Algorithms for Massive Data Problems

A B. Tech Project Report Submitted
in Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Devaishi Tiwari
(170101021)

under the guidance of

Dr. Benny George Kenkireth

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Algorithms for Massive Data Problems**" is a bonafide work of **Devaishi Tiwari** (**Roll No. 170101021**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Benny George Kenkireth**

Assistant Professor,

Department of Computer Science & Engineering,

Indian Institute of Technology Guwahati, Assam.

April, 2021

Guwahati.

Acknowledgements

The topic ”Algorithms for Massive Data Problems” was suggested by my guide Prof. Benny George Kenkireth to pursue for my thesis project. I am very grateful to all our faculties, friends and family who helped me on the way. I would like to thank my supervisor and guide Prof. Benny George Kenkireth for continuously helping and encouraging me.

Abstract

At this day and age, data is growing faster than ever. Around 300 billion emails, 65 billion messages and 350 millions pictures are sent out everyday. According to IDC's study – "The Digital Universe in 2020", we have approximately 40 trillion gigabytes data by the end of the year 2020. This rapid growth in the amount of data has led to existence of a new field called **Big / Massive Data Analysis**

With this massive amount of data, comes a huge opportunity to increase **enhanced decision making, insight discovery**, and **process optimization**.

The objective of this project is to **introduce the field of massive data analysis, communicate its usefulness in different sectors and examine some of the important algorithms** related to massive data. In this report, we are going to discuss various approaches to deal with data that is too large to be stored in the Random Access Memory.

We will also look into various existing algorithms for handling and analysing a massive collection of data and evaluate their performance and correctness theoretically. Furthermore, we will discuss the implementation of some of these efficient algorithms for a specific use case and try to optimize the operations that are to be performed.

Contents

Chapter 1

Introduction

1.1 Big Data

According to Gartner, “Big Data are **high volume, high velocity**, or **high-variety** information assets that require new forms of processing to enable **enhanced decision making, insight discovery**, and **process optimization**.” Big Data have certain characteristics that distinguish it from regular data. These characters are called the **8V’s of Big Data**, namely volume, velocity, variety, value, visualization, validity, veracity and variability. With the huge amount of data being generated every second, it is imperative to adapt and learn how to handle different types of Big Data. In the next subsection, we will discuss how big data analysis is a must for every organisation, industry and research.

1.2 Motivation behind Big Data Analysis

These massive amounts of data, generated on a daily basis, can be effectively analysed for various useful insights. These insights can, in turn, enhance customer experience, reduce costs and increase efficiency. Some of the general benefits of Big Data Analysis in various fields and industries are as follows:

- **Banking and Security**: Finance industry is highly dependent on data analysis for high frequency trading algorithms, stock analysis etc. Big Data can also be used to monitor the banks and trading markets and check for fraudulent practices.

- **Healthcare**: Healthcare industry generates a huge amount of data in the form of diagnostic reports and patients' records. This data can be analysed to detect symptoms of various diseases and help diagnose other patients.

- **Customer Behaviour Analysis**: The e-commerce industry has collected a lot of data over time, using customer loyalty cards, scanners and receipts. This data can then be used to analyze the customer's choices and behaviour and accordingly enhance the customer experience.

1.3 Objective of the Report

The objective of this project is to introduce the field of massive data analysis, communicate its usefulness in different sectors and examine and implement some of the important algorithms related to massive data.

The main focus of the report is to **discuss both theoretical and practical aspects of various deterministic and non-deterministic algorithms that deal with data that is too large to be stored in the Random Access Memory.**

We will also look into various existing algorithms for handling and analysing a massive collection of data and evaluate their performance and correctness theoretically. Furthermore, we will discuss the implementation of some of these efficient algorithms for a specific use case and try to optimize the operations that are to be performed.

1.4 Organization of the Chapters

This chapter provides a basic introduction on massive data analysis and its applications in the real world.

The remaining report is organized into chapters, as follows:

- Chapter 2 introduced three different techniques of computations, namely data streaming, sampling and sketching.
- In Chapter 3, we discuss the process of data streaming and its advantages in detail.
- In Chapter 4, we explore various algorithms for massive data stream problems. We also theoretically analyse the accuracy and failure probability of non-deterministic algorithms and establishe correctness for deterministic-algorithms.
- In Chapter 5, we discuss implementation of the algorithms and the results generated
- Finally, Chapter 6 concludes the entire report and discusses the proposed direction for the project.

Chapter 2

Techniques to Handle Big Data

There are various existing models of computation available to deal with massive data sets. The most efficient methods of computation are based on **data streaming, data sampling** and **data sketching**. Our primary focus will be over Data Streaming Algorithms.

2.1 Data Streaming

Streaming data has turn out to be a central component of data architecture and analysis because of the sizeable increase in data from numerous non-conventional sources such as IoT Sensors, real-time advertising, service logs, telemetry from devices etc.

A data stream is simply continuously generated data. In other words, the data items in a data stream arrive sequentially, instead of being stored inside the memory as a block. We can better apprehend this concept by thinking of data items as packets on a computer network, passing across a router sequentially. There is a wide variety of streaming algorithms, both deterministic and randomized, that are useful for computing various statistics or operations over the data, like summation, average of items, or maximum, minimum, or majority items. We will discuss some of these algorithms in the later chapters.

2.2 Data Sampling

Sampling is generally defined as a rule-based process that selects a smaller set of items from a bigger group. Thus, in the data sampling model, the data is reduced into its much smaller approximation before being stored in the memory. We can think of this as selecting a small set of people to represent a large population for estimating various statistics and summaries over the entire population.

Data sampling is widely used to make standard matrix algorithms like matrix multiplication, single value decomposition, linear regression, etc faster. However, there are certain class of problems, like counting the distinct elements, that cannot be solved using sampling methods. This limitation can be eliminated by using yet another method of computation, called data sketching.

2.3 Data Sketching

Data sketching is essentially creation of a space-efficient report of the data which is useful for answering the given queries. In the data sketching model, we go through the data stream and quickly process it to generate an in-memory summary for the whole collection of data. This compact summary/description, that contains all the relevant information about the data items, is called a **sketch** of the data. Mathematically, a sketch of a data set x, with respect to some method f, is defined as a compressed representation of x which is sufficient for computation of f(x).

One of the important applications of data sketching is storing the context of web pages. Instead of caching millions of web pages, we can use sketching to create and store a brief sketch/description of the web pages in the memory. This sketch can be used to estimate the similarity between two or more web pages, search for relevant web pages and many other functionalities.

Chapter 3

Working with Data Streams

3.1 Data Stream and its Advantages

In instances where we have got end devices which are constantly producing hundreds of thousands of records, forming a data stream, in unstructured or semi-structured form, is highly beneficial.

One of the major advantages of using data stream is **easy scalability**. Modern data stream infrastructure is super-scalable, capable of handling Gigabytes of data per second per processor. Unlike traditional batch algorithms which require pausing/stopping the data stream for processing and drawing conclusion, data stream algorithms performs computation continuously with the arrival of data. As a result, data streaming is more suitable for **high performance real-time analysis**. Data stream processing is also more favorable for **finding out trends/patterns** in the data over time as compared to batch processing.

3.2 Mathematical Representation of Data Streams

Suppose our data contains n data items, $\{a_1, a_2, \dots a_n\}$ each of which has a size of at most b bits. Thus, storing the entire data would take O(n*b) space. On the other hand, data streaming access/stores only one data element (i.e. b bits) at a time and hence requires only

O(b) space. Throughout the report, we will assume the data stream to be of **length n, represented by $\{a_1, a_2, a_3, ...a_n\}$, where a_i can take up any value from 1 to $m = 2^b$.**

Another important terminology is the frequency of an element. Frequency of a symbol i, denoted by f_i, can be defined as the number of times that symbol occurs in the data stream. Summation of the p^{th} power of these frequencies over all the symbols is called the **p$^{\text{th}}$ frequency moment**.

$$F.M.(p) = \sum_{s=1}^{m} f_s^p$$

Note that the frequency moment with p=0, is defined as the number of distinct symbols present in the stream. The first frequency moment will be the sum of frequencies of all the symbols in the stream, which essentially is the length of the stream itself. Second frequency moment, on the other hand, can be associated with the variance of the stream.

Chapter 4

Algorithms over Data Streams

There is a wide variety of data streaming algorithms available for different operations over big data. We will only discuss algorithms that compute some function of a massive data stream like number of distinct elements, most frequent element etc. Our primary goal is efficiently processing on the data stream using only a small amount of space.

Recall that we have assumed a data stream of length **n**, $\{a_1, a_2, a_3, ...a_n\}$, where a_i can take up any value from 1 to m. We will follow this assumption throughout the chapter.

4.1 Distinct Elements' Count

4.1.1 Naive Method

One simple deterministic approach is to create a bit-vector in $\mathbf{O(m)}$ space to record which of the m symbols have occurred in the stream so far. Another method can be to store all the distinct elements in a set, which will take $\mathbf{O(n.log_2(m))}$.

Note that we need at least O(m) space for any deterministic algorithm to find out the count of distinct numbers.

4.1.2 Intuition behind Algorithm

We need to think of a non-deterministic algorithm that can estimate the value within a constant factor of the origin count, using randomization, with a small probability of failure.

Let **D** be the set of distinct elements present in the data stream. Let the minimum element present in D be **Min**. Now the expected value for min will be,

$$\begin{aligned} E(Min) &\approx \frac{m}{2}\ if\ |D| = 1 \\ &\approx \frac{m}{3}\ if\ |D| = 2\ and\ so\ on... \\ &\approx \frac{m}{|D|+1} \end{aligned}$$

Using the above equation, we can approximate the count of distinct elements as follows,

$$|D| \approx \frac{m}{Min} - 1$$

4.1.3 Flajolet-Martin Algorithm

The basic objective of the algorithm is to improve the accuracy of basic non-deterministic distinct count algorithm by using a hash mapping over the set of symbols. Suppose, we use a hash function h where,

$$h : 1, 2, 3, ...m \ \longmapsto\ 0, 1, 2, ...M - 1$$

One example of a suitable hash function is as follows,

$$h_{ab}(x) = (ax + b)\ mod\ M$$

Here M is a prime number smaller than m.

We can store this hash function h_{ab} into memory, by storing the two coefficients a and b, which requires only $\mathbf{O(log_2 M)}$ space. Now, we will store the minimum hash value, instead of the minimum element a_i. Thus, we can approximate the count using,

$$|D| \approx \frac{M}{Min}$$

4.1.4 Correctness:

Our approximation of $|D|$ satisfies $\mathbf{\frac{d}{6} \leq \frac{M}{Min} \leq 6d}$ with a probability greater than $\mathbf{\frac{2}{3} - \frac{d}{M}}$.

Note that the accuracy of the algorithm can be improved by using multiple hash functions and taking the median of all the approximated counts corresponding to different functions. This leads to space consumption of $\mathbf{O(log_{1+\gamma}n)}$, for a small value γ greater than 0, in place of $O(log_2n)$, in exchange of higher probability.

4.2 Majority Element

A majority element is an element that occurs more than n/2 times in the stream. Considering the example of an election, suppose there are **m** candidates and **n** number of votes, coming in as a stream of integers $\{\mathbf{a_1, a_2, a_3, ...a_n}\}$ where a_i can take up any value from 1 to m.

4.2.1 Naive Method

A simple deterministic approach can be to create a vector in $\mathbf{O(m.log_2(n))}$ space to record the number of occurrences of each symbol, in the stream. Another method can be to store all the distinct elements in a set, along with their frequency. This will take $\mathbf{O(n.log_2(n))}$ space.

4.2.2 Moore's Voting Algorithm

We can maintain a single data element along with a counter. We initialise the element and the counter with 0. For each incoming element a_i in the stream, we check if this a_i is equal to the stored element. If both the elements are equal, we increase the counter by one, else we decrease the counter by 1. The decrease counter step can be seen as cancellation of the current item and the saved item. At any point if the counter reaches 0, we replace the saved element with the current element and reset the counter to 1.

The saved value, at the end of the stream, is taken as the majority element.

Algorithm 1 Moore's Voting Algorithm: Majority Element

```
$element \leftarrow 0$
$count \leftarrow 0$
for each $i$ do
    if $i = element$ then
        $count \leftarrow count + 1$
    else
        $count \leftarrow count - 1$
    if $count = 0$ then
        $element \leftarrow i$
        $count \leftarrow 1$
```

4.2.3 Correctness

The deterministic algorithm returns the correct majority element, provided there is a majority element in the stream. In case the stream doesn't have a majority element, the algorithm can return any random element.

Thus, to ensure the correctness of the result, we can check whether the frequency of the returned element is greater than n/2 by going through the stream once more.

4.3 K Most Frequent Elements

4.3.1 Intuition

An updated version of the above majority element algorithm can be used to find out all the elements with frequency above a threshold. More specifically, we approximate the frequency (number of occurrences) of each element to within a constant difference from the exact value, i.e. $\mathbf{\frac{n}{k+1}}$, in $\mathbf{O(k.logn + k.logm)}$ space.

4.3.2 Misra-Gries Algorithm

We can maintain a set of k data elements being counted along with their counters. Initially, the set is empty and the counters are set to 0.

For each incoming element a_i in the stream, we check if this a_i is equal to any of the stored elements. If it is equal to some element in the set, we increase the counter of that element by one. If the element is different from all the elements in the set, we decrement the counters for all the stored elements by 1.

If there are already k elements saved, we decrement the counters for all the stored elements by 1. If the counter for any element reaches 0, we delete that element from the set. The elements that are finally present in the set have frequency above $\frac{n}{k+1}$.

Algorithm 2 Misra-Gries Algorithm: K Most Frequent Elements

```
$setElements \leftarrow \phi$
$count \leftarrow 0$
for $each\ i$ do
    if $a_i \in setElements$ then
        $count_i \leftarrow count_i + 1$
    else if $|setElements| < k + 1$ then
        $setElements \leftarrow setElements \cup \{a_i\}$
        $count_i \leftarrow 1$
    else
        for $all\ j \in setElements$ do
            $count_j \leftarrow count_j - 1$
            if $count_j = 0$ then
                $setElements \leftarrow setElements - \{j\}$
```

4.3.3 Correctness

For each element s in the stream, the frequency calculated by the algorithm, $\bar{f_s}$ is in the range,

$$\bar{f_s} \in \left\{ f_s - \frac{n}{k+1},\ f_s \right\}$$

where f_s is the actual frequency of the element. Note that the frequency of all the elements, which weren't saved in the set, is considered as 0.

Chapter 5

Simulation of Streaming Algorithms

As a part of the thesis project, I have implemented modifications of both Flajolet-Martin algorithm and Misra-Gries algorithm, discussed earlier in the report. I have also run these algorithms on multiple datasets to analyse their computation time and accuracy. The code for the same could be found here. The code has been written in python programming language. I have used **Google Colab Python Notebook** to implement the algorithms, since it provides an interactive environment for data science and analytics and can be used not only as an IDE, but also as a virtual ”notebook”.

5.1 Generating Data Stream

Data stream processing is generally implemented in two sections, namely the storage layer and processing layer. The storage layer is responsible for temporary record storage. The processing layer needs to consume data from storage layer, process and analyse the data, and finally instruct the storage layer to delete the already-processed records. Thus, efficiency, scalability, durability, and fault tolerance is essential in processing data stream. There are various infrastructures available to enable data streaming, some of these are Amazon Kinesis Data Streams, Apache Spark Streaming, and Apache Storm.

Since our major focus is the analysis of the streaming algorithms, I have implemented data stream simply as a text file, which will be read and processed one word at a time. Any of the above-mentioned infrastructures, however, can be used instead to create a data stream.

5.1.1 Twitter Hashtags' Data

Twitter is an online social networking service that allows its users to tweet/post about current events, opinions, feelings and much more. As a result, Twitter produces a massive quantity of data and statistics each day. Many data scientists have been working on creating an infrastructure for real-time analysis of Twitter's data. Hence, I have generated a dataset consisting of the trending hashtags on Twitters using **Twitter API** to effectively demonstrate the results of the algorithms.

Using Flajolet Martin Algorithm, we can approximate the number of new trending hashtags. On the other hand, Misra-Gries algorithm determines which hashtags are the most trending.

5.1.2 Data Extraction using Tweepy

Tweepy is an open source Python package that offers handy access to the Twitter API with Python. Twitter API, in turn, allows us to effectively read and write Twitter data.

In the project, I created a Twitter application to generate the API credentials. These credentials are then used for connecting to Twitter API and running queries to obtain Twitter data. I have preprocessed the returned data and extracted only the hashtags from the tweets. These hashtags were collected in a file, which was used as a data stream.

5.2 Flajolet-Martin Algorithm : Distinct Count

The Flajolet-Martin Algorithm is a non-deterministic algorithm that can estimate the value within a constant factor of the original count of distinct elements. We know that the basic objective of this algorithm is to improve the accuracy of distinct by using a hash mapping over the set of symbols.

I have used multiple hash functions over the dataset, found the approximated distinct count in each case and took the median of the results to increase the accuracy of the algorithm.

5.2.1 Implementation

The algorithm utilizes 4 different types of hash functions. I hashed each element in the data stream with respect to every hash function. Next, I converted all the hashed values into their binary/bit representations and compute the number of trailing zeroes present. These count of trailing zeroes in the hash value were then stored into the corresponding hash files.

These files were then used to calculate the maximum number of trailing zeroes, corresponding to each hash function, say M, then the distinct count is approximated to 2^M for each of the hash function. The final result was however the median of the four distinct results.

A proof for correctness for this implementation of FM algorithm is mentioned in the original paper listed in the reference section.

5.2.2 Result

In the Twitter hashtags dataset, there are **80** distinct trending hashtags, calculated by the naive deterministic method. Our implementation of algorithm returns the distinct count as **64**, which substantially varies from the original count. But since, in actual applications, the input streams have exponentially more volume and velocity. This result might not be an accurate depiction of the algorithm's performance.

Thus, to better analyse the performance of the algorithm, I used another dataset consisting of all the words used in Shakespeare's plays. I used the same algorithm over this dataset to estimate the number of distinct words written in the plays. The actual count of distinct words in the plays came to be **43,281** words. The FM Algorithm, on the other hand, returns the distinct count of **49,152** words. In this case, the approximate value is comparatively closer to the accurate value.

Note that the time taken by the modified FM Algorithm to process the Shakespeare's dataset, is around 1-2 seconds, while the deterministic algorithm takes much larger amount of time.

5.3 Misra-Gries Algorithm: K Most Frequent

Another important data streaming algorithm is Misra-Gries. It is a deterministic algorithm which estimates the frequency for each element s in the stream in the range, $\{f_s - \frac{n}{k+1}, f_s\}$, where f_s is the actual frequency of the element in the stream.

I have slightly changed the algorithm to increase the accuracy of the algorithm, at the cost of space consumed. I doubled the size of the dictionary that stores the frequent elements and their updated counts. Rest of the implementation of the algorithm is similar to the pseudo code provided in the previous chapter.

5.3.1 Result

The algorithm provided the following results,

Ten Most Frequent Elements				
Actual Elements			Misra-Gries Elements	
1	POSTPONE_SSC_CHSL	98	POSTPONE_SSC_CHSL	91
2	TXT_TheDoomsNight	36	TXT_TheDoomsNight	29
3	txt	36	txt	29
4	(Thai) LetsGoTo3Next	30	(Thai) LetsGoTo3Next	23
5	ASAPlivenalive	16	ASAPlivenalive	9
6	GulfKanawut	11	GulfKanawut	4
7	100DaysLockey	10	100DaysLockey	3
8	(Korean) SecretNumber	10	(Korean) SecretNumber	3
9	MadhanyaOutNow	6	Madhanya	1
10	Madhanya	5	Viral	1

It can be observed that the result of the Misra-Gries Algorithm is very close to the original sequence or ranking of most frequent elements. The differences can only be seen in the last few words, which can be due to the closeness in frequency of these words.

Chapter 6

Conclusion

6.1 Conclusion

The aim of this project was to **introduce the field of massive data analysis, understand its usefulness in different sectors and examine some of the important algorithms** related to massive data.

In this report, we introduced three different modes of computations, namely data streaming, sampling and sketching. We discussed in detail, the process of data streaming, and explored various algorithms for massive data stream problems. We also theoretically analysed the accuracy and failure probability of non-deterministic algorithms and established correctness for deterministic-algorithms. Additionally, we implemented these algorithms to re-establish our bounds and conclusion.

The two algorithms we focused on were the FM Algorithm and the Misra-Gries Algorithm. In case of FM Algorithm, we observed that the results are more accurate for larger-sized datasets, but the result within a constant factor of the original count for even for small-sized datasets. Another thing to note is that the accuracy of the algorithm is significantly improved by using multiple hash functions and taking the median of all the approximated counts corresponding to different functions. But this modification leads to more space consumption and hence is a space-accuracy tradeoff.

We also discussed Mira-Gries algorithm in detail. We observed that the algorithm provides close to accurate results. The algorithm may however provide inaccurate results if frequencies of different elements is very close/identical. The returned frequencies of elements are in a close range of their original frequencies. This range gradually decreases if the required number of frequent elements increase. Thus our modification of taking twice the number of frequent elements decreases the error range to almost half of the previous values, hence increasing the accuracy of the result.

6.2 Proposed Direction

In the limited time frame of the thesis project, a basic version of the algorithm has been implemented. There are various directions that can be pursued to increase the efficiency and accuracy of these algorithms. A certain degree of **parallelism** can be introduced by processing records in small-sized batches, instead of processing them one by one. The algorithm can also be extended to an framework, involving **data stream ingestion, processing and data visualization** etc. In future, these points can be taken into account.

References

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020.

[Cor16] Graham Cormode. *Misra-Gries Summaries*, pages 1334–1337. 01 2016.

[Cor17] Graham Cormode. *What is Data Sketching, and Why Should I Care?* 09 2017.

[IZ13] E. Ikonomovska and Mariano Zelke. Algorithmic techniques for processing data streams. In *Data Exchange, Information, and Streams*, 2013.

[KNW10] Daniel Kane, Jelani Nelson, and David Woodruff. An optimal algorithm for the distinct elements problem. 01 2010.

[LÖ09] LING LIU and M. TAMER ÖZSU, editors. *Flajolet-Martin Algorithm*, pages 1141–1141. Springer US, Boston, MA, 2009.

[YYT17] B. Yadranjiaghdam, S. Yasrobi, and N. Tabrizi. Developing a real-time data analytics framework for twitter streaming data. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 329–336, 2017.

final

by Devaishi TIWARI

Submission date: 18-Apr-2021 05:51PM (UTC+0530)
Submission ID: 1562327450
File name: BTP_Phase1_1.pdf (241.41K)
Word count: 4709
Character count: 24435

Algorithms for Massive Data Problems

A B. Tech Project Report Submitted
in Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Devaishi Tiwari
(170101021)

under the guidance of

Dr. Benny George Kenkireth

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

This is to certify that the work contained in this thesis entitled ***"Algorithms for Massive Data Problems"*** *is a bonafide work of* ***Devaishi Tiwari*** *(**Roll No. 170101021**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Benny George Kenkireth**

Assistant Professor,

April, 2021 Department of Computer Science & Engineering,

Guwahati. Indian Institute of Technology Guwahati, Assam.

Acknowledgements

The topic "Algorithms for Massive Data Problems" was suggested by my guide Prof. Benny George Kenkireth to pursue for my thesis project. I am very grateful to all our faculties, friends and family who helped me on the way. I would like to thank my supervisor and guide Prof. Benny George Kenkireth for continuously helping and encouraging me.

Abstract

At this day and age, data is growing faster than ever. Around 300 billion emails, 65 billion messages and 350 millions pictures are sent out everyday. According to IDC's study – "The Digital Universe in 2020", we have approximately 40 trillion gigabytes data by the end of the year 2020. This rapid growth in the amount of data has led to existence of a new field called **Big / Massive Data Analysis**

With this massive amount of data, comes a huge opportunity to increase **enhanced decision making, insight discovery**, and **process optimization**.

The objective of this project is to **introduce the field of massive data analysis, communicate its usefulness in different sectors and examine some of the important algorithms** related to massive data. In this report, we are going to discuss various approaches to deal with data that is too large to be stored in the Random Access Memory.

We will also look into various existing algorithms for handling and analysing a massive collection of data and evaluate their performance and correctness theoretically. Furthermore, we will discuss the implementation of some of these efficient algorithms for a specific use case and try to optimize the operations that are to be performed.

Contents

Chapter 1

Introduction

1.1 Big Data

According to Gartner, “Big Data are **high volume, high velocity**, or **high-variety** information assets that require new forms of processing to enable **enhanced decision making, insight discovery**, and **process optimization**.” Big Data have certain characteristics that distinguish it from regular data. These characters are called the **8V’s of Big Data**, namely volume, velocity, variety, value, visualization, validity, veracity and variability. With the huge amount of data being generated every second, it is imperative to adapt and learn how to handle different types of Big Data. In the next subsection, we will discuss how big data analysis is a must for every organisation, industry and research.

1.2 Motivation behind Big Data Analysis

These massive amounts of data, generated on a daily basis, can be effectively analysed for various useful insights. These insights can, in turn, enhance customer experience, reduce costs and increase efficiency. Some of the general benefits of Big Data Analysis in various fields and industries are as follows:

- **Banking and Security**: Finance industry is highly dependent on data analysis for high frequency trading algorithms, stock analysis etc. Big Data can also be used to monitor the banks and trading markets and check for fraudulent practices.

- **Healthcare**: Healthcare industry generates a huge amount of data in the form of diagnostic reports and patients' records. This data can be analysed to detect symptoms of various diseases and help diagnose other patients.

- **Customer Behaviour Analysis**: The e-commerce industry has collected a lot of data over time, using customer loyalty cards, scanners and receipts. This data can then be used to analyze the customer's choices and behaviour and accordingly enhance the customer experience.

1.3 Objective of the Report

The objective of this project is to introduce the field of massive data analysis, communicate its usefulness in different sectors and examine and implement some of the important algorithms related to massive data.

The main focus of the report is to **discuss both theoretical and practical aspects of various deterministic and non-deterministic algorithms that deal with data that is too large to be stored in the Random Access Memory.**

We will also look into various existing algorithms for handling and analysing a massive collection of data and evaluate their performance and correctness theoretically. Furthermore, we will discuss the implementation of some of these efficient algorithms for a specific use case and try to optimize the operations that are to be performed.

1.4 Organization of the Chapters

This chapter provides a basic introduction on massive data analysis and its applications in the real world.

The remaining report is organized into chapters, as follows:

- Chapter 2 introduced three different techniques of computations, namely data streaming, sampling and sketching.
- In Chapter 3, we discuss the process of data streaming and its advantages in detail.
- In Chapter 4, we explore various algorithms for massive data stream problems. We also theoretically analyse the accuracy and failure probability of non-deterministic algorithms and establishe correctness for deterministic-algorithms.
- In Chapter 5, we discuss implementation of the algorithms and the results generated
- Finally, Chapter 6 concludes the entire report and discusses the proposed direction for the project.

Chapter 2

Techniques to Handle Big Data

There are various existing models of computation available to deal with massive data sets. The most efficient methods of computation are based on **data streaming, data sampling** and **data sketching**. Our primary focus will be over Data Streaming Algorithms.

2.1 Data Streaming

Streaming data has turn out to be a central component of data architecture and analysis because of the sizeable increase in data from numerous non-conventional sources such as IoT Sensors, real-time advertising, service logs, telemetry from devices etc.

A data stream is simply continuously generated data. In other words, the data items in a data stream arrive sequentially, instead of being stored inside the memory as a block. We can better apprehend this concept by thinking of data items as packets on a computer network, passing across a router sequentially. There is a wide variety of streaming algorithms, both deterministic and randomized, that are useful for computing various statistics or operations over the data, like summation, average of items, or maximum, minimum, or majority items. We will discuss some of these algorithms in the later chapters.

2.2 Data Sampling

Sampling is generally defined as a rule-based process that selects a smaller set of items from a bigger group. Thus, in the data sampling model, the data is reduced into its much smaller approximation before being stored in the memory. We can think of this as selecting a small set of people to represent a large population for estimating various statistics and summaries over the entire population.

Data sampling is widely used to make standard matrix algorithms like matrix multiplication, single value decomposition, linear regression, etc faster. However, there are certain class of problems, like counting the distinct elements, that cannot be solved using sampling methods. This limitation can be eliminated by using yet another method of computation, called data sketching.

2.3 Data Sketching

Data sketching is essentially creation of a space-efficient report of the data which is useful for answering the given queries. In the data sketching model, we go through the data stream and quickly process it to generate an in-memory summary for the whole collection of data. This compact summary/description, that contains all the relevant information about the data items, is called a **sketch** of the data. Mathematically, a sketch of a data set x, with respect to some method f, is defined as a compressed representation of x which is sufficient for computation of f(x).

One of the important applications of data sketching is storing the context of web pages. Instead of caching millions of web pages, we can use sketching to create and store a brief sketch/description of the web pages in the memory. This sketch can be used to estimate the similarity between two or more web pages, search for relevant web pages and many other functionalities.

Chapter 3

Working with Data Streams

3.1 Data Stream and its Advantages

In instances where we have got end devices which are constantly producing hundreds of thousands of records, forming a data stream, in unstructured or semi-structured form, is highly beneficial.

One of the major advantages of using data stream is **easy scalability**. Modern data stream infrastructure is super-scalable, capable of handling Gigabytes of data per second per processor. Unlike traditional batch algorithms which require pausing/stopping the data stream for processing and drawing conclusion, data stream algorithms performs computation continuously with the arrival of data. As a result, data streaming is more suitable for **high performance real-time analysis**. Data stream processing is also more favorable for **finding out trends/patterns** in the data over time as compared to batch processing.

3.2 Mathematical Representation of Data Streams

Suppose our data contains n data items, $\{a_1, a_2, \ldots a_n\}$ each of which has a size of at most b bits. Thus, storing the entire data would take O(n*b) space. On the other hand, data streaming access/stores only one data element (i.e. b bits) at a time and hence requires only

O(b) space. Throughout the report, we will assume the data stream to be of **length n, represented by** $\{a_1, a_2, a_3, ...a_n\}$**, where** a_i **can take up any value from 1 to** $m = 2^b$**.**

Another important terminology is the frequency of an element. Frequency of a symbol i, denoted by f_i, can be defined as the number of times that symbol occurs in the data stream. Summation of the p^{th} power of these frequencies over all the symbols is called the **p$^{\text{th}}$ frequency moment**.

$$F.M.(p) = \sum_{s=1}^{m} f_s^p$$

Note that the frequency moment with p=0, is defined as the number of distinct symbols present in the stream. The first frequency moment will be the sum of frequencies of all the symbols in the stream, which essentially is the length of the stream itself. Second frequency moment, on the other hand, can be associated with the variance of the stream.

Chapter 4

Algorithms over Data Streams

There is a wide variety of data streaming algorithms available for different operations over big data. We will only discuss algorithms that compute some function of a massive data stream like number of distinct elements, most frequent element etc. Our primary goal is efficiently processing on the data stream using only a small amount of space.

Recall that we have assumed a data stream of length **n**, $\{a_1, a_2, a_3, ...a_n\}$, where a_i can take up any value from 1 to m. We will follow this assumption throughout the chapter.

4.1 Distinct Elements' Count

4.1.1 Naive Method

One simple deterministic approach is to create a bit-vector in $\mathbf{O(m)}$ space to record which of the m symbols have occurred in the stream so far. Another method can be to store all the distinct elements in a set, which will take $\mathbf{O(n.log_2(m))}$.

Note that we need at least O(m) space for any deterministic algorithm to find out the count of distinct numbers.

4.1.2 Intuition behind Algorithm

We need to think of a non-deterministic algorithm that can estimate the value within a constant factor of the origin count, using randomization, with a small probability of failure.

Let **D** be the set of distinct elements present in the data stream. Let the minimum element present in D be **Min**. Now the expected value for min will be,

$$\begin{aligned} E(Min) &\approx \frac{m}{2} \; if\, |D| = 1 \\ &\approx \frac{m}{3} \; if\, |D| = 2 \; and\; so\; on... \\ &\approx \frac{m}{|D|+1} \end{aligned}$$

Using the above equation, we can approximate the count of distinct elements as follows,

$$|D| \approx \frac{m}{Min} - 1$$

4.1.3 Flajolet-Martin Algorithm

The basic objective of the algorithm is to improve the accuracy of basic non-deterministic distinct count algorithm by using a hash mapping over the set of symbols. Suppose, we use a hash function h where,

$$h : 1, 2, 3, ...m \;\longmapsto\; 0, 1, 2, ...M - 1$$

One example of a suitable hash function is as follows,

$$h_{ab}(x) = (ax + b)\; mod\, M$$

Here M is a prime number smaller than m.

We can store this hash function h_{ab} into memory, by storing the two coefficients a and b, which requires only $\mathbf{O(log_2 M)}$ space. Now, we will store the minimum hash value, instead of the minimum element a_i. Thus, we can approximate the count using,

$$|D| \approx \frac{M}{Min}$$

4.1.4 Correctness:

Our approximation of $|D|$ satisfies $\mathbf{\frac{d}{6} \leq \frac{M}{Min} \leq 6d}$ with a probability greater than $\mathbf{\frac{2}{3} - \frac{d}{M}}$.

Note that the accuracy of the algorithm can be improved by using multiple hash functions and taking the median of all the approximated counts corresponding to different functions. This leads to space consumption of $\mathbf{O(log_{1+\gamma}n)}$, for a small value γ greater than 0, in place of $O(log_2 n)$, in exchange of higher probability.

4.2 Majority Element

A majority element is an element that occurs more than n/2 times in the stream. Considering the example of an election, suppose there are **m** candidates and **n** number of votes, coming in as a stream of integers $\{\mathbf{a_1, a_2, a_3, ...a_n}\}$ where a_i can take up any value from 1 to m.

4.2.1 Naive Method

A simple deterministic approach can be to create a vector in $\mathbf{O(m.log_2(n))}$ space to record the number of occurrences of each symbol, in the stream. Another method can be to store all the distinct elements in a set, along with their frequency. This will take $\mathbf{O(n.log_2(n))}$ space.

4.2.2 Moore's Voting Algorithm

We can maintain a single data element along with a counter. We initialise the element and the counter with 0. For each incoming element a_i in the stream, we check if this a_i is equal to the stored element. If both the elements are equal, we increase the counter by one, else we decrease the counter by 1. The decrease counter step can be seen as cancellation of the current item and the saved item. At any point if the counter reaches 0, we replace the saved element with the current element and reset the counter to 1.

The saved value, at the end of the stream, is taken as the majority element.

Algorithm 1 Moore's Voting Algorithm: Majority Element

```
$element \leftarrow 0$
$count \leftarrow 0$
for $each\ i$ do
    if $i = element$ then
        $count \leftarrow count + 1$
    else
        $count \leftarrow count - 1$
    if $count = 0$ then
        $element \leftarrow i$
        $count \leftarrow 1$
```

4.2.3 Correctness

The deterministic algorithm returns the correct majority element, provided there is a majority element in the stream. In case the stream doesn't have a majority element, the algorithm can return any random element.

Thus, to ensure the correctness of the result, we can check whether the frequency of the returned element is greater than n/2 by going through the stream once more.

4.3 K Most Frequent Elements

4.3.1 Intuition

An updated version of the above majority element algorithm can be used to find out all the elements with frequency above a threshold. More specifically, we approximate the frequency (number of occurrences) of each element to within a constant difference from the exact value, i.e. $\frac{\mathbf{n}}{\mathbf{k+1}}$, in $\mathbf{O}(\mathbf{k}.\mathbf{logn} + \mathbf{k}.\mathbf{logm})$ space.

4.3.2 Misra-Gries Algorithm

We can maintain a set of k data elements being counted along with their counters. Initially, the set is empty and the counters are set to 0.

For each incoming element a_i in the stream, we check if this a_i is equal to any of the stored elements. If it is equal to some element in the set, we increase the counter of that element by one. If the element is different from all the elements in the set, we decrement the counters for all the stored elements by 1.

If there are already k elements saved, we decrement the counters for all the stored elements by 1. If the counter for any element reaches 0, we delete that element from the set. The elements that are finally present in the set have frequency above $\frac{n}{k+1}$.

Algorithm 2 Misra-Gries Algorithm: K Most Frequent Elements

```
$setElements \leftarrow \phi$
$count \leftarrow 0$
for $each\ i$ do
    if $a_i \in setElements$ then
        $count_i \leftarrow count_i + 1$
    else if $|setElements| < k + 1$ then
        $setElements \leftarrow setElements \cup \{a_i\}$
        $count_i \leftarrow 1$
    else
        for $all\ j \in setElements$ do
            $count_j \leftarrow count_j - 1$
            if $count_j = 0$ then
                $setElements \leftarrow setElements - \{j\}$
```

4.3.3 Correctness

For each element s in the stream, the frequency calculated by the algorithm, $\bar{f_s}$ is in the range,

$$\bar{f_s} \in \left\{ f_s - \frac{n}{k+1},\ f_s \right\}$$

where f_s is the actual frequency of the element. Note that the frequency of all the elements, which weren't saved in the set, is considered as 0.

Chapter 5

Simulation of Streaming Algorithms

As a part of the thesis project, I have implemented modifications of both Flajolet-Martin algorithm and Misra-Gries algorithm, discussed earlier in the report. I have also run these algorithms on multiple datasets to analyse their computation time and accuracy. The code for the same could be found here. The code has been written in python programming language. I have used **Google Colab Python Notebook** to implement the algorithms, since it provides an interactive environment for data science and analytics and can be used not only as an IDE, but also as a virtual ”notebook”.

5.1 Generating Data Stream

Data stream processing is generally implemented in two sections, namely the storage layer and processing layer. The storage layer is responsible for temporary record storage. The processing layer needs to consume data from storage layer, process and analyse the data, and finally instruct the storage layer to delete the already-processed records. Thus, efficiency, scalability, durability, and fault tolerance is essential in processing data stream. There are various infrastructures available to enable data streaming, some of these are Amazon Kinesis Data Streams, Apache Spark Streaming, and Apache Storm.

Since our major focus is the analysis of the streaming algorithms, I have implemented data stream simply as a text file, which will be read and processed one word at a time. Any of the above-mentioned infrastructures, however, can be used instead to create a data stream.

5.1.1 Twitter Hashtags' Data

Twitter is an online social networking service that allows its users to tweet/post about current events, opinions, feelings and much more. As a result, Twitter produces a massive quantity of data and statistics each day. Many data scientists have been working on creating an infrastructure for real-time analysis of Twitter's data. Hence, I have generated a dataset consisting of the trending hashtags on Twitters using **Twitter API** to effectively demonstrate the results of the algorithms.

Using Flajolet Martin Algorithm, we can approximate the number of new trending hashtags. On the other hand, Misra-Gries algorithm determines which hashtags are the most trending.

5.1.2 Data Extraction using Tweepy

Tweepy is an open source Python package that offers handy access to the Twitter API with Python. Twitter API, in turn, allows us to effectively read and write Twitter data.

In the project, I created a Twitter application to generate the API credentials. These credentials are then used for connecting to Twitter API and running queries to obtain Twitter data. I have preprocessed the returned data and extracted only the hashtags from the tweets. These hashtags were collected in a file, which was used as a data stream.

5.2 Flajolet-Martin Algorithm : Distinct Count

The Flajolet-Martin Algorithm is a non-deterministic algorithm that can estimate the value within a constant factor of the original count of distinct elements. We know that the basic objective of this algorithm is to improve the accuracy of distinct by using a hash mapping over the set of symbols.

I have used multiple hash functions over the dataset, found the approximated distinct count in each case and took the median of the results to increase the accuracy of the algorithm.

5.2.1 Implementation

The algorithm utilizes 4 different types of hash functions. I hashed each element in the data stream with respect to every hash function. Next, I converted all the hashed values into their binary/bit representations and compute the number of trailing zeroes present. These count of trailing zeroes in the hash value were then stored into the corresponding hash files.

These files were then used to calculate the maximum number of trailing zeroes, corresponding to each hash function, say M, then the distinct count is approximated to 2^M for each of the hash function. The final result was however the median of the four distinct results.

A proof for correctness for this implementation of FM algorithm is mentioned in the original paper listed in the reference section.

5.2.2 Result

In the Twitter hashtags dataset, there are **80** distinct trending hashtags, calculated by the naive deterministic method. Our implementation of algorithm returns the distinct count as **64**, which substantially varies from the original count. But since, in actual applications, the input streams have exponentially more volume and velocity. This result might not be an accurate depiction of the algorithm's performance.

Thus, to better analyse the performance of the algorithm, I used another dataset consisting of all the words used in Shakespeare's plays. I used the same algorithm over this dataset to estimate the number of distinct words written in the plays. The actual count of distinct words in the plays came to be **43,281** words. The FM Algorithm, on the other hand, returns the distinct count of **49,152** words. In this case, the approximate value is comparatively closer to the accurate value.

Note that the time taken by the modified FM Algorithm to process the Shakespeare's dataset, is around 1-2 seconds, while the deterministic algorithm takes much larger amount of time.

5.3 Misra-Gries Algorithm: K Most Frequent

Another important data streaming algorithm is Misra-Gries. It is a deterministic algorithm which estimates the frequency for each element s in the stream in the range, $\{f_s - \frac{n}{k+1}, f_s\}$, where f_s is the actual frequency of the element in the stream.

I have slightly changed the algorithm to increase the accuracy of the algorithm, at the cost of space consumed. I doubled the size of the dictionary that stores the frequent elements and their updated counts. Rest of the implementation of the algorithm is similar to the pseudo code provided in the previous chapter.

5.3.1 Result

The algorithm provided the following results,

Ten Most Frequent Elements				
	Actual Elements		Misra-Gries Elements	
1	POSTPONE_SSC_CHSL	98	POSTPONE_SSC_CHSL	91
2	TXT_TheDoomsNight	36	TXT_TheDoomsNight	29
3	txt	36	txt	29
4	(Thai) LetsGoTo3Next	30	(Thai) LetsGoTo3Next	23
5	ASAPlivenalive	16	ASAPlivenalive	9
6	GulfKanawut	11	GulfKanawut	4
7	100DaysLockey	10	100DaysLockey	3
8	(Korean) SecretNumber	10	(Korean) SecretNumber	3
9	MadhanyaOutNow	6	Madhanya	1
10	Madhanya	5	Viral	1

It can be observed that the result of the Misra-Gries Algorithm is very close to the original sequence or ranking of most frequent elements. The differences can only be seen in the last few words, which can be due to the closeness in frequency of these words.

Chapter 6

Conclusion

6.1 Conclusion

The aim of this project was to **introduce the field of massive data analysis, understand its usefulness in different sectors and examine some of the important algorithms** related to massive data.

In this report, we introduced three different modes of computations, namely data streaming, sampling and sketching. We discussed in detail, the process of data streaming, and explored various algorithms for massive data stream problems. We also theoretically analysed the accuracy and failure probability of non-deterministic algorithms and established correctness for deterministic-algorithms. Additionally, we implemented these algorithms to re-establish our bounds and conclusion.

The two algorithms we focused on were the FM Algorithm and the Misra-Gries Algorithm. In case of FM Algorithm, we observed that the results are more accurate for larger-sized datasets, but the result within a constant factor of the original count for even for small-sized datasets. Another thing to note is that the accuracy of the algorithm is significantly improved by using multiple hash functions and taking the median of all the approximated counts corresponding to different functions. But this modification leads to more space consumption and hence is a space-accuracy tradeoff.

We also discussed Mira-Gries algorithm in detail. We observed that the algorithm provides close to accurate results. The algorithm may however provide inaccurate results if frequencies of different elements is very close/identical. The returned frequencies of elements are in a close range of their original frequencies. This range gradually decreases if the required number of frequent elements increase. Thus our modification of taking twice the number of frequent elements decreases the error range to almost half of the previous values, hence increasing the accuracy of the result.

6.2 Proposed Direction

In the limited time frame of the thesis project, a basic version of the algorithm has been implemented. There are various directions that can be pursued to increase the efficiency and accuracy of these algorithms. A certain degree of **parallelism** can be introduced by processing records in small-sized batches, instead of processing them one by one. The algorithm can also be extended to an framework, involving **data stream ingestion, processing and data visualization** etc. In future, these points can be taken into account.

References

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020.

[Cor16] Graham Cormode. *Misra-Gries Summaries*, pages 1334–1337. 01 2016.

[Cor17] Graham Cormode. *What is Data Sketching, and Why Should I Care?* 09 2017.

[IZ13] E. Ikonomovska and Mariano Zelke. Algorithmic techniques for processing data streams. In *Data Exchange, Information, and Streams*, 2013.

[KNW10] Daniel Kane, Jelani Nelson, and David Woodruff. An optimal algorithm for the distinct elements problem. 01 2010.

[LÖ09] LING LIU and M. TAMER ÖZSU, editors. *Flajolet-Martin Algorithm*, pages 1141–1141. Springer US, Boston, MA, 2009.

[YYT17] B. Yadranjiaghdam, S. Yasrobi, and N. Tabrizi. Developing a real-time data analytics framework for twitter streaming data. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 329–336, 2017.

final

ORIGINALITY REPORT

6%	5%	2%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	152.3.140.5 Internet Source	1%
2	docplayer.net Internet Source	1%
3	www.upgrad.com Internet Source	1%
4	www.upsolver.com Internet Source	<1%
5	dokumen.pub Internet Source	<1%
6	Goldsmith, J.. "Complexity of DNF minimization and isomorphism testing for monotone formulas", Information and Computation, 200806 Publication	<1%
7	niitlagos.com Internet Source	<1%
8	spd.group Internet Source	<1%

9	publishup.uni-potsdam.de Internet Source	<1 %
10	www.iitmjanakpuri.com Internet Source	<1 %
11	Jarosław Błasiok. "Optimal streaming and tracking distinct elements with high probability", Society for Industrial & Applied Mathematics (SIAM), 2018 Publication	<1 %
12	Sam Goundar, Akashdeep Bhardwaj, Shavindar Singh, Mandeep Singh, Gururaj H. L.. "chapter 1 Big Data and Big Data Analytics", IGI Global, 2021 Publication	<1 %
13	serval.unil.ch Internet Source	<1 %
14	techjury.net Internet Source	<1 %
15	www.tandfonline.com Internet Source	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		