Accurate Counting Bloom Filters for Large-Scale Data Processing

ACCURATE COUNTING BLOOM FILTERS FOR LARGE-SCALE DATA PROCESSING

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

N Mahema Reddy (221810303036)
Gunturu Sai Ram (221810303019)
Anvita Veeramallu (221810303003)
Chagi Nikhil (221810303066)

Under the esteemed guidance of

Dr. Pravin Mundhe
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GITAM (Deemed to be University)
HYDERABAD
APRIL-2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to University)

DECLARATION

We submit this Major Project entitled **“Accurate Counting Bloom Filters for Large-Scale Data Processing”** is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University), Hyderabad campus submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. The work has not been submitted to any other college or university for the award of any degree or diploma.

DATE: 11-04-2022

Registration No(s).	**Name**	**Signature(s)**
221810303036	N Mahema Reddy	
221810303019	Gunturu Sai Ram	
221810303003	Anvita Veeramallu	
221810303066	Chagi Nikhil	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the major project report entitled **"Accurate Counting Bloom Filters for Large-Scale Data Processing"** is being submitted by N Mahema Reddy (221810303036**),** Gunturu Sai Ram (221810303019)**,** Anvita Veeramallu (221810303003)**,** and Chagi Nikhil (221810303066) in partial fulfillment of the requirement for the award of Bachelor of Technology in "**COMPUTER SCIENCE ENGINEERING**" at GITAM University, Hyderabad.

It is faithful record work carried out by them at the **Computer Science Engineering Department,** GITAM School of Technology, GITAM Deemed to be University, Hyderabad campus under my guidance and supervision.

Project Guide	**Project Coordinator**	**Head of the Department**
Dr. Pravin Mundhe	**Dr. S. Aparna**	**Prof. S. Phani Kumar**
Assistant Professor	Assistant Professor	Professor and HOD
Department of CSE	Department of CSE	Department of CSE

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

1. ABSTRACT

Bloom filters are space-efficient randomized data structures that allow for false positives in query searches. On the given dynamic sets that can be updated through insertions and deletions, Counting Bloom Filters (CBFs) conduct the same actions. CBFs have been widely employed in memory to reduce the size of datasets for large-scale data processing on massive clusters.

For filtering out more redundant data, the false positive likelihood of the CBF should be generated low. This research suggests a multilevel optimization strategy for minimizing the false positive probability of an Accurate Counting Bloom Filter (ACBF). We present an optimized ACBF that maximizes the first level size to reduce false positives while keeping the same functionality as CBF.

We discovered counting bloom filters' functioning, efficiency, and error rate in the previous project (mini-project). And we found that it performs better than the standard Bloom Filter.

CHAPTER 2

2. INTRODUCTION

A multilevel optimization strategy to develop an Accurate Counting Bloom Filter is presented in this study. The purpose of the Accurate Counting Bloom Filter is to limit the likelihood of a false positive. The counter vector is partitioned into numerous levels, and offsets indexing is used to manage them. The first level of ACBFs is used to perform set membership queries, while the other levels calculate insertion and deletion counters. We present an algorithm that is ACBF that maximizes the first level size while preserving the same methodology as the regular Counting Bloom Filter to reduce the false positive probability. Visualization studies show that Accurate CBF outperforms CBF in false-positive probability at identical memory usage..

A Bloom filter is a basic randomized data structure which describes a set in order to facilitate rapid membership searches. Bloom filters employ separate hash algorithms to represent elements using bits. When querying an element, False positives, or true replies when an element isn't in the set, are allowed by Bloom filters. However, false negatives or giving false when the element is not in the set are not acceptable. However, when the chance of false positives is low enough, the space savings of Bloom filters generally offset this disadvantage.

Because normal Bloom filters don't allow you to delete items, Bloom filters and variants have become extremely popular. The Counting Bloom Filter (CBF) is a well-known variation that permits the set to alter dynamically through element insertion and deletion. Bloom filters are extended by CBF, which uses a fixed-size counter instead of a single bit for each vector entry. The corresponding counters are incremented when an element is added into CBF; By decrementing the counts, deletions may now be done safely. For most applications, we use four bits per counter to avoid counter overflow.

Large-scale data processing, on the other hand, is a huge performance barrier for MapReduce. For starters, online search engines and log processing are examples of data-intensive applications. deal with massive amounts of data. China Mobile, for example, must process 5–8 TB of call records each day. Every day, Facebook collects about six TB of fresh log data. Distributing data over tens of thousands of low-cost devices for division is time intensive for these applications.

Memory consumption, processing overhead, and false positive probability are the performance indicators of CBF. The processing overhead, which dominates the CBF throughput, For each primitive operation, is the number of memory accesses. The counter vector size of CBF is the memory consumption. Insertions and deletions are normally supported by four bits per counter. Because the counters consume so much memory, many modifications have been proposed to reduce CBF's memory usage by fitting the entire filter into a small amount of high-capacity memory (i.e., SRAMs). The chance of a false positive is when an element is claimed to be a set member when it is not. A compromise exists between the chance of a false positive and the quantity of memory used.

2.1 ACBF CONSTRUCTION:

The primary idea behind ACBF is, partition the counter vector into numerous levels using a multilayer technique for increased accuracy. We can now separate ACBF's query and insertion/deletion operations using this method. This distinction is used to reduce the likelihood of false positives while updating dynamic sets. This is owing to the fact that CBF is being monitored. We find that the CBF counter vector is ideal for rapidly modifying the counts at the bottom by incrementing or decrementing them at the expense of false positive probability. The Figure depicts a simple CBF example using and, where is the number of counters,is the number of hash functions, and is the maximum number of entries.

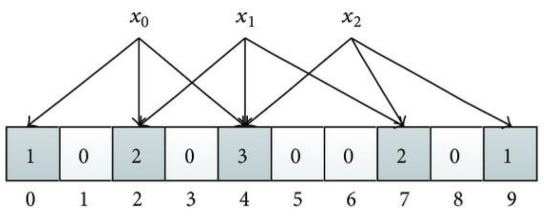


Fig 2.1.1 ACBF Construction

2.2 COUNTING BLOOM FILTERS:

Bloom Filters have the identical functions as Counting Bloom Filters; however, instead of adding counters to each slot, we add counters to each slot in Counting Bloom Filters (0 or 1). That means we can remove elements from the data structure; something Bloom Filters cannot do.

2.3 COUNTING BLOOM FILTERS' FUNCTIONS:

2.3.1 Initialization:

To begin using Counting Bloom Filters, we must first create an array of m slots, each of which will contain the value of a counter. Because no items have been added to the data structure yet, all of the slots will be set to zero at the start.

Bloom filters are fantastic because they allow us to rapidly determine whether a given collection contains a specific value. They achieve this result while taking up the least amount of space and providing O(1) inserts and lookups. It's no surprise that Bloom filters and its derivatives (a class of approximate set membership methods) are widely employed. Bloom filters are one of my favorite applications of hash functions, which are used in a lot of fantastic algorithms.

Bloom filters are an example of an algorithm that does not exploit the data distribution (in fact, most of the interesting variants include techniques to deal with the fact that the distribution isn't necessarily uniform, rather than abusing it). Whether or if a particular query is contained within a piece of data is one of the most basic queries one may ask of that. Is q, the query, a member of S, the number of observations we've chosen?

This membership set query can be found in a variety of computing environments, including databases, network routing, and firewalls. S is stored in its whole, and q is compared to each element.. One might query set membership. There are, however, other space-saving options. The Bloom filter gives up space in exchange for a lower allowable fp rate caused by hash collisions. However, it makes a one-sided error: if an element q is present in S, it will always be recognised. It doesn't produce any false negatives. Bloom Filters can be found in a wide variety of industrial systems, including network security. Bloom Filters are popular because of their good compression, but they also include capability for dynamic updates. In O(1) time, new elements can be added. Not all approximation set membership data structures are like this. Exact hashing, for example, saves 40% of the time.

We concentrate on the data stream computing paradigm, which assumes that incoming observations are transitory and can only be viewed once. Many real-world applications are covered here, including database query serving, network traffic analysis, and reinforcement learning complex fields. Creating a data structure for approximation set membership that is not just Bloom Filters are more compressive., but also more efficient.

We add to the expanding body of knowledge about employing neural networks to replace heuristic-based techniques that don't use data dispersion to your advantage. Bloom Filters, for example, are unconcerned about data distribution. Previous research on To explain our findings, we first present the Bloom filter we learned informally. It gives a compact representation of a collection of keys K that permits membership queries, just like a conventional Bloom filter. (The keys are sometimes referred to as elements.) A trained Bloom filter, given a key y, always returns yes if y is in K, ensuring no false negatives, and generally returns no if y is not in K, however this may result in false positives. The function functions as a pre-filter that offers a probabilistic estimate that a query key y is in K. This learning function can be used to determine whether the key is in K, and a smaller backup Bloom filter can be used to avoid false negatives.

Our more formal model sheds light on how learnt Bloom filters work and how they could be improved. We explain how to predict what size the learning function must obtain in order to increase performance; present a simple strategy for optimizing learnt Bloom filters; and illustrate our approach may be applicable for other comparable structures.

CHAPTER 3

3. LITERATURE SURVEY

3.1 BASE PAPER:

For a long time, verifiable data erasure has already been intensively researched., yielding a plethora of solutions.Investigation carried on the objective of safe data deletion and suggested a pivotal attribute-based encryption mechanism for data access control with finer granularity and destruction assurance. They achieve data deletion by eliminating the attributes and achieving verifiability with the Merkle hash tree, but their technique requires a trustworthy authority. For multi-copy deletion, he created the Associated deletion approach (ADM), which achieves data integrity verification and proven deletion through the use of a pre-deleting sequence and MHT. Their solution, however, necessitates the use of a TTP to maintain the data keys. In 2018, he introduced a Blockchain-based cloud data erasure method. The deletion is carried out by the cloud, which then publishes the relevant deletion evidence on Blockchain. The deletion outcome can then be checked by any verifier by confirming the deletion evidence. Additionally, they eliminate the need for a TTP as a bottleneck. Although all of these techniques can guarantee data destruction, they can not guarantee secure data transport. Many ways have been proposed to migrate data from one cloud to another and remove the transferred data from the original cloud. Yu et al proposed a Provable data possession system in 2015, may be used for safe data movement. Their technique is the first, to our knowledge, to efficiently solve data transmission between two clouds. However, It is useless in the data deletion process since it re-encrypts data, requiring the data owner to send a large amount of data..

The false positive rate of the Counting Bloom Filter is larger than that of the standard Bloom Filter, which necessitates a larger memory footprint. CBF can eliminate erroneous negatives, but it has a significant likelihood of false positives and a large memory footprint.To limit the likelihood of false positives, the Counting Bloom Filter sacrifices memory footprint.. As a result, we offer countBF, a new counting Bloom Filter that addresses the difficulties stated above. The countBF algorithm can dramatically minimize the likelihood of false positives while maintaining a small memory footprint. In every aspect, CountBF outperforms the conventional Bloom Filter (SBF) and counting Bloom Filter, according to our results. The major aims of our proposed system are to minimize memory footprint, reduce fp probability, and accuracy improvement without sacrificing insertion query efficiency. The counting Bloom Filter we propose is comparatively the same to the traditional Bloom Filters. SBF does not have counters, whereas counting bloom filters does. In addition, countBF is built into the 2D Bloom Filter platform. Rather than using a bitmap array, 2DBF uses a 2D integer array. This bitmap is made up of a 2D integer array with each integer representing blocks of bits. By fine-tuning the murmur hash function, countBF improves its performance. Murmur is the finest accessible non-cryptographic string hash function.. There's also string hash functions, but they fail to improve performance or reduce the likelihood of fps. As a result, we assess the properties of SBF, countBF, and CBF utilizing a variety of test circumstances.

3.2 INTENDED AUDIENCE AND READING SUGGESTIONS:

This record is meant for prolonged engineers, clients, directors, analysts, and documentation writers. This paper appears at plan and execution requirements, in addition to conditions, framework highlights, outside interface requirements, and different non-vital requirements.

3.3 IDENTIFYING NEEDS:

Identify requirements to understand how they are performed in the market and how to defeat the competitors, confirm commercial companies or organizations' first and essential needs.
To this end, you must explore data based on all available variables.

CHAPTER 4

4. PROBLEM IDENTIFICATION & OBJECTIVES

4.1 PROBLEM STATEMENT:

Cloudsfer, a data transmission tool that is outsourced, has been built to ensure secure data migration by applying cryptographic algorithms to avoid the leaking of personal information throughout the transfer process.

4.2 EXISTING SYSTEM:

From the perspective of the data owners, the data reservation is unanticipated. In a nutshell, cloud storage is a service that allows you to store files on the cost-effective, but it necessarily faces major security issues, particularly in terms of integrity verification, safe data transfer, and verifiable deletion. If these issues are not addressed properly, the public may be hesitant to embrace and use cloud storage services.

4.3 PROPOSED SYSTEM:

We investigate the issues of securely transporting data and deletion of it in cloud storage, with a particular focus on achieving public verifiability. Then we offer a Bloom filter based counting system that not only allows for provable data transmission between two clouds but also allows for publicly verifiable data deletion. The verifier can discover malicious acts by evaluating the returned transfer and deletion evidence if the originating cloud server does not migrate or remove the data honestly. Furthermore, unlike previous alternatives, our suggested scheme does not require the use of a Trusted third party (TTP). We also use security analysis to show that our new approach may meet the needed design goals. Finally, simulation trials demonstrate that our novel concept is effective and feasible.

4.4 METHODOLOGY:

Step 1: Create a web application.

Step 2: Use the server as a cloud provider.

Step 3: Encryption and decryption of uploaded data

Step 4: Data deletion, updating, and so on.

Step 5: Predict the results

4.4.1 Project Management:

Data collection and preprocessing on the ground truth:

Processing

Post-production

Results and Analysis

4.5 HARDWARE AND SOFTWARE TO BE USED:

4.5.1 Hardware:

RAM : 8GB
Hard Disk : 500gb min
Graphic Processing unit if needed.

4.5.2 Software:

OS : Windows 11 Home
Technology : Python,
Domain : Machine Learning, cloud computing

The motive for this SRS document is to become aware of the necessities and abilities of the Intelligent Network Backup Tool. The SRS will describe how our group and the purchaser view the very last item, in addition to the functions and software it ought to possess. This record additionally consists of a listing of optionally available necessities that we intend to satisfy however aren't wished for the project's operation.

This degree evaluates the desired stipulations for Image Processing in a prepared manner. A few strategies are blanketed on this degree to assess the stipulations in an orderly manner. The first step in dissecting the framework's necessities is to understand the idea of framework for a radical inspection, and all instances are mentioned to assist higher recognition of the dataset's inquiry.

4.6 FEASIBILITY STUDY

Technology and Economic justification Accuracy Verification The confidence in verification is aimed to objectively and carefully disclose the benefits required to implement the current business or predicted path, natural discovery and problems, and future and long-term advanced opportunities. The two criteria for the trial were severe injury and the necessary motivation being satisfied under the most specific conditions.

Internal and SUV. For an unforgettable justification of business or corporate written records, explanations of objects and organizations, accounting explanations, operations, organizational motivation, investigation, game plan publishing, budget data, precise requirements, and obligation costs. Probabilities look at overall changes and wandering use before specific changes. Reachability can be divided into three categories:

• Feasibility from a financial standpoint

• Feasibility from a technical standpoint

• Feasibility of Operation

4.7 SYSTEM REQUIREMENTS:

4.7.1 Software Requirements:

Operating System: Windows
Framework: Jupyter
Programming Language : Python
IDE : Anaconda

4.7.2 Hardware Requirements:

Processor	:	Pentium 4
Hard disc	:	500GB
RAM	:	4GB

System with all standard accessories like keyboard, monitor, mouse, etc.

4.8 MODULES PRESENTED:

4.8.1 ACBF Construction:

The primary idea behind ACBF is to partition the counter vector into numerous levels using a multilayer technique for increased accuracy. We can now separate ACBF's query and insertion/deletion operations using this method. This distinction is used to reduce the likelihood of false positives while updating dynamic sets. This is owing to the fact that CBF is being monitored. We find that the CBF counter vector is ideal for rapid updates by incrementing or decrementing the counters at the expense of false positive probability.

4.8.2 Feature Extraction:

In machine learning and image processing, features are produced from the initial dataset, which speeds up the learning process. When an algorithm's input data becomes too large, it can be condensed down to a smaller set of features. The technique of obtaining a subset of the fundamental characteristics set is known as feature extraction [15]. The selected elements give information on the incoming data, allowing for a simplified representation of the agent rather than a complete representation of the initial data set.

4.8.3 Feature extraction method:

Central clustering is a clustering approach. For a fixed number of clusters, this method uses an exact mechanism that iteratively tries to find locations as cluster centers, essentially the identical mean points belonging to each cluster. Assign a category to each sample data set.

CHAPTER 5

5. SYSTEM METHODOLOGY

The System Design Document outlines the system's requirements, operating environment, system and subsystem architecture, file and database design, input and output formats, processing logic, human-machine interfaces,external interfaces, and detailed design.

This section describes the system in non-technical terms. You should provide a high-level architectural diagram of your system by categorizing it into subsystems. When appropriate, interfaces to external systems should be represented in high-level system architecture or subsystem diagrams. Optionally include high-level context diagrams of systems and subsystems. To determine the distribution of functional requirements in this design document, use the Requirements Traceability Matrix (RTM) in the Functional Requirements Document (FRD).

This part contains all assumptions made by the design team when developing the system design and the constraints of the system design (see, for example, assessing balances such as resource usage versus performance or conflicts with other systems).

Organizational code and name of the principal contact for developing the information system (replace if necessary). Project managers, user organizations, security managers, quality assurance (QA) managers, system advocates and configuration managers should be contacted.

5.1 SYSTEM ARCHITECTURE:

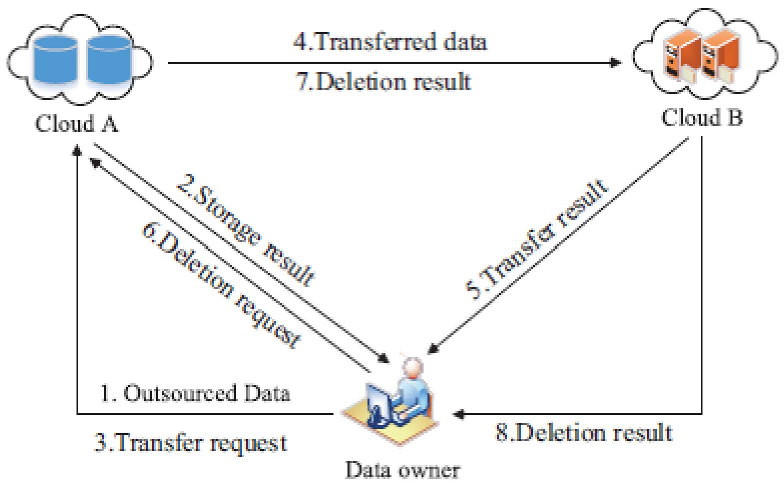


Fig 5.1.1 System architecture

5.2 UML DIAGRAM:

Unified Modeling Language, also abbreviated as UML, is a common language for choosing, visualizing, developing, and expressing structural programming components. UML is a visual language for developing programming plans. It may also show non-programming entities like a process stream in a gathering unit. UML gives a detailed explanation of the different stages in the process.

5.2.1 Use case Diagram:

Demonstrates the direct structure.

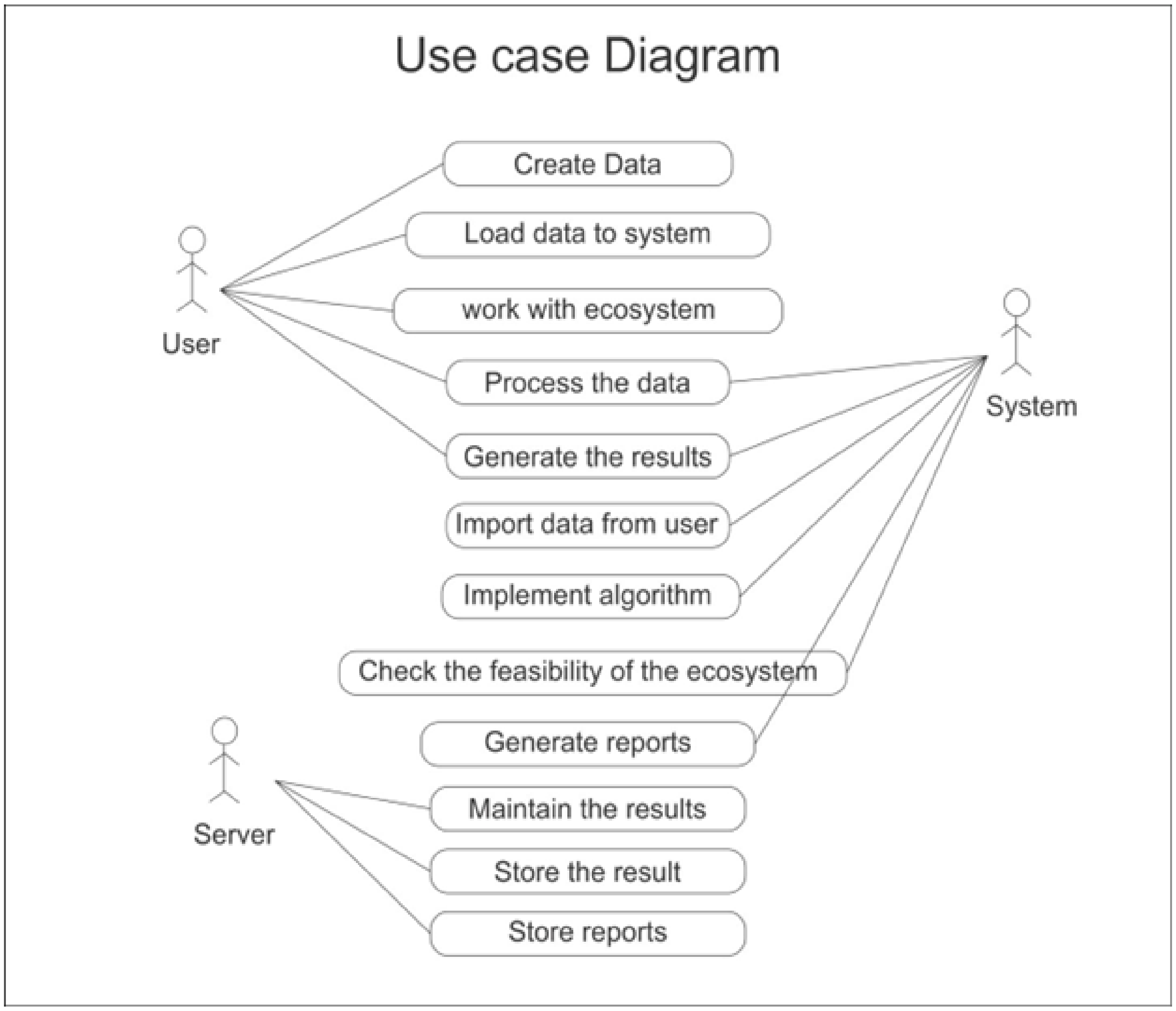


Fig 5.2.1.1 Use Case Diagram

5.2.2 Class Diagram:

It is concerned with the structure's static course of operation.

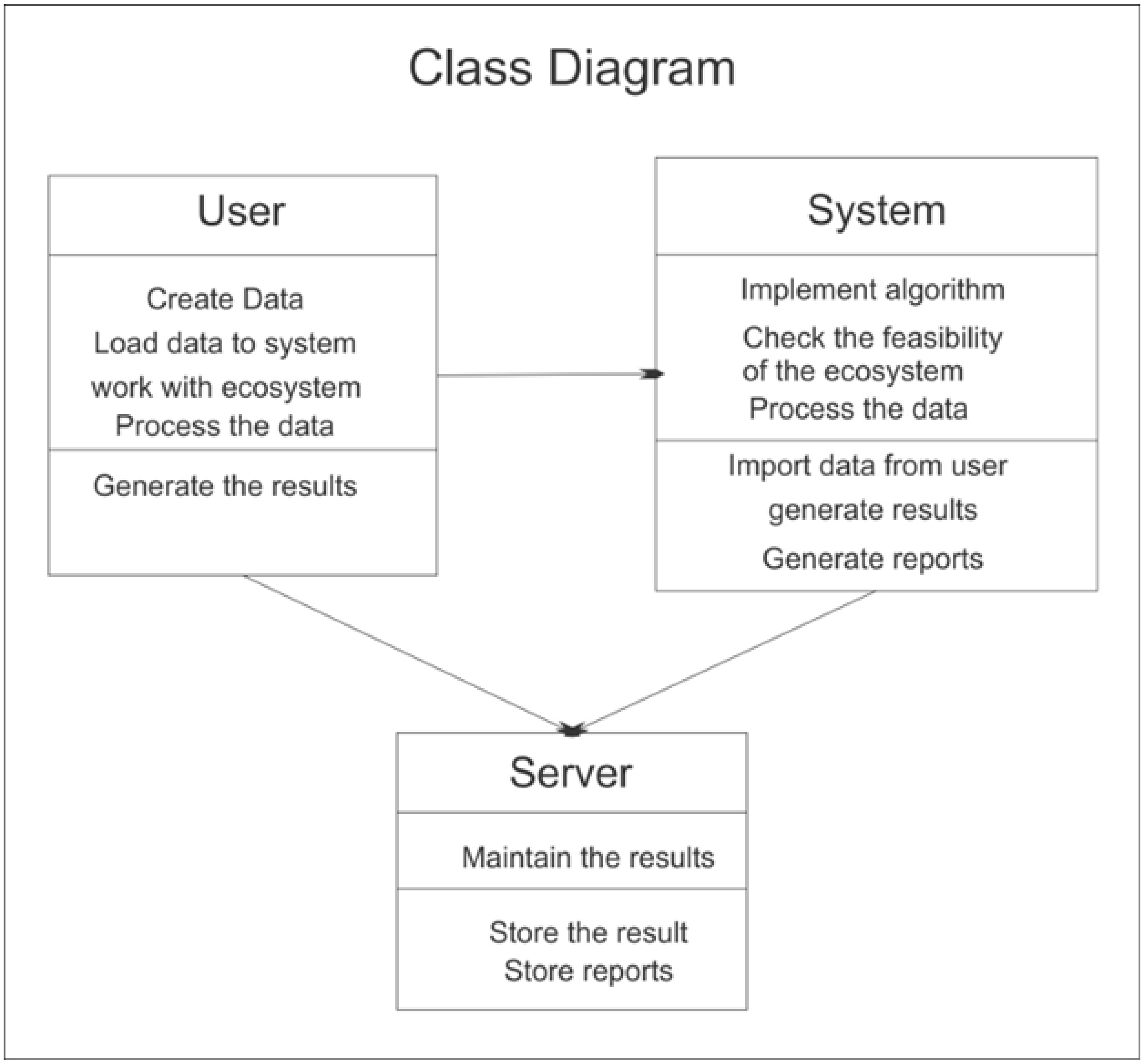


Fig 5.2.2.1 Class Diagram

5.2.3 Sequence Diagram:

This is a cooperative design that focuses on message request time. It contains a collection of parts as well as the messages delivered and received by the instance of parts.

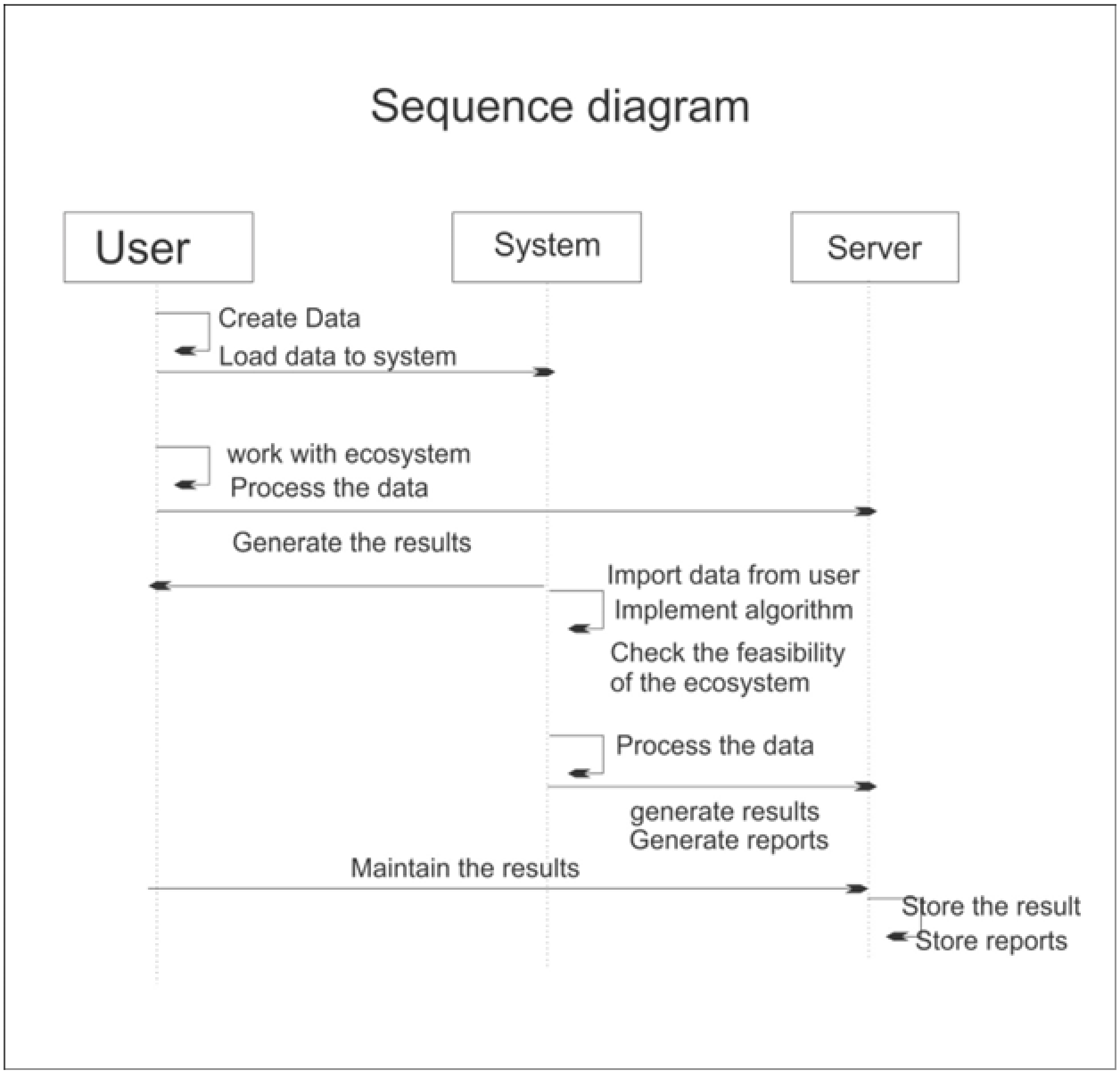


Fig 5.2.3.1: Sequence diagram

5.3 DATA FLOW DIAGRAMS:

An information stream design (DFD) is a graphical representation of a data framework's "stream" of information, displaying its strategy edges. A DFD is frequently used as a preliminary walk to get a general idea of the structure, which may then be clarified afterwards.

5.3.1 DFD Symbols:

In the DFD, there are four symbols

- A square defines a source or destination of system data.

- An arrow identifies data flow. It is the pipeline through which the information flows.

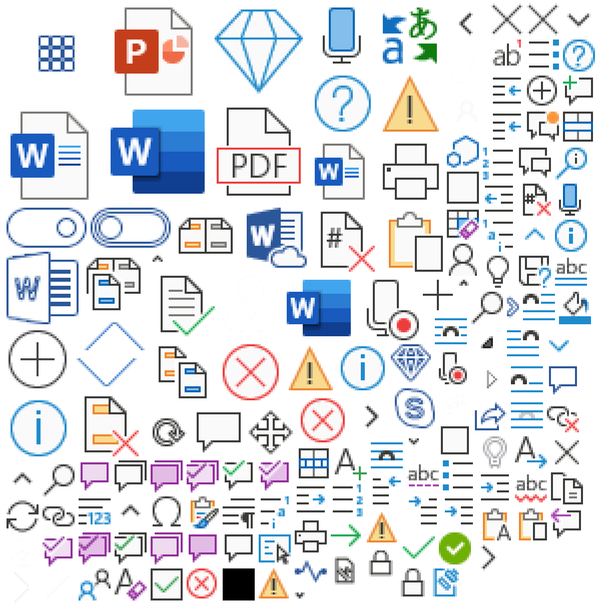


Fig 5.3.1.1 Different Types of Symbols

- A circle represents a process that transforms incoming data flow into outgoing data flow.

- An open rectangle is a data store, data at rest or a temporary repository of data.

5.3.2 Level 0: System input/ output level:

A level 0 DFD specifies the system's broad boundaries, as well as the system's input and output flow, and primary processes.

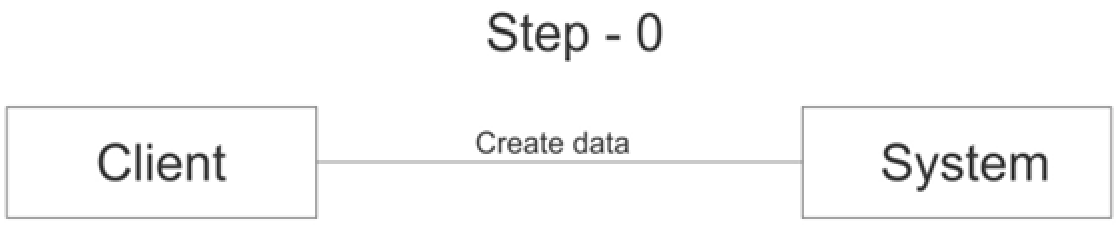


Fig 5.3.2.1 Level 0 DFD

5.3.3 Level 2: File level detail data flow:

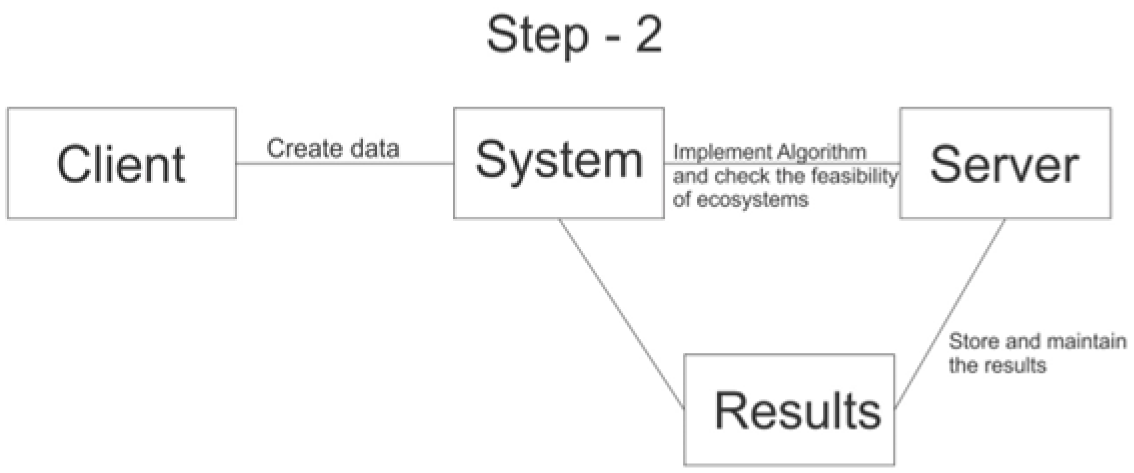

Step - 2
Client
Create data
System
Implement Algorithm and check the feasibility of ecosystems
Server
Store and maintain the results
Results

Fig 5.3.3.1 Level 2 DFD

5.4 MODULES:

5.4.1 Image Acquisition and Data Preprocessing:

Obtaining the picture dataset is critical in the creation of any object detection-based system. When gathering datasets, there are a few things to keep in mind that might aid with prediction and training the convolutional neural network model effectively. In general, a dataset with more data yields a better result. The information gathered from the VALID website is completely unprocessed and unfiltered.

5.4.2 Data preparation and image labeling of objects:

The act of collecting and using domain knowledge to sort and iterate over the photos in a dataset is referred to as data preparation. In this and subsequent sections, the actions involved in processing the raw data are enumerated and discussed in depth. The massive quantity of picture data obtained from the 3D satellite imaging dataset is next subjected to data preparation, which involves converting and cleaning the raw data in order to create a trainable dataset. Each image is opened and labeled according to the things present.

5.4.3 CNN Training and Construction:

The Training phase is a subset of the model's training, and the Test set is a subset of the model's testing. CNN creation is the process of creating a neural network architecture specifically for this application. As a result, we may feed CNN the training data, which should contain the right response, referred to as the target attribute. This research makes use of a unique neural network created specifically for satellite images.

5.4.4 Validation of CNN and Results Analysis

Validation is one of the most critical processes of verifying the model against the input data collection. The test data is used to compare the ground truth with the projected outcome from the CNN's predicted output prediction after the training phase.

CHAPTER 6

6. OVERVIEW OF TECHNOLOGY

6.1 PYTHON:

It is an extremely popular programming language in the software industry. When people decide to learn coding, python is going to be the first priority to choose as a beginner. Python was created by Guido van rossum, a dutch programmer with high technical skills. He loves to do things in a simple manner and that type of his thinking helps a lot of young coders in the world to learn coding by his popular creation "Python Programming language".

Python interpreters are available for every operating system that are present in the industry. C Python, the one of the best things with python is, it is an open source, everyone can use it, upgrade it, and do as they like. There is a big community for python, where all python enthusiasts will be there and try to use the python in an efficient manner.

In python we have Cpython where it can be used as both interpreted and compiled language. this cpython will help a lot of c-background coders who change their coding language from c programming language to python.

6.2 THE POWER PYTHON HOLDS:

Due to the increase of popularity of machine learning and data science in the industry, the python popularity got increased. We can implement these machine learning, data science and many other trending technologies with other programming languages also, but using python ,because of many inbuilt libraries we can do a lot of machine learning ,data science , artificial intelligence projects in a most efficient manner.

There are many jobs available for python developers, where they can earn some good sort of salaries for their career.

6.2.1 Pandas:

A library used for data manipulation as well as data analysis.To keep track of numerical tables and time series, the library contains data structures and functions.

6.2.2 NumPy:

It is another great library that supports enormous, multi--dimensional arrays and matrices, and vast library of high-level mathematical functions to perform on these arrays, which have been added

6.2.3 SciPy:

A library used by scientists, analysts, and engineers doing scientific computing and technical computing. Being a free, cross-platform, general-purpose and high-level programming language, Python has been widely adopted by the scientific community. Scientists value Python for its precise and efficient syntax, relatively flat learning curve and the fact that it integrates well with other languages (e.g. C/C++).

As a result of this popularity there are plenty of Python scientific packages.

CHAPTER 7

7. IMPLEMENTATION

7.1 CODING

```
import numpy as np #importing libraries
import math

n = 100 #items to have in filter - words
length = 2*n #having double sized to deal with collisions
print("(n) Number of items we expect to have in filter:", n)

p = 0.1 #false positive rate, from 0.0 to 1.0
print("(p) False positive rate set as:",p*100,"%")

m = (-n*np.log(p)) / (0.48045) #bits needed in bloom filter
m = int(round(m)) #space of actual data structure that holds the data is O(m)
print("(m) Calculated number of bits needed in bloom filter:",m)
print("\t m = -n*ln(p) / (ln(2)^2)") # memory size as a function of number of items and false positive rate

k = m/n * np.log(2) #hash functions required
k = int(math.ceil(k))
print("(k) Calculated number of hash functions required:", k)
print("\t k = m/n * ln(2)")

--------------------------------------------------------------------------

array = [None for i in range(length)] #intiliazing empty array of expected elements we're adding(twice as n to prevent collisions)
bloom = [0 for i in range(m)] #initlializing the bloom filter of size m, as calculated above

"""
one hash function which can acts as many hash functions as requiredby changing x argument input
"""
def hashing(element,x): #hash function
  if x%2 == 0:
     return int(((element*((x+1)/20))-(x-10))) % m
  else:
        return int(((element*(1+x))-(x*0.9+1))) % m

--------------------------------------------------------------------------

"""
function to insert an element into the array list and updating the bloom filter accordingly
"""
def insert(element, i = 0):
    global array
    global k

    hashedidx = [] #to store list of indexes from multiple hash functions

    for j in range(k): #iterate as many times as we needed hash function, (k)
        hashedidx.append(hashing(element,j)) #get hash functions outputs

     idx = ((sum(hashedidx) - hashedidx[len(hashedidx)-1]) + (hashedidx[len(hashedidx)-1]*i)) % length
#hashing

    if array[idx] == None or array[idx] == 0: #if slot available
        array[idx] = element #insert element
        addBloom(hashedidx) #after element inserted, change values in count bloom filter
        return True

    else: #otherwise recursive by incrementing iteration variable
        return insert(element, i+1)

def addBloom(idx): #update the bloom filter after inserting element
    global bloom
    for i in idx: #for each index from hash functions we increment values in bloom filter
        bloom[i]+=1

--------------------------------------------------------------------------

"""
function to probabilistically check if element in the list**
"""
def check(element): #chekcing if the element is in the list using bloom filter
    global array
    global bloom
    global k

    count = 0 #counter

    hashedidx = [] #to store hash functions ouput

    for i in range(k): #iterate as many times as we needed hash function, (k)
        hashedidx.append(hashing(element,i)) #get hash functions outputs

    for i in hashedidx: #check for each hash function output
        if bloom[i] > 0: #check if the index is greater than 0
            count+=1 #increase counter
    if count == len(hashedidx): #if all the hashed output index are > 0, then
        return True #there is probability that element may be present
    else:
        return False #otherwise absolutely not present

-----------------------------------------------------------------------------

"""
funciton to delete an element from the list and update count bloom filter accordingly
"""
def delete(element, i=0):
    global k
    hashedidx = [] #to store hash funcitons ouput

    for j in range(k): #iterate as many times as we needed hash function, (k)
        hashedidx.append(hashing(element,j)) #get hash functions outputs

     idx = ((sum(hashedidx) - hashedidx[len(hashedidx)-1]) + (hashedidx[len(hashedidx)-1]*i)) % length
#hashing

    if array[idx] == element: #if that slot that has the element
        array[idx] = 0 #delete the element
        delBloom(hashedidx) #update bloom
    elif array[idx] == None: #if the slot is empty
        return print("Element not in the list") #hence, the element isn't in the list
    else:
        return delete(element, i+1) #otherwise, check next possible occurence
```

```
def delBloom(idx): #update the bloom filter
    global bloom
    for i in idx: #for each index from hash functions we decrement values in bloom filter
        bloom[i]-=1

-------------------------------------------------------------------------

"""
Now, we will start running test to check the practically obtained false positive rate (FPR) and compare it with our
initialized set FPR

For our test, we are feeding data of 'n' unique elements and then testing it with another 'n' unique elements
technically, we should get 'n' times "absolutely not in the list" as all the test elements are unique
therefore every positive result we get is actually false postive
"""
import random

testSet = [1000*i for i in range(1,30)] #creating number of elements set
fpSet = [] #to store FPR for each test set elements

for size in testSet:
    n = size #items to have in filter
    length = 2*n
    p = 0.05 #false positive rate, from 0.0 to 1.0 <--- set FPR
    m = (-n*np.log(p)) / (0.48045) #calculations as done above
    m = int(round(m))
    k = m/n * np.log(2)
    k = int(math.ceil(k))
    array = [None for i in range(length)] #resetting
    bloom = [0 for i in range(m)]

    feedData = list(range(1,size))
    testData = list(range(size+1, 2*size+1))

    fp = 0
    for y in feedData:
        insert(y)
    for y in testData:
        if check(y) == True:
            fp+=1

    fpSet.append((fp/size)*100)

----------------------------------------------------------------------------

from matplotlib import pyplot as plt
#plots each graph
print("Using memory size of",m,"bits in counting bloom filter")
plt.plot(testSet,fpSet, label='FPR')
plt.title('Practical Implemention and FPR observations')
plt.xlabel("Unique Elements Fed and Unique Elements Tested")
plt.ylabel("False Positive Rate")
plt.legend()
plt.show()
print("FPR initialized as",p*100,"%")
print("Practical observation average:",sum(fpSet)/len(fpSet),"%")

----------------------------------------------------------------------------

"""
using the same code as above, we will only change the initialized set FPR and compare it with average
observed FPR
"""

fpSet = []

for size in testSet:
    n = size #items to have in filter
    length = 2*n
    p = 0.2 #false positive rate, from 0.0 to 1.0
    m = (-n*np.log(p)) / (0.48045)
    m = int(round(m))
    k = m/n * np.log(2)
    k = int(math.ceil(k))
    array = [None for i in range(length)]
    bloom = [0 for i in range(m)]
    feedData = list(range(1,size))
    testData = list(range(size+1, 2*size+1))

    fp = 0
    for y in feedData:
        insert(y)

    for y in testData:
        if check(y) == True:
            fp+=1

    fpSet.append((fp/size)*100)
```

```
#plots each graph
print("Using memory size of",m,"bits in counting bloom filter")
plt.plot(testSet,fpSet, label='FPR')
plt.title('Practical Implemention and FPR observations')
plt.xlabel("Unique Elements Fed and Unique Elements Tested")
plt.ylabel("False Positive Rate")
plt.legend()
plt.show()
print("FPR initialized as",p*100,"%")
print("Practical observation average:",sum(fpSet)/len(fpSet),"%")
```

```
fpSet = []

for size in testSet:
    n = size #items to have in filter
    length = 2*n
    p = 0.25 #false positive rate, from 0.0 to 1.0
    m = (-n*np.log(p)) / (0.48045)
    m = int(round(m))
    k = m/n * np.log(2)
    k = int(math.ceil(k))
    array = [None for i in range(length)]
    bloom = [0 for i in range(m)]
    feedData = list(range(1,size))
    testData = list(range(size+1, 2*size+1))

    fp = 0
    for y in feedData:
        insert(y)

    for y in testData:
        if check(y) == True:
            fp+=1

    fpSet.append((fp/size)*100)

---------------------------------------------------------------------------

#plots each graph
print("Using memory size of",m,"bits in counting bloom filter")
plt.plot(testSet,fpSet, label='FPR')
plt.title('Practical Implemention and FPR observations')
plt.xlabel("Unique Elements Fed and Unique Elements Tested")
plt.ylabel("False Positive Rate")
plt.legend()
plt.show()
print("FPR initialized as",p*100,"%")
print("Practical observation average:",sum(fpSet)/len(fpSet),"%")
```

7.2 TESTING

The process of executing a programme with the goal of identifying faults is known as testing. Our programme must be free of errors in order to function properly. If successful testing is completed, the programme will be free of any errors. It is the most important quality measure used during programming development. The design of tests for programming and other constructed products may be as difficult as the underlying design of the item itself.

7.2.1 Testing strategies:

A software testing strategy is a set of steps that must be performed in order to verify that the final product is of the highest quality feasible. It's a set of procedures that an in-house or outsourced QA team must follow in order to achieve the quality criteria you've established. If you choose the strategy that your project does not need to be perfect, you will waste time and money.

7.2.1.1 Unit Testing:

Unit testing is a sort of software testing that examines individual units or operations. This project's main goal is to thoroughly test each unit or function. A unit is the base unit of a program that can be tested. It generally just has a single or a few inputs and outputs.

7.2.1.2 Black Box Testing

It is a software evaluation process that involves examining the functioning of software programmes without knowing how they work.Internal code architecture, implementation, or internal routes are all examples of internal code. Black Box Testing is a type of software testing that concentrates on the input and output of software applications and is totally driven by software requirements and specifications. Behavioral testing is another name for it.

7.2.1.3 White Box testing

It evaluates and validates a software system's inner workings, including its infrastructure, code, and links to other systems. White box testing is crucial for automated build operations in today's Continuous Integration and Delivery development pipeline.

7.2.1.4 Integration Testing

It is defined as a methodical approach to product engineering development. While the reconciliation is taking place, do lead tests to uncover any interface-related errors. Its goal is to take unit-tested modules and put together a programme structure based on the suggested blueprint.

7.2.1.5 System Testing

This testing assesses the framework's functioning from the client's perspective using a specified report. It does not need a thorough understanding of the framework, such as the code plan or structure. Framework testing is the first step in true testing, in which you test the entire object rather than just a module or highlight.

7.3 TEST APPROACH:

A test approach is the test strategy for a project that outlines how testing will be done. The test technique employs two strategies:

Proactive - A strategy in which the test design process is started as soon as feasible to detect and correct flaws before the build is built.

Reactive: Testing does not begin until the design and code are finished, which is known as a reactive method.

7.3.1 Bottom up Approach

A bottom-up technique is combining systems to build larger, more complicated systems, with the original systems serving as subsystems of the larger system. Bottom-up processing is a type of information processing in which incoming data from the environment is used to construct a perception.

7.3.2 Top down Approach

This method includes testing directed from the main module to the sub module. If the sub module is not built, it's imitated by a temporary programme called STUB. For this form of testing, upper-level modules are the starting point.Stubs are manufactured since the nitty gritty workouts are often done in the lower level programmes.

7.4 VALIDATION

The approach for determining if programming fulfills business criteria that are declared during the development process or at the end of the development process. The item must pass approval testing to guarantee that it genuinely answers the customer's needs. It might alternatively be described as proving that the thing is fit for its intended use when supplied in excellent shape.

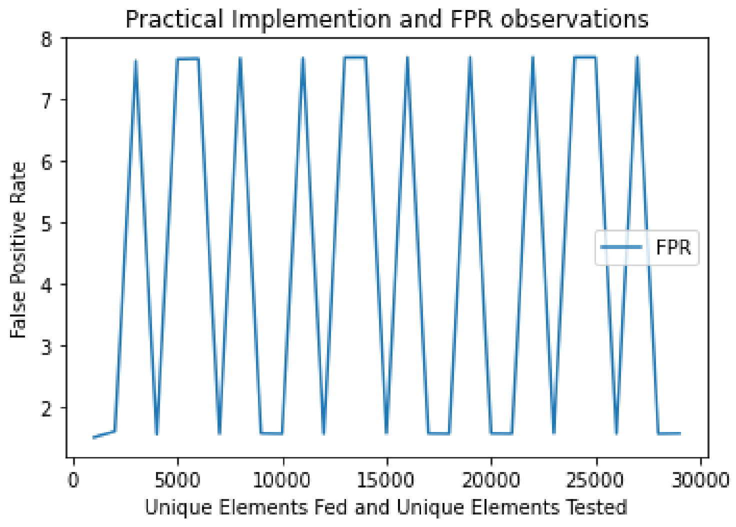
7.5 TEST CASES:

Experiments are a collection of actions, conditions, and data sources that may be used to carry out testing tasks. This activity's main purpose is to see if a product passes or fails in terms of usefulness and other factors.The process of creating tests can also help find problems in an application's requirements or design. The application generates a definite conclusion and quits the framework at a certain end point, also known as the execution post condition, after applying a set of information values.

CHAPTER 8

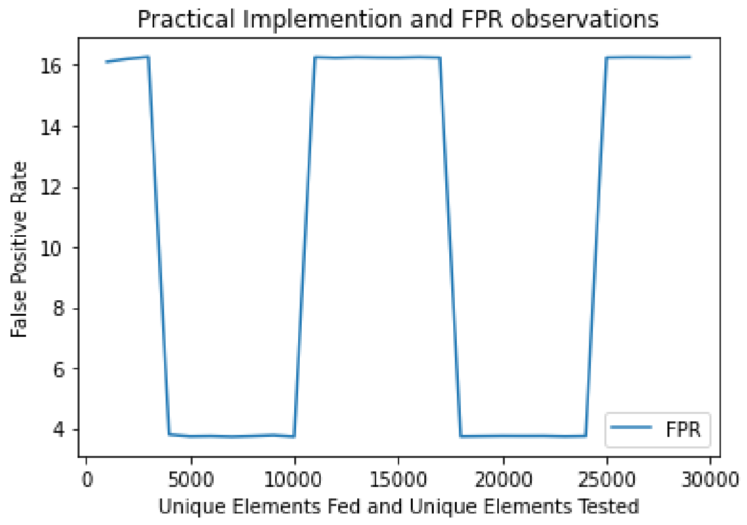
8. RESULTS & DISCUSSION

By constantly changing the False Positive Rate(initializing), we see the number of words that can be transmitted has increased. The words are continuously transmitted without any distortions in the process. The project's major purpose is to decrease the False Positive Rate(fpr). The POC demonstrates it perfectly.



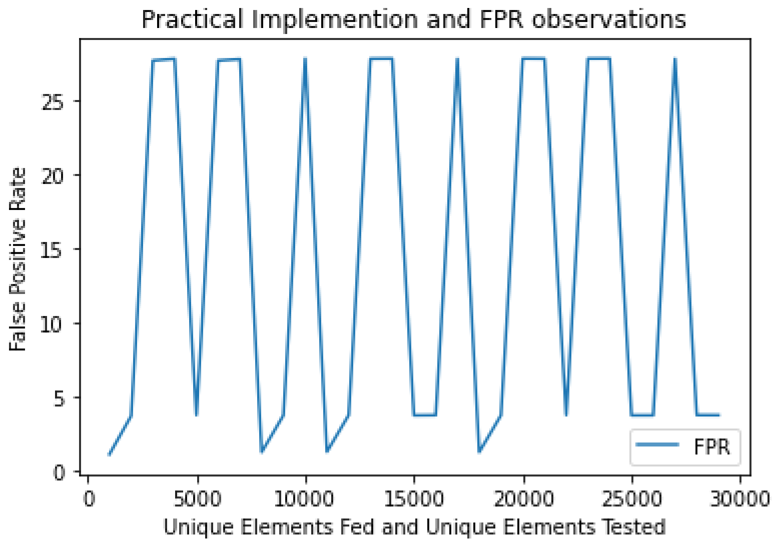
```
FPR initialized as 5.0 %
Practical observation average: 4.302708654133927 %
```

Fig 8.1 Practical Implementation and FPR observations-1



```
FPR initialized as 20.0 %
Practical observation average: 10.20874612312953 %
```

Fig 8.2 Practical Implementation and FPR observations-2



```
FPR initialized as 25.0 %
Practical observation average: 14.196855258603863 %
```

Fig 8.3 Practical Implementation and FPR observations-3

CHAPTER 9

9. CONCLUSION AND FUTURE ENHANCEMENT

We offer ACBF, a multilevel optimization strategy for creating an accurate CBF to minimize the false positive probability. The counter vector is partitioned over numerous layers to create ACBF. We present an optimized ACBF that maximizes the first level size while minimizing false positive likelihood and preserving the same functionality as CBF, where is the number of counters, is the number of items, and is the number of hash functions. To boost the reduce-side join performance, we use ACBFs in MapReduce. In the map phase, ACBF is utilized to filter out duplicate records that have been shuffled. ACBF is built in a distributed manner by combining all map jobs' local hash tables. We show that ACBF is a precise data format suited for large-scale data processing.

10. REFERENCES

A. Journals / Articles:

[1] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Commun. ACM, Vol. 13, no. 7, pp. 422–426, 1970.

[2] A. Singh, S. Garg, K. Kaur, S. Batra, N. Kumar, and K. R. Choo, “Fuzzy-folded bloom filter-as-a-service for big data storage in the cloud,” IEEE Transactions on Industrial Informatics, Vol. 15, no. 4, pp. 2338–2348, 2019.

[3] A. Singh, S. Garg, S. Batra, N. Kumar, and J. J. Rodrigues, “Bloom filter based optimization scheme for massive data handling in the iot environment,” Future Generation Computer Systems, Vol. 82, pp. 440 – 449, 2018.

[4] K. Rabieh, M. M. Mahmoud, K. Akkaya, and S. Tonyali, “Scalable certificate revocation schemes for smart grid ami networks using bloom filters,” IEEE Transactions on Dependable and Secure Computing, Vol. 14, no. 4, pp. 420–432, 2017.

[5] R. Patgiri, S. Nayak, and S. K. Borgohain, “PassDB: A password database with strict privacy protocol using 3d bloom filter,” Information Sciences, Vol. 539, pp. 157–176, 2020.

[6] M. Gomez-Barrero, C. Rathgeb, G. Li, R. Ramachandra, J. Galbally, and C. Busch, “Multi-biometric template protection based on bloom filters,” Information Fusion, Vol. 42, no. Supplement C, pp. 37 – 50, 2018.

[7] S. Nayak and R. Patgiri, “A review on role of bloom filter on dna assembly,” IEEE Access, Vol. 7, pp. 66 939–66 954, 2019.

[8] R. Patgiri, S. Nayak, and S. K. Borgohain, “Preventing ddos using bloom filter: A survey,” EAI Endorsed Transactions on Scalable Information Systems: Online First, Vol. 5, no. 19, pp. 1–9, 11 2018.

[9] R. Patgiri, S. Nayak, and S. K. Borgohain, “Shed more light on bloom filter’s variants,” CoRR, Vol. abs/1903.12525, 2019.

[10] R. Patgiri, S. Nayak, and S. K. Borgohain, “Hunting the Pertinency of Bloom Filter in Computer Networking and Beyond: A Survey,” J. Comput. Networks Commun., Vol. 2019, Feb 2019.

[12] S. Z. Kiss, E. Hosszu, J. Tapolcai, L. Ronyai, and O. Rottenstreich, ´ “Bloom filter with a false positive free zone,” IEEE Transactions on Network and Service Management, pp. 1–1, 2021.

[13] O. Rottenstreich, P. Reviriego, E. Porat, and S. Muthukrishnan, “Avoiding flow size overestimation in the count-min sketch with bloom filter constructions,” IEEE Transactions on Network and Service Management, pp. 1–1, 2021.

[14] P. Reviriego, J. Mart´ınez, D. Larrabeiti, and S. Pontarelli, “Cuckoo filters and bloom filters: Comparison and application to packet classification,” IEEE Transactions on Network and Service Management, Vol. 17, no. 4, pp. 2690–2701, 2020.

[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: A scalable wide-area web cache sharing protocol,” IEEE/ACM Trans. Netw., Vol. 8, no. 3, pp. 281–293, Jun. 2000.

[16] A. Kirsch and M. Mitzenmacher, Less Hashing, Same Performance: Building a Better Bloom Filter. LNCS, Springer, 2006, pp. 456–467.

[17] A. Kirsch and M. Mitzenmacher, “Less hashing, same performance: Building a better Bloom filter,” Random Structures Algorithms, Vol. 33, no. 2, pp. 187–218, Sep 2008.

[18] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” ACM Trans. Comput. Syst., Vol. 26, no. 2, pp. 4:1–4:26, 2008.