subjectively extensive size can be taken, yet that would be a loss as there is an attempt to optimize or to streamline in any case and the motivation behind are embracement to pick Bloom Filter. This could be settled to make a bloom filter dynamic to the rundown of things that need to fit, yet relying upon the application, this may not be constantly conceivable. There is a variation called Scalable Bloom Filter which progressively alters its size for various number of things. This could moderate some of its inadequacies.

3.1.2 Developing and Membership Existence in Bloom Filter

While utilizing the Bloom Filters, one will acknowledge false positive rates, as well as there will have a tad bit overhead as far as speed. Contrasting with a hash map, there is certainly an overhead as far as hashing the things and in addition building the bloom filter.

3.1.3 Can't give the embedded items

Blossom Filter can't deliver a rundown of things that are embedded, there is only a check if an item is present, yet never get the full thing list as a result of hash collisions and hash functions. This is expected to seemingly the most noteworthy preferred standpoint over other information structures; its space proficiency which accompanies this burden.

3.1.4 Evacuating a component

Expelling a component from the Bloom Filter is unrealistic, there is no way to reverse an addition operation as hash comes about for various things can be filed similarly situated. In the event of need to do fix embeds, it is possible by numbering the supplements for each file in the Bloom Filter or have to develop the Bloom Filter from the begin barring a solitary thing. The two techniques include an overhead and not clear. Contingent upon the application, one might need to attempt to recreate the bloom filter from the begin as opposed to expelling or erasing things from the Bloom Filter.

3.1.5 Executions in Different Languages

Underway, there will be a preference not to reveal particular bloom filter execution. There are two reasons; one of them picking and executing great hash functions is essentially critical to disperse the mistake rate for any number of information sources. Second of them, it should be fight tried and ought not be mistake inclined both regarding blunder rate and its size.

Chapter 4

Errors

Errors can happen when at least two changes guide to a similar component. The participation test for a key F functions by checking the components that would have been manipulated if the key had been embedded into the vector. On the off chance that all the suitable hail bits have been set by hashes then f will be accounted for as an individual from the set. In the event that the components have been refreshed by hashes on different keys | and not F | then the participation test will mistakenly report F as a part.

4.1 False Positive

A Bloom Filter has an inherent issue of false positives, which distinguishes a contribution as a part despite the fact that the information is not really an individual from the set. Rate of False positive is in real called the approximation of such fallacious queries analyzed by bloom filter. Though this rate can be controlled by elevating bloom filter's size and by number of indices of hash. Be that as it may, when a given set is expansive, expanding the span of a Bloom Filter is restricted by the required memory sum since for fast processing Bloom Filters are normally actualized utilizing an on-chip memory.

For an element x not belonging to α false positive probability *p* can be computed after insertion of n components stated that superbly autonomous and irregular hash functions are utilized given that determined bit is still zero :

$$p = \left(1 - \frac{1}{j}\right)^{wi} \approx e^{-wi/j}$$

where *j* is the bits amount present in array , *i* is embedded elements and *w* the no. of functions of hash.

After all n inclusions the fraction of bits in 0 since a similar calculation can be connected to each bit in the exhibit, by and large, is $p \approx e^{-wi/j}$. Therefore (1-*p*) will be fractions of bits in 1 after n

inclusions. For every *w* locations pointed out of a non-member the presence of a bit in 1 is the false positive probability f_p:

$$f_p = (1-p)^w \approx \left(1 - e^{-wi/j}\right)^w$$

Two fascinating facts that can be concluded from the above equation are

I. Decrement in false positive probability when large amount of bits per element are utilized as a result of decrement of the fraction $\left(1 - e^{-wi/j}\right)^w$ of bits in 1.

II. Increment in the value of *w* results in much larger bits set fraction , which results in higher false positive rate. Then again, the likelihood of finding each showed bit set reduces with higher estimations of *w*.

The false positive likelihood emphatically relies upon the quantity of bits set in the array which is drawback of the bloom filter. False positive probability increments as more bits are set for any number of hash functions. Right when all bits of the channel are set, the false-positive likelihood is unmistakably 100%. In this condition, each non-part attempted against the channel is recognized as a true blue individual from the set.

4.1.1 Probability of False Positive

False positives are the consequences of probable impacts of hash and absence of character consistency check as far as multidimensional qualities from distinct things. The principle explanation behind false positive is the mislaying factor of the multidimensional union credits to one personality and it can be just confirm that the point q which being queried earlier is surmised to the data set in each measurement.

The probability of a specific bit not being set to one with a specific function of hash amid the addition of a component in the event that j is the quantity of bits in the array, is

$$1 - \frac{1}{j}$$

The probability whether a bit not settled to one by a particular of the hash functions , in the event that w is the quantity of hash works is ,

$$\left(1-\frac{1}{j}\right)^{w}$$

On the off chance that we have embedded i components, the likelihood that a specific piece is still 0 is ,

$$\left(1-\frac{1}{j}\right)^{wi}$$

probability of being 1 is ,

$$1-\left(1-\frac{1}{j}\right)^{wi}$$

The probability that would make the algorithm incorrectly guarantee that the component is in the set of every one of them being 1, is frequently given as

$$\left(1-\left[1-\frac{1}{j}\right]^{wi}\right)^{w}$$

It is not entirely right as it accept autonomy for the probabilities of each piece being set. In any case, expecting it is a nearby estimation that the approximation of false positives diminishes as j (the quantity of bits in the exhibit) increments, and increments as i (the quantity of embedded components) increments.

4.2 False Negative

When non-embedded components are tried against the filter, false positives can happen. For each checked component the false positive probability is similar. While, the false negatives happen for embedded components and each component has an alternate approximation of being false negative. If no less than one of its stamped bits is reversed by a subsequent component and that bit stays rearranged until the finish of the inclusions then a component is said to be a false negative. As an outcome, the false-negative probability relies upon the inclusion request and

components embedded first having a larger possibility of bits transformed by consequent components.

The false negative probability can be intended out of the approximation that some particular bit of $(i-l)^{th}$ component is not reversed by the upcoming l lodged component, for $0 \leq l \leq i-1$.

Contingent upon the approach, it might be essential to diminish likelihood of false negatives to detriment of a much larger rate of false-positive.

4.2.1 Probability of False Negative

Then again, false negatives basically originate with the probabilistic characteristics of area delicate hashing capacities which hashes the proximate things to a similar piece having a large, yet not equivalent to 100 percent probability.

A false negative happens with a questioned thing q if $\| p, q \| \leq R$ and $\| h(p) \neq h(q) \| \leq R$ (hash miss as the denomination) of a LSH work when p is an individual from the questioned informational set S. The approximation for the collision of hashes with p ; q is at the very least P_1.

The false negative shows up if no less than one of hit bits is "0," which is unique in relation to false positives that happen assuming that every hit bits is set to "1." The hash function probability ought to be no under P_1 in one hash work for things p and q. Because of the hashing probabilistic property, regardless of the possibility that p and q don't crash for one hash work, the hit bit for q could similarly be "1" with supplements. For this situation, the false negative won't happen.

The tradeoff cost is the presentation of false negatives in participation questions, which did not exist in the standard channel. A false negative suggests not identifying a really embedded component.

The original Bloom Filter to reset bits never uses hash functions therefore it is impossible to have false negatives as a consequences as expected the false-negative approximation will be zero or almost negligible for the standard Bloom Filter. Approximation of false-negative is some monotonically diminishing capacity of the quantity of embedded components. At the point when

every bit of the standard Bloom Filter are at first settled, the false-positive likelihood achieves 100% . At this express, the channel is futile because it can't separate components of the set from outer components.

4.3 Hash collision

A collision is a circumstance when two unmistakable bits of information have a similar hash esteem.

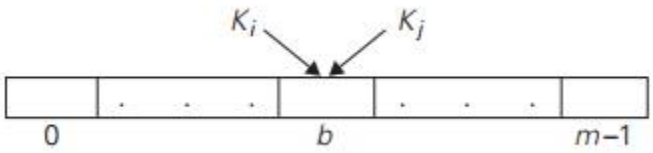


Fig.4 collision of two keys in hashing $h(k_i)=h(k_j)$

A Collision Attack is an attempt to find two information strings of a hash work that make a comparative hash result. Since hash limits have unbounded information length and a predefined yield length, there is unavoidably going to be the probability of two one of a kind information sources that convey a comparative yield hash. If two separate wellsprings of information make a comparative hash yield, it is known as a crash. This crash would then be able to be abused by any application that thinks about two hashes together –, for example, secret key hashes, record respectability checks, and so on.

The chances of a crash are obviously low, particularly so for capacities with extensive yield sizes. However as accessible computational power expands, the capacity to savage compel hash impacts turns out to be increasingly doable.

A false positive happens when a non rough thing has"1" as its bits being hashed for each L region delicate hash works in the state of hash collision. The false positive likelihood is henceforth firmly connected to the probability of collision that takes after s-stable appropriation.

Give $f_s(t)$ a chance to be the function of probability density of s-stable conveyance. As indicated by the conclusion the likelihood that two things p_i and q to collide for a LSH, q_i, is

$$Pr_{\cdot a,b}[h_{a,b}(p_i) = h_{a,b}(q)] = \int_0^{\omega} \frac{1}{\kappa} f_s\left(\frac{t}{\kappa}\right)\left(1 - \frac{t}{\omega}\right)dt,$$

where $k = \| p_i - q \|_s$, vector a is taken from an s-stable distribution and vector b is uniformly drawn from [0,w). Account that qi is neither less than P_1 if $k \leq R$ nor larger than P_2 if $k > cR$.

The false positive for the subject of thing q suggests for each hash work, it must crash into a thing in set S. We use q_i to mean the crash likelihood to thing p_i. Subsequently, the likelihood is $(1-q_i)$ for hash work as q does not struggle with p_i.

Chapter 5

Hash Functions

5.1 Hash Function

The hash functions utilized as a part of a Bloom filter ought to be free and consistently disseminated. They ought to likewise be as quick as could reasonably be expected (cryptographic hashes, for example, sha1, however broadly utilized in this way are bad decisions). Hashing is considered as one of the most widely recognized approaches to speak to sets. Every item of the set is hashed into Θ(log n) bits, and an (arranged) rundown of hash esteems at that point speaks to the set. This proposition yields little mistake probabilities.

Hashing can be said as the change among the arrangement of characters to an ordinarily short settled length regard or key that addresses the primary string generally called the hash. It is utilized for ordering and recovering the information things in huge informational index since it plays out the pursuit speedier since just hash esteem is sought rather than the first incentive from the m-bit exhibit. Blossom channel likewise utilize Hash capacities to figure the hash esteem and afterward store these hashes in sprout channel cluster of m–bit.

5.2 Practical Hash Functions used for bloom filter

A Bloom filter needs some unvaried and non-dependent hash functions. In the event that the hash work properties are compromised, the genuine false positive proportion can be much more awful than the hypothetical investigation[10].

Bloom filter speed depends on the count of hash functions utilized, the more hash functions, the slower a bloom filter, and the speedier it tops off. In the event that you have excessively few, be that as it may, you may endure an excessive number of false positives.

5.2.1 Cryptographic Hash Functions

They have great arbitrariness confirmation, so they are famous alternative for actualizing Bloom filters. For instance, MD5 is used in Bloom filter executions. The complexity of MD5 is high as

the cost is relative to key size. It requires 6.8 CPU cycles for each byte all things considered. The cost on hashing long keys can be restrictive for a few applications.

5.2.2 Non- Cryptographic Hash Functions

A few moderately straight forward hash capacities, for example, CRC32, FNV and BKDR, are frequently used to execute Bloom filters. Likewise, the calculation many-sided quality i.e. hash function complexity is corresponding to the component estimate. While these hash capacities are less calculation escalated than the cryptographic hash functions, their haphazardness is not as great, which means higher Bloom filter false positive proportions.

5.2.3 Universal Hash Functions

Hash functions can be chosen from a group of hash functions with a specific numerical property. The Bloom filter usage with these hash capacities can reaches the ideal false positive proportion. Since the universal hash functions should be "arbitrarily" chosen from a family, the practical execution still need the guidance of hash functions which are traditional (i.e., cryptographic and non-cryptographic hash functions).

5.3 The ideal hash functions count

The number to be utilized are the quantity of hash capacities, w, must be a positive whole number. Setting this imperative aside, for a given j and i, the estimation of w that limits the false positive probability is

$$\boldsymbol{w = \frac{j}{i} \ln 2}$$

The number of bits needed, w, given i (the quantity of embedded components) and coveted false positive likelihood p (and expecting the ideal estimation of w is utilized) can be processed by replacing ideal estimation of w in the prospect articulation :

$$\boldsymbol{p = \left(1 - e^{-\left(\frac{j}{i} \ln 2\right)\frac{j}{i}}\right)^{\frac{j}{i} \ln 2}}$$

This equation can be made more simpler as :

$$\ln p = -\frac{j}{i}(\ln 2)^2$$

which implies corresponding hash functions which are ideal are :

$$w = -\frac{\ln p}{\ln 2}$$

This implies that for a stated false positive approximation p, the span of a Bloom filter *j* will be proportionate to quantity of components being separated *i* and the needed hash works amount just relies upon the objective false positive likelihood *p*.

The value of *j* in above equation imprecise due to three grounds

i. The one with minimal distress , which actually is an adept asymptotic estimation (i.e it confine as j→∞) is that it approximates $1 - \frac{1}{j}$ as e^{-j}.

ii. It is of comparatively much more distress, it expect that amid the test of membership the occasion that one tried bit is set to 1 is free of the occasion that whatever other tried bit is set to 1.

iii. This is the most distress factor , it expects $w = \frac{j}{i}\ln 2$ to be felicitously integral.

Chapter 6

Implementation

6.1 Locality Sensitivity Hashing

LSH maps comparative things into the same hash pails with inflated probability to perform duty principle memory calculations for closeness seek. The process of LSH includes hashing each query item q to the pails in various hash tables and integrating each item in the selected buckets by grading them in accordance with the distance to the query points. Afterwards, items which are near to the queried one are chosen. There is a characteristics of LSH family that items which are comparatively near to each other than those which are distant from one another has the higher probability of collision.

6.1.1 Definition

Let S be a storage space of elements and ‖*‖ as the measure of distance among two items.

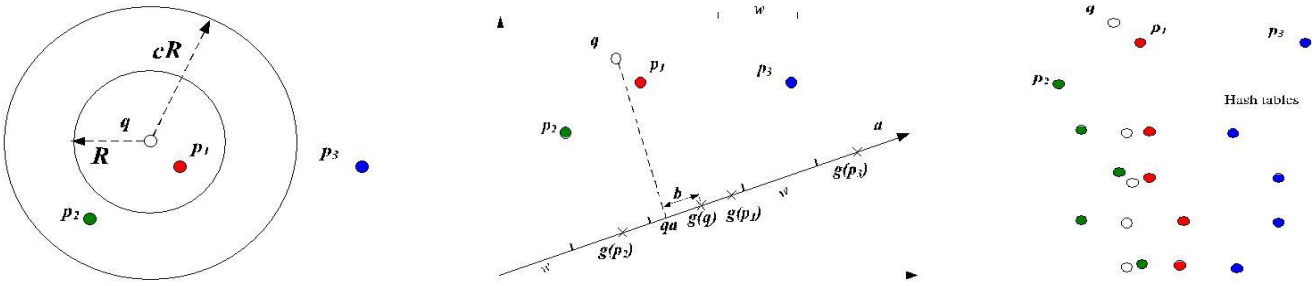
Family of LSH function, i.e., $IH = \{h : S \rightarrow U\}$ is called (R, cR, P_1,P_2) tactful for function od distance ‖*‖ if for any p, q Є S.

- If ‖ p,q‖ ≤ R then P_{rIH} [h(p) = h(q)] ≥ P_1.
- If ‖ p,q‖ > cR then P_{rIH} [h(p) = h(q)] ≤ P_2.

It is necessitate to select c > 1 and $P_1 > P_2$ to enable similarity search. By and by, it is required to grow the crevice amongst P_1 and P_2 by utilizing different hash functions. In light of s-stable dispersion, to various LSH families of l_s norms corresponds distance functions ‖*‖ so as to permit every hash function $h_{a,b} : Rd \rightarrow Z$ to delineate d-dimensional vector v on an arrangement of whole numbers. In IH, hash function can be characterized as:

$$h_{a,b}(v) = \left\lfloor \frac{a.v + b}{g} \right\rfloor$$

where a will be a d-dimensional arbitrary vector with picked passages following a s-stable dissemination and b is a genuine number picked consistently within the range (0,g) where g is an extensive steady[1].



(a) Illustration of measured distance. (b) Geometry result of hash functions. (c) Storage form.

Fig.5 Computation display of accurate and false outcomes as a result of probabilistic hash collision

Fig.5 demonstrates a case to show the LSH working plan as far as measured separation, geometry aftereffect of hash capacities, and the capacity type of hash tables.

In particular, by analyzing separation amongst two points in a metric space LSH can decide the proximate territory between two focuses. On the off chance that the hover focused at q with span R covers no less than one point, e.g., p_1, as appeared in Fig. 5a, LSH can furnish a point without any than cR separation to q as a result of query. Within R and cR it can be observed that there is a questionable space in LSH and from one of points whether it could be p_1 or p_2 the query point q will get an answer, as two of the points situates inside the distance cR, i.e $\| p_1,q\| < cR$ and $\| p_2,q\| < cR$. Then again, because of the massive separation than cR, the point p_3 is not near the queried q.

Fig.5b additionally shows the geometry aftereffect in a 2D space of locality sensitive hash function. q.a is represented as dot product, given that a is a vector and q as the query point. Out of the interval [0,w), a vector b is chosen statically. H(q) is the denotation of dot product q.a which actually is the projection of q onto a , out of this with a moved distance b , g(p) is achieved. Every vector compares to the position arrangement number of point q , as line of vector a with length w is separated into intervals. At such scenario, proximate focuses, e.g., q and p_1, have large approximation to be situated into a similar interval.

As appeared in Fig.5c LSH executes the locality-sensitive method by utilizing various hash tables, keeping in mind the end goal to create an eminent probability of control inside a bucket for proximate things. The query point s is stored within a hash table bucket along with elevated probability, as when compared to point p_1 q is a nearby neighbor. For example in the first and second hash tables, they are in the same bucket. Interestingly, for point p_3 there is a low chance of finding it together with point q in a single bucket because they have a large Euclidean separation. Together with this because of the uncertain location for p_2 as it is located within R and cR, the LSH represents its approximate property.

6.2 MD5 Algorithm

The MD5 computation is an extensively used hash work making a 128 piece hash regard. Slighting how MD5 was at first intended to be utilized as a cryptographic hash work, it has been found to experience the malicious effects of sweeping condition of being presented to the likelihood of being assaulted. It can at show be used as a checksum to affirm data reliability, yet just against unplanned debasement [8].

Algorithm :

i. Padding bits and Append Length

Cushion the bits with "0" and "1" as it is a need , first and last individually until the consequent ≠ bit length is not proportional to 448 mod 512, and the rest of bit length of the main message as 64-bit number. The last piece length of the message which is authoritatively cushioned is 512 N for a genuine whole number N.

ii. Spilt the inserted data to 512-bit blocks

Subdivide the message which was padded in earlier step into m_1, m_2,...., m_n i.e. to n blocks of 512 bits continuously.

iii. Initialize the variables of Channing

Initialize 32-bit number in chaining variables form and these values are represented in hash only.

iv. Process blocks

The four buffers messages (content) consolidated with the information words, utilizing the four assistant function. 4 rounds are executed and every round includes 16 rudimentary operations. The Processing P block is exerted to the four by using message word and constant.

v. Hashed Output

To 128 bits message digest 5 (MD5) 4 rounds are conducted .

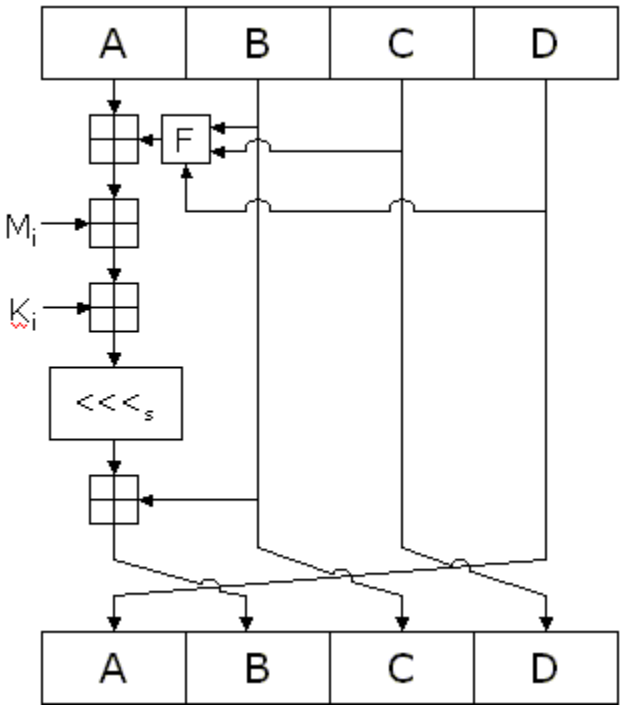


Fig.6 MD5 Procedure

6.3 SHA-1 Algorithm

Secure Hash Algorithm is a cryptographic hash work which produces a 160 piece hash esteem called as message digest. SHA-1 outlines some segment of a couple of for the most part used security applications and traditions, including TLS and SSL, PGP, SSH, S/MIME, and IPsec[8].

Algorithm:

i. Padding

At the remainder of bona fide message add padding length of 64 bits and multiple of 512.

ii. Include length

To such progression the barring length is figured.

iii. Split the inserted data to blocks of 512 bits.

At this part separate the contribution to the 512 piece squares

iv. Initialize chaining variables

Initialize chaining variables at this progression and initialize v. affixing factors of each of 32 bit equals 160 bit of aggregate.

v. Process Blocks

a. Replicate variables of chaining.

b. Separate 512 to 16 sub blocks

c. Process every rounds with 20 steps each .

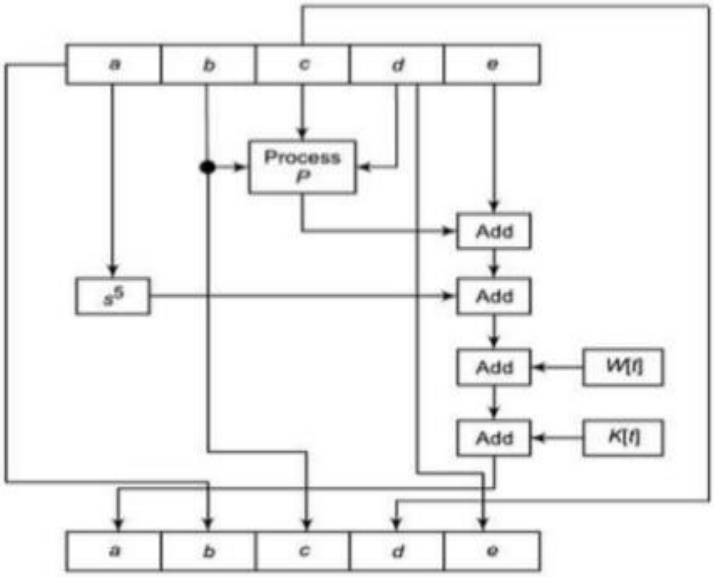


Fig.7 SHA1 Procedure

Chapter 7

Result and Analysis

The proposed method is used to obtain the following result by implementing three different hash functions. The time consumed taken by individual hashes are obtained and a comparative result is produced to determine which of the hash method among the three implemented hashes is best when the bloom filter of optimal size is used. Subsequently, the number of collisions are obtained from the implementation of all the hashing methods. Along with all these the false positive is also produced as a factor to compare the best among the three.

The three hash methods are Locality Sensitivity Bloom Filter , MD5 and SHA1. All the three produces different output with different value of p i.e. probability which equivalent to false positive probability :

$$f_p = \left(1 - \left(1 - \frac{1}{m}\right)\right)^{(k*n)^k}$$

The optimal size of bloom filter is computed as :

$$m = -ceil\left(\frac{(n * \log p)}{((\log 2)^2)}\right)$$

The diagram below displays the output of the implemented code which shows the difference in the time consumption by the three hash methods having the same optimal size of the bloom filter and the value of p.

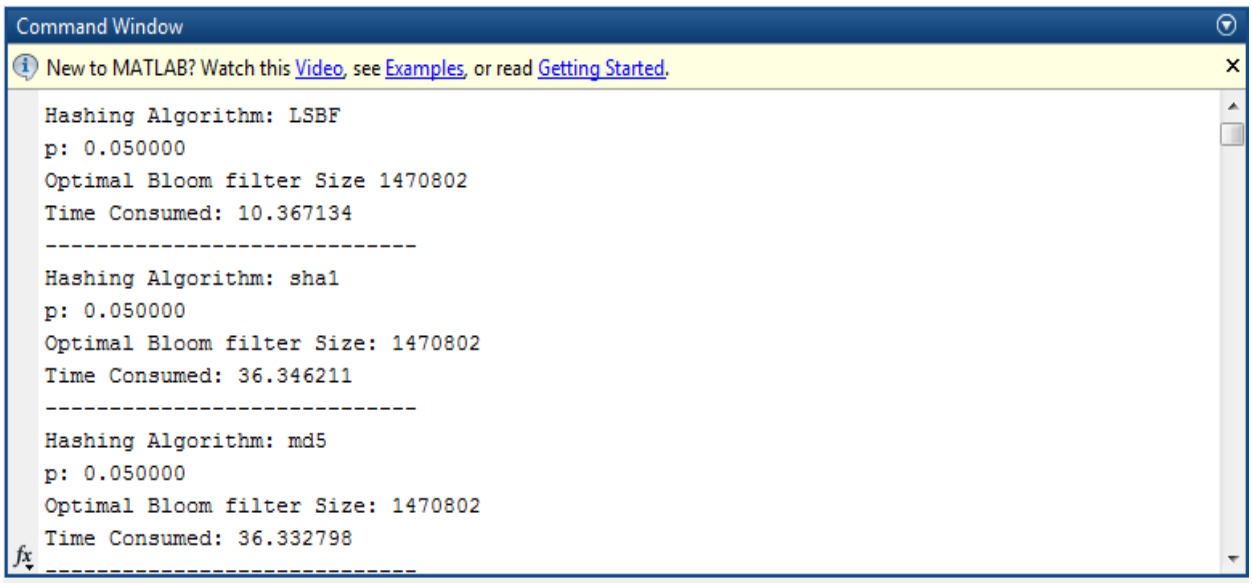


Fig.8 Difference in time consumed with p=0.05 and BF size = 1470802

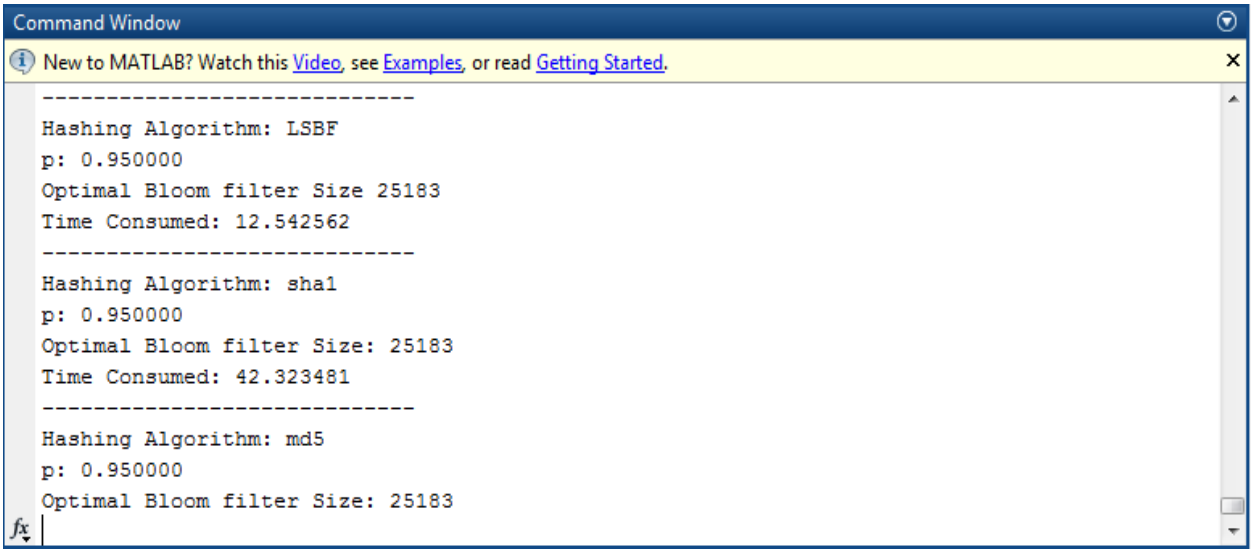


Fig.9 Difference in time consumed with p=0.95 and BF size = 25183

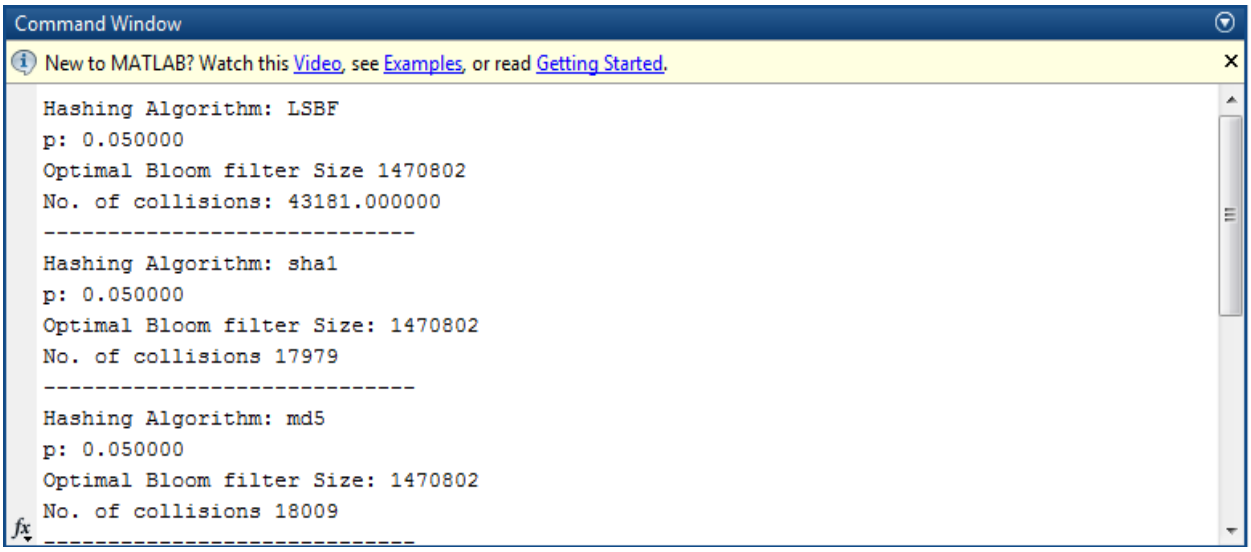


Fig.10 Difference in no. of collisions with p=0.05 and BF size = 1470802

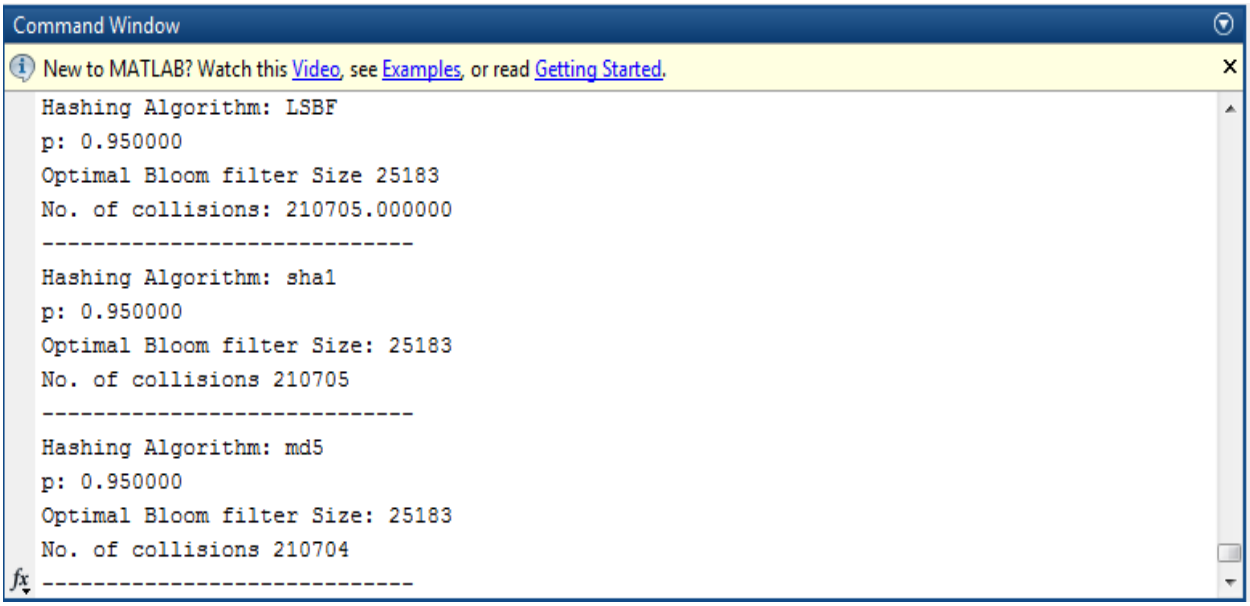


Fig.11 Difference in no. of collisions with p=0.95 and BF size = 210704

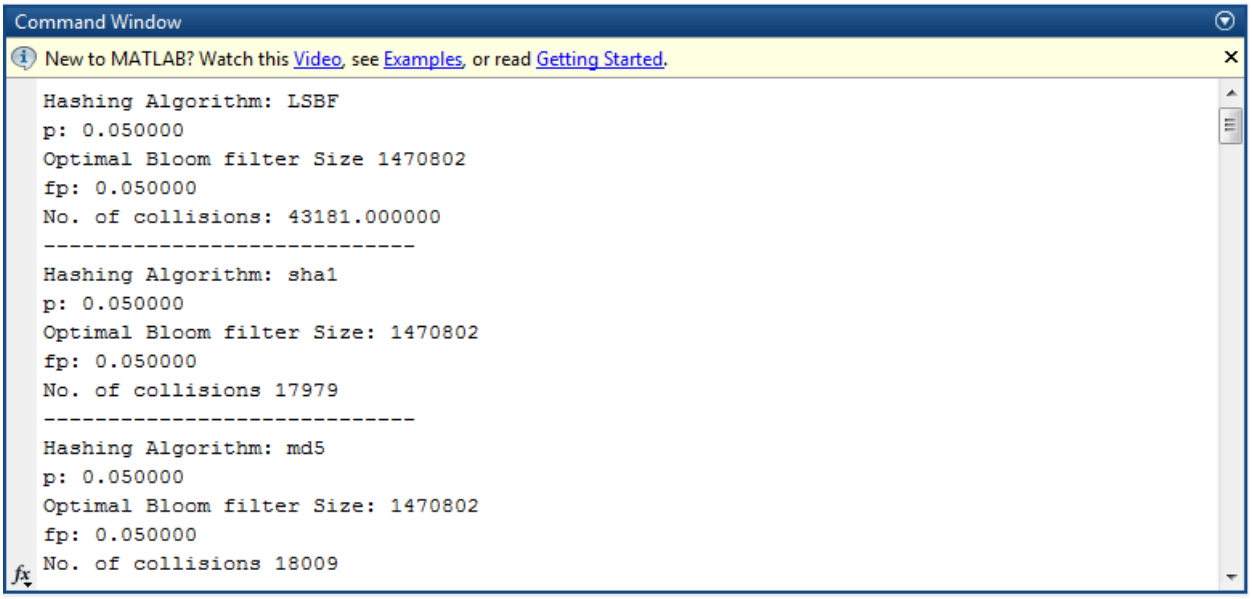


Fig. 12 Comparison of three hash function with similar fp = 0.05

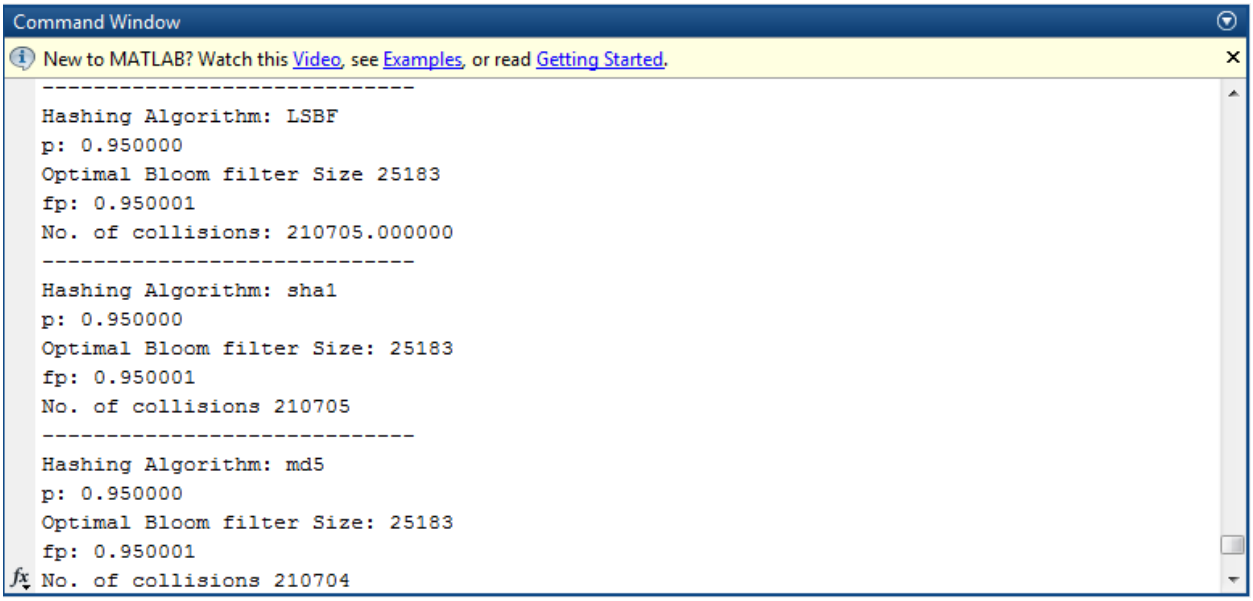


Fig. 13 Comparison of three hash function with similar fp = 0.95

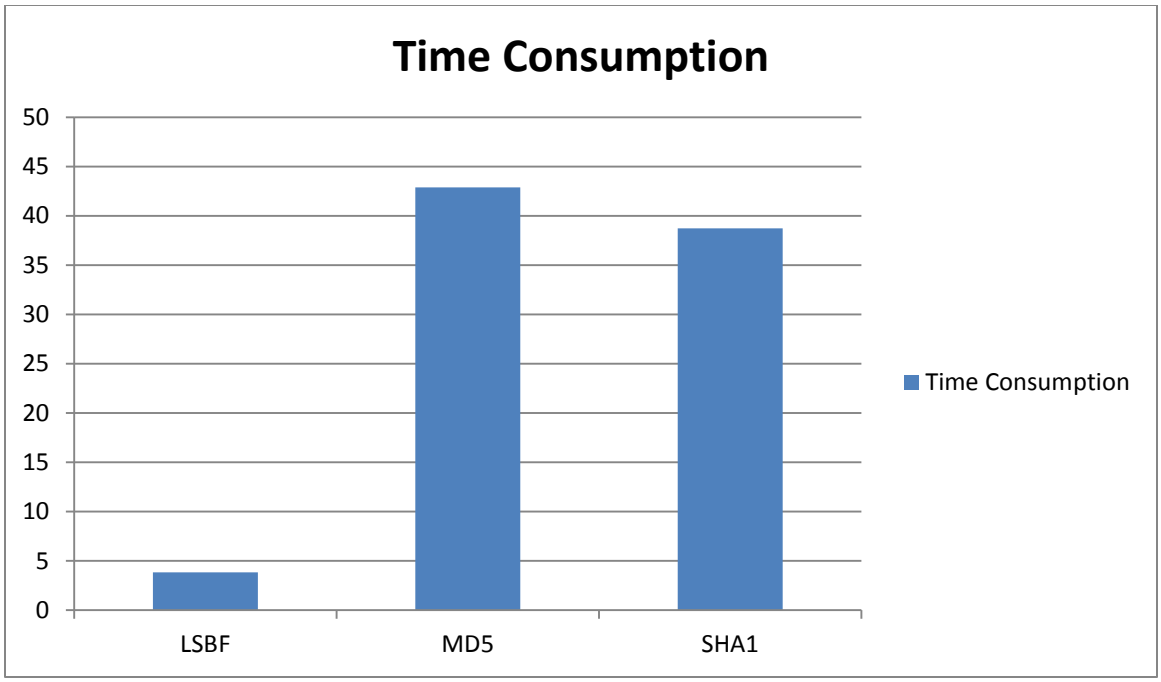


Fig.14 Comparison in time consumption with p=0.05

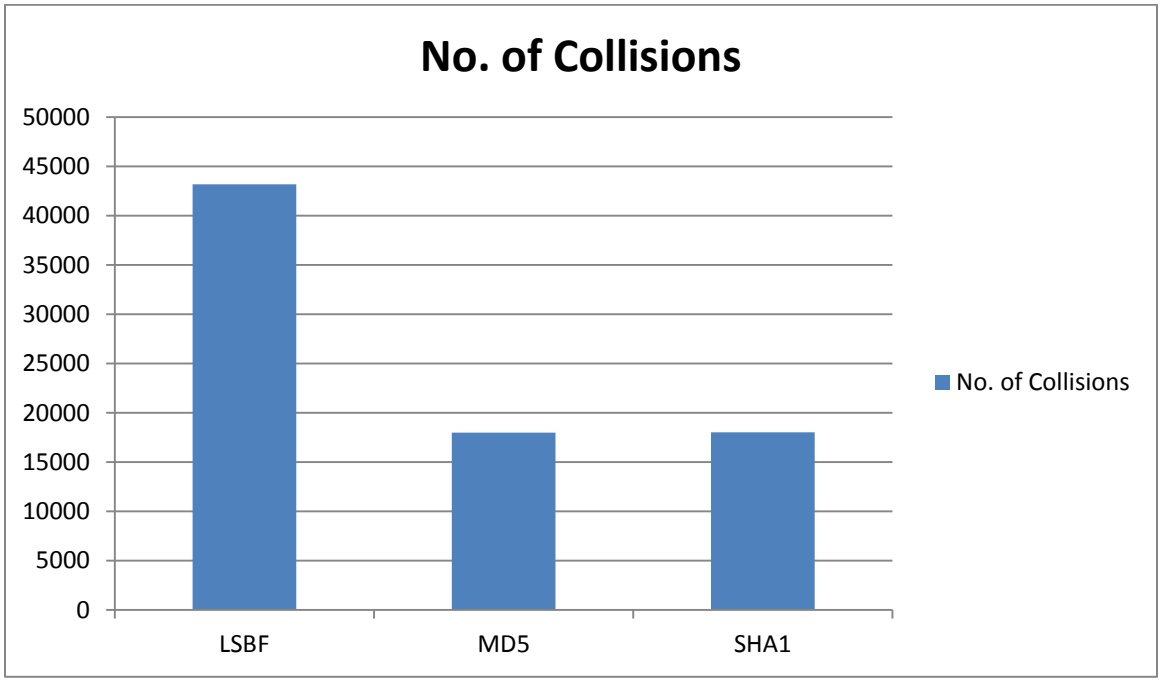


Fig.15 Comparison in no of collisions with p=0.05

Chapter 8

Conclusion and Future Scope

8.1 Conclusion

From this research work it can be concluded that there is a variation in all the three aspects that is in number of collisions , false positive rate and time consumption in each of the three hash methods used. The Locality Sensitive Bloom filter takes the least time compared to MD5 and SHA1 when the same size of Bloom Filter is implemented. This result exhibits that LSBF can be used where there is a need to diminish the time complexity. MD5 and SHA1 exhibits the best outcome when there is a speak about the number of collisions. Both MD5 and SHA1 produces almost equivalent number of collisions , though amongst the two MD5 shows better results.

8.2 Future Scope

In this work LSBF has been implemented in C language therefore as a result it shows optimal result when time consumption is discussed, while the other two, MD5 and SHA1, are implemented in JAVA language. Despite the fact that MD5 and SHA1 exhibits less number of collisions LSBF is better. So as to improve the performance of Bloom Filter it can be implemented in any other language using MD5 or SHA1. This will in future delivers much better results when implemented in some other language and with the use of other hash functions such as SHA256 and SHA384 as with the utilization of these hash functions the number of collisions will be reduced.

References

1. Yu Hua; Bin Xiao; Veeravalli, Dan Feng, "Locality-Sensitive Bloom Filter for Approximate Membership Query," Computers,IEEE Transactions on, vol.61, no.6, pp. 817, 830, June 2012 doi:10.1109 / TC.2011.108.
2. L. Carter, R. Floyd, J. Gill, G. Markowsky, and M. Wegman, "Exact and Approximate Membership Testers," *Proc. 10th Ann. ACM Symp. Theory of Computing (STOC '78), pp. 59-65, 1978.*
3. Mayank Bhushan , Monica Singh ,Sumit K Yadav ," Big Data query optimization by using Locality Sensitive Bloom Filter" *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom).*
4. Jiangbo Qian, Qiang Zhu, Senior Member, IEEE, and Huahui Chen, "Multi-Granularity Locality-Sensitive Bloom Filter", *IEEE TRANSACTIONS ON COMPUTERS, VOL. 64, NO. 12, DECEMBER 2015.*
5. B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM, vol. 7, no. 13, pp. 442–426, July 1970.*
6. A. Broder and Mitzenmacher, "Using multiple Hash Functions to Improve IP Lookups," *Proc. IEEE INFOCOM, pp. 1454-1463, 2001.*
7. J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," in *Proc. ACM SIGMOD Int. Conf. Manage. Data, 2012, pp. 541–552.*
8. Piyush Gupta, Sandeep Kumar, "A Comparative Analysis of SHA and MD5 Algorithm", *International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 4492-4495.*
9. A. A. Iqbal, M. Ott, and A. Seneviratne, "Simplistic Hashing for Building a Better Bloom Filter on Randomized Data," in *The 13th In-ternational Conference on Network-Based Information Systems (NBiS)2010* .
10. Jianyuan Lu, Tong Yang, Yi Wang, Huichen Dai, Linxiao Jin, Haoyu Song , an Bin Liu, "One Hashing Bloom Filter", *IWQoS 2015.*

11. Andoni, A. and Indyk, P. 2006., "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions", In *Proceedings of the Symposium on Foundations of Computer Science.*
12. J. L. Carter and M. N. Wegman. ,"Universal classes of hash functions", *Journal of Computer and System Sciences, 18, pages 143-154, 1979.*
13. G. MARKOWSKY, J. L. CARTER, AND M. N. WEGMAN," Analysis of a universal class of hash functions" , in *Proceedings of the Seventh Mathematical Foundations of Computer Science Conference Lecture Notes in Computer Science, Vol. 64, Springer-Verlag, Berlin.*
14. A. Z. Broder and M. Mitzenmacher. "Network Applications of Bloom Filters: A Survey." *In Proceedings of the Fortieth Annual Allerton Conference on Communication, Control, and Computing. CD-ROM. Coordinated Science Laboratory and the Department of Electrical and Computer Engineering of the University of Illinois at Urbana-Champaign, 2002.*
15. S. Cohen and Y. Matias. "Spectral Bloom Filters." *In Proceedings of the 2003 ACM SIGMOD International Conference on the Management of Data, pp. 241—252. New York: ACM Press, 2003.*
16. Laufer, Rafael P., Pedro B. Velloso, and Otto Carlos MB Duarte. "A generalized bloom filter to secure distributed network applications." *Computer Networks* 55.8 (2011): 1804-1819.
17. M. Mitzenmacher. "Compressed Bloom Filters." *IEEE/ACM Transactions on Networking 10:5 (2002), 604—612.*
18. F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "Beyond Bloom Filters: From Approximate Membership Checks to Approximate State Machines," in *Proceedings of the ACM SIGCOMM'06 Conference, Pisa, Italy, Nov. 2006, pp. 315–326.*
19. Y. Zhu and H. Jiang. False rate analysis of bloom filter replicas in distributed systems. In *Proc. 35th International Conference on Parallel Processing (ICPP), pages 255–262, Ohio, USA, August 2006.*
20. A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Comm.ACM, vol. 51, no. 1, pp. 117-122, 2008.*

21. Laufer, Rafael P., Pedro B. Velloso, and O. C. M. B. Duarte. "Generalized bloom filters." *Electrical Engineering Program, COPPE/UFRJ, Tech. Rep. GTA-05-43* (2005).
22. L. Fan, P. Cao, J. Almeida and A. Z. Broder, "Summary Cache: A Scalable Wide-area Cache Sharing Protocol," in *Proceedings of ACM SIGCOMM '98.*
23. Guo, Deke, et al. "False negative problem of counting bloom filter." *IEEE Transactions on Knowledge and Data Engineering* 22.5 (2010): 651-664.