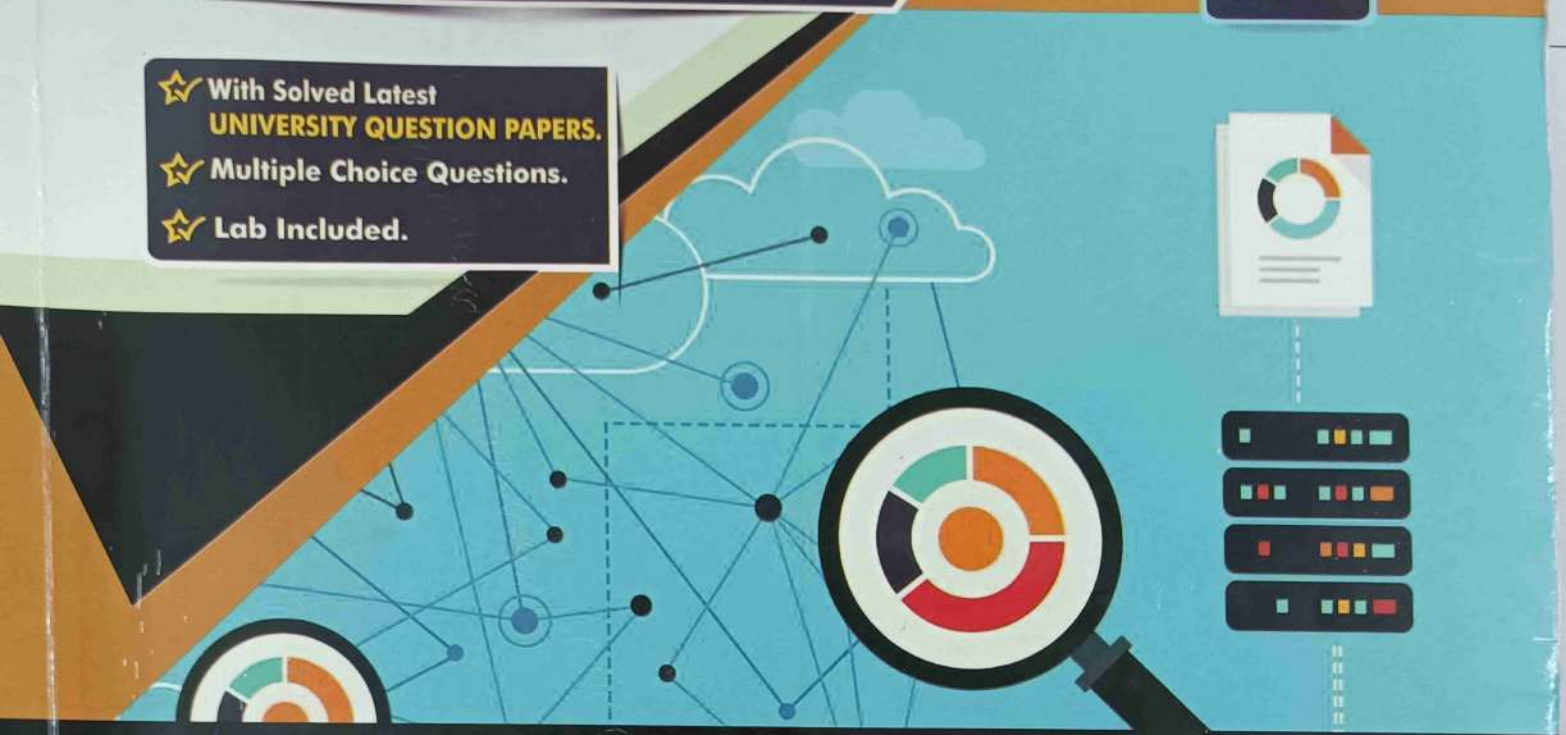
MU Sem 7 Computer Engineering

New Syllabus 2022-23

- With Solved Latest UNIVERSITY QUESTION PAPERS.
- Multiple Choice Questions.
- Lab Included.



Strictly as per the New Syllabus (REV-2019 'C' Scheme) of Mumbai University w.e.f. academic year 2022-2023

COMPULSORY SUBJECT(Code : CSC702)

Big Data Analytics

Dr. Sangeeta Vhatkar
(Thakur College of Engineering and Technology)

Rupali D. Pashte
(Shree L. R. Tiwari College of Engineering)

Dipali Pawar
(Zeal College of Engineering)

583-080-7

9789355830807

TECH-NEO PUBLICATIONS
Where Authors Inspire Innovation
A Sachin Shah Venture

www.techneobooks.in
info@techneobooks.in

M7-80A

Price ₹ 385/-

Syllabus

Mumbai University – Computer Engineering

Course Code	Course Name	Credit
CSC702	Big Data Analytics	03

Prerequisite : Database, Data mining.

Course Objectives : The course aims

1. To provide an overview of the big data platforms, its use cases and Hadoop ecosystem.
2. To introduce programming skills to build simple solutions using big data technologies such as MapReduce, Scripting for No SQL and R
3. To learn the fundamental techniques and principles in achieving big data analytics with scalability and streaming capability.
4. To enable students to have skills that will help them to solve complex real-world problems for decision support.

Course Outcomes

1. Understand the building blocks of Big Data Analytics.
2. Apply fundamental enabling techniques like Hadoop and MapReduce in solving real world problems.
3. Understand different NoSQL systems and how it handles big data.
4. Apply advanced techniques for emerging applications like stream analytics.
5. Achieve adequate perspectives of big data analytics in various applications like recommender systems, social media applications, etc.
6. Apply statistical computing techniques and graphics for analyzing big data.

Module		Detailed Content	Hours
1		**Introduction to Big Data and Hadoop**	2
	1.1	Introduction to Big Data - Big Data characteristics and Types of Big Data	
	1.2	Traditional vs. Big Data business approach	
	1.3	Case Study of Big Data Solutions	
	1.4	Concept of Hadoop, Core Hadoop Components; Hadoop Ecosystem **(Refer chapter 1)**	
2		**Hadoop HDFS and MapReduce**	8
	2.1	Distributed File Systems: Physical Organization of Compute Nodes, Large- Scale File-System Organization.	
	2.2	MapReduce: The Map Tasks, Grouping by Key, The Reduce Tasks, Combiners, Details of MapReduce Execution, Coping With Node Failures.	

Module		Detailed Content	Hours
	2.3	Algorithms Using MapReduce: Matrix-Vector Multiplication by MapReduce, Relational-Algebra Operations, Computing Selections by MapReduce, Computing Projections by MapReduce, Union ,Intersection, and Difference by MapReduce	
	2.4	Hadoop Limitations **(Refer chapter 2)**	
3		**NoSQL**	10
	3.1	Introduction to NoSQL, NoSQL Business Drivers	
	3.2	NoSQL Data Architecture Patterns: Key-value stores, Graph stores, Column family (Bigtable)stores, Document stores, Variations of NoSQL architectural patterns, NoSQL Case Study	
	3.3	NoSQL solution for big data, Understanding the types of big data problems; Analyzing big data with a shared-nothing architecture; Choosing distribution models: master-slave versus peer-to-peer; NoSQL systems to handle big data problems. **(Refer chapter 3)**	
4		**Mining Data Streams**	11
	4.1	The Stream Data Model: A Data-Stream-Management System, Examples of Stream Sources, Stream Queries, Issues in Stream Processing.	
	4.2	Sampling Data techniques in a Stream	
	4.3	Filtering Streams: Bloom Filter with Analysis.	
	4.4	Counting Distinct Elements in a Stream, Count- Distinct Problem, Flajolet-Martin Algorithm, Combining Estimates, Space Requirements	
	4.5	Counting Ones in a Window : The Cost of Exact Counts, The Datar - Gionis-Indyk-Motwani Algorithm, Query Answering in theDGIM Algorithm, Decaying Windows. **(Refer chapter 4)**	
5		**Real-Time Big Data Models**	
	5.1	A Model for Recommendation Systems, Content-Based Recommendations, Collaborative Filtering	
	5.2	Case Study: Product Recommendation	
	5.3	Social Networks as Graphs, Clustering of Social-Network Graphs, Direct Discovery of Communities in a social graph. **(Refer chapter 5)**	

Module		Detailed Content	Hours
6		**Data Analytics with R**	
	6.1	Exploring Basic features of R, Exploring RGUI, Exploring RStudio, Handling Basic Expressions in R, Variables in R, Working with Vectors, Storing and Calculating Values in R, Creating and using Objects, Interacting with users, Handling data in R workspace, Executing Scripts, Creating Plots, Accessing help and documentation in R	
	6.2	Reading datasets and Exporting data from R, Manipulating and Processing Data in R, Using functions instead of script, built-in functions in R	
	6.3	Data Visualization: Types, Applications. **(Refer chapter 6)**	

Assessment

Internal Assessment

Assessment consists of two class tests of 20 marks each. The first-class test is to be conducted when approx. 40% syllabus is completed and second class test when additional40% syllabus is completed. Duration of each test shall be one hour.

End Semester Theory Examination

1. Question paper will consist of 6 questions, each carrying 20 marks.
2. The students need to solve a total of 4 questions.
3. Question No.1 will be compulsory and based on the entire syllabus.
4. Remaining question (Q.2 to Q.6) will be selected from all the modules.

Lab Practicals

Lab Code	Lab Name	Credit
CSL7012	Big Data Analytics Lab	1

Prerequisite : C Programming Language.

Lab Objectives : Students will be able to

1. Solve Big Data problems using Map Reduce Technique and apply to various algorithms.
2. Identify various types of NoSQL databases and execute NOSQL commands
3. Understand implementation of various analytic techniques using Hive/PIG/R/Tableau, etc.
4. Apply streaming analytics to real time applications.

Lab Outcomes :

1. To interpret business models and scientific computing paradigms, and apply software tools for big data analytics.
2. To implement algorithms that uses Map Reduce to apply on structured and unstructured data
3. To perform hands-on NoSql databases such as Cassandra, HadoopHbase, MongoDB, etc.
4. To implement various data streams algorithms.
5. To develop and analyze the social network graphs with data visualization techniques.

SUGGESTED LIST OF EXPERIMENTS
(Select a case study and perform the experiments 1 to 8.).

Star (*) marked experiments are compulsory.

Sr. No.	Name of the Experiment
1*	Hadoop HDFS Practical: - HDFS Basics, Hadoop Ecosystem Tools Overview. - Installing Hadoop. - Copying File to Hadoop. - Copy from Hadoop File system and deleting file. - Moving and displaying files in HDFS. - Programming exercises on Hadoop
2	Use of Sqoop tool to transfer data between Hadoop and relational database servers. a. Sqoop - Installation. b. To execute basic commands of Hadoop eco system componentSqoop.
3*	To install and configure MongoDB/ Cassandra/ HBase/ Hypertable to execute NoSQL commands
4	Experiment on Hadoop Map-Reduce: - Write a program to implement a word count program using MapReduce.
5	Experiment on Hadoop Map-Reduce: - Implementing simple algorithms in Map-Reduce: Matrix multiplication, Aggregates, Joins, Sorting, Searching, etc
6	Create HIVE Database and Descriptive analytics-basic statistics.
7*	Data Stream Algorithms (any one): – Implementing DGIM algorithm using any Programming Language – Implement Bloom Filter using any programming language Implement Flajolet Martin algorithm using any programming language
8	Social Network Analysis using R (for example: Community Detection Algorithm)
9	Data Visualization using Hive/PIG/R/Tableau/.
10	Exploratory Data Analysis using Spark/ Pyspark.

Sr. No.	Name of the Experiment
11*	Mini Project: One real life large data application to be implemented (Use standard Datasets available on the web). – Streaming data analysis – use flume for data capture, HIVE/PYSpark for analysis of twitter data, chat data, weblog analysis etc. – Recommendation System (for example: Health Care System, Stock Market Prediction, Movie Recommendation, etc.) SpatioTemporal DataAnalytics

Term Work

1. Term work should consist of 8 experiments.
2. The final certification and acceptance of term work ensures satisfactory performance of laboratory work and minimum passing marks in term work.
3. The final certification and acceptance of term work ensures satisfactory performance of laboratory work and minimum passing marks in term work. Total 25 Marks (Experiments: 15-marks, Attendance Theory & Practical: 05-marks, Assignment: 05-marks)

Oral & Practical exam

Based on the entire syllabus of and CSC702 : Big Data Analytics and CSL702 Big Data Analytics Lab

❑❑❑

Index

❑❑❑

MODULE I

CHAPTER 1

Introduction to Big Data and Hadoop

University Prescribed Syllabus w.e.f Academic Year 2022-2023

1.1 Introduction to Big Data- Big Data characteristics and Types of Big Data

1.2 Traditional vs. Big Data business approach

1.3 Case Study of Big Data Solutions

1.4 Concept of Hadoop, Core Hadoop Components; Hadoop Ecosystem

1.1 INTRODUCTION TO BIG DATA AND HADOOP

GQ. Firstly, We need to know "what is data?"

(1) Now a day the amount of data created by various advanced technologies like Social networking sites, E-commerce etc. is very large. It is really difficult to store such huge data by using the traditional data storage facilities.

(2) Until 2003, the size of data produced was 5 billion gigabytes. If this data is stored in the form of disks it may fill an entire football field. In 2011, the same amount of data was created in every two days and in 2013 it was created in every ten minutes. This is really tremendous rate.

(3) In this topic, we will discuss about big data on a fundamental level and define common concepts related to big data. We will also see in deep about some of the processes and technologies currently being used in this field.

1.1.1 What is Big Data ?

GQ. What is Big Data?

1. Big Data is a massive collection of data that continues to grow dramatically over time.
2. It is a data set that is so huge and complicated that no typical data management technologies can effectively store or process it.
3. Big Data is like regular data, but it is much larger. A data which are very large in size.
4. Normally we work on data of size MB(WordDoc ,Excel) or maximum GB(Movies, Codes) but data in Peta bytes i.e. 1015 byte size is called Big Data.
5. It is stated that almost 90% of today's data has been generated in the past 3 years.

1.1.2 Sources of Big Data

There are various sources of big data. Now a days in number of fields such huge data get created. Following are the some of fields.

1. **Stock Exchange :** The data in the share market regarding information about prices and status details of shares of thousands of companies is very huge.

2. **Social Media Data :** The data of social networking sites contains information about all the account holders, their posts, chat history, advertisements etc. On topmost sites like facebook and whatsapp, there are literally billions of users.

3. **Video sharing portals :** Video sharing portals like youtube, Vimeo etc. contains millions of videos each of which requires lots of memory to store.

4. **Search Engine Data :** The search engines like Google and Yahoo holds lot much of metadata regarding various sites.

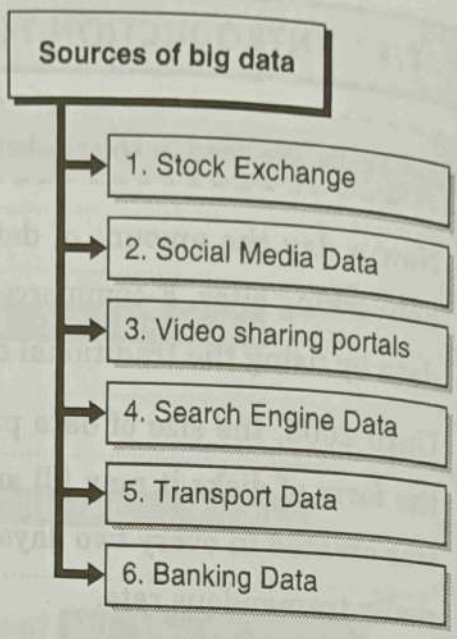


Fig. 1.1.1 : Sources of big data

5. **Transport Data :** Transport data contains information about model, capacity, distance and availability of various vehicles.

6. **Banking Data :** The big giants in banking domain like SBI or ICICI hold large amount of data regarding huge transactions of account holders.

1.2 BIG DATA CHARACTERISTICS

GQ. What are Characteristics of Big Data?

UQ. Describe any five characteristics of Big Data. **MU - Dec. 17, 5 Marks**

UQ. Explain what characteristic of Social Networks make it Big Data. **MU - May 18, 5 Marks**

GQ. Explain Big data along with 5V's

(1) Volume represents the volume i.e. amount of data that is growing at a high rate i.e. data volume in Petabytes.

(2) Value refers to turning data into value. By turning accessed big data into values, businesses may generate revenue.

(3) Veracity refers to the uncertainty of available data. Veracity arises due to the high volume of data that brings incompleteness and inconsistency.

(4) Visualization is the process of displaying data in charts, graphs, maps, and other visual forms.

(5) Variety refers to the different data types i.e. various data formats like text, audios, videos, etc.

(6) Velocity is the rate at which data grows. Social media contributes a major role in the velocity of growing data.

(7) Virality describes how quickly information gets spread across people to people (P2P) networks.

1.2.1 Volume

- As it follows from the name, big data is used to refer to enormous amounts of information.
- We are talking about not gigabytes but terabytes and petabytes of data.
- The IoT (Internet of Things) is creating exponential growth in data.
- The volume of data is projected to change significantly in the coming years.
- Hence, 'Volume' is one characteristic which needs to be considered while dealing with Big Data.

Volume

[Data at Rest]

- Terabytes, Petabytes
- Records/Arch
- Table/Files
- Distributed

1.2.2 Variety

- Variety refers to heterogeneous sources and the nature of data, both structured and unstructured.
- Data comes in different formats – from structured, numeric data in traditional databases to unstructured text documents, emails, videos, audios, stock ticker data and financial transactions.
- This variety of unstructured data poses certain issues for storage, mining and analysing data.
- Organizing the data in a meaningful way is no simple task, especially when the data itself changes rapidly.
- Another challenge of Big Data processing goes beyond the massive volumes and increasing velocities of data but also in manipulating the enormous variety of these data.

Variety

[Data in many Forms]

- Structured
- Unstructured
- Text
- Multimedia

1.2.3 Veracity

- Veracity describes whether the data can be trusted. Veracity refers to the uncertainty of available data.
- Veracity arises due to the high volume of data that brings incompleteness and inconsistency.
- Hygiene of data in analytics is important because otherwise, you cannot guarantee the accuracy of your results.
- Because data comes from so many different sources, it's difficult to link, match, cleanse and transform data across systems.
- However, it is useless if the data being analysed are inaccurate or incomplete.
- Veracity is all about making sure the data is accurate, which requires processes to keep the bad data from accumulating in your systems.

Veracity

[Data in Doubt]

- Trustworthiness
- Authenticity
- Accurate
- Availability

1.2.4 Velocity

- Velocity is the speed in which data is grows, process and becomes accessible.
- A data flows in from sources like business processes, application logs, networks, and social media sites, sensors, Mobile devices, etc.
- The flow of data is massive and continuous.
- Most data are warehoused before analysis, there is an increasing need for real-time processing of these enormous volumes.
- Real-time processing reduces storage requirements while providing more responsive, accurate and profitable responses.
- It should be processed fast by batch, in a stream-like manner because it just keeps growing every years.

Velocity

[Data in Motion]

- Streaming
- Batch
- Real / Near Time
- Processes

1.2.5 Value

- It refers to turning data into value. By turning accessed big data into values, businesses may generate revenue.
- Value is the end game. After addressing volume, velocity, variety, variability, veracity, and visualization – which takes a lot of time, effort and resources – you want to be sure your organization is getting value from the data.
- For example, data that can be used to analyze consumer behavior is valuable for your company because you can use the research results to make individualized offers.

Value

[Data into Money]

- Statistical
- Events
- Correlations

1.2.6 Visualization

- Big data visualization is the process of displaying data in charts, graphs, maps, and other visual forms.
- It is used to help people easily understand and interpret their data at a glance, and to clearly show trends and patterns that arise from this data.
- Raw data comes in a different formats, so creating data visualizations is process of gathering, managing, and transforming data into a format that's most usable and meaningful.
- Big Data Visualization makes your data as accessible as possible to everyone within your organization, whether they have technical data skills or not.

Visualization

[Data Readable]

- Readable
- Accessible
- Presentation
- Visual Forms

1.2.7 Virality

- Virality describes how quickly information gets spread across people to people (P2P) networks.
- It is measures how quickly data is spread and shared to each unique node.
- Time is a determinant factor along with rate of spread.

Virality

[Data Spread]

- P2P
- Shared
- Rate of Spread

1.3 EXAMPLES OF BIG DATA APPLICATIONS

GQ. List the examples of big data. **(2 Marks)**

GQ. Explain the examples of big data. **(6 Marks)**

There are various big data applications as shown in Fig 1.3.1

1. Fraud detection

- Fraud detection is a Big Data application example for businesses which has operations like any type of claims or transaction processing.
- Number of times the detection of fraud is concluded long after the fact. At this point the damage has been already done all that's left is to decrease the harm and revise policies to prevent it in future.

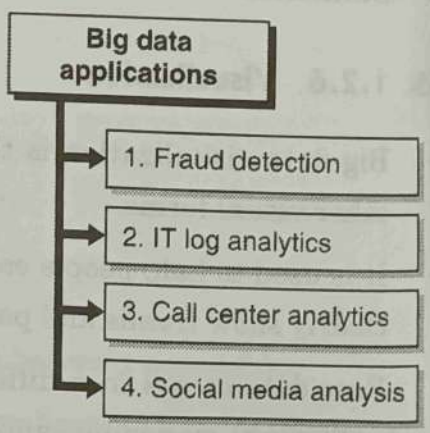


Fig. 1.3.1 : Big data applications

- The Big Data platforms can analyze claims and transactions of businesses. They identify large-scale patterns across many transactions or detect anomalous behaviour of a some user. This helps to avoid the fraud.

2. IT log analytics

- An enormous quantity of logs and trace data is generated in IT solutions and IT departments. Many times such data go unexamined: organizations simply don't have the manpower or resource to go through all such information.

- Big data has the ability to quickly identify large-scale patterns to help in diagnosing and preventing problems. It helps the organization with a large IT department.

3. Call center analytics

- Now we turn to the customer-facing Big Data application examples, of which call center analytics are particularly powerful. Without a Big Data solution, much of the insight that a call center can provide will be ignored or exposed later.
- By making sense of time/quality resolution metrics, the Big Data solutions are able to identify recurring problems or customer and staff behaviour patterns. Big data can also capture and process call content itself.

4. Social media analysis

- With the help of Social media we can observe the real-time insights into how the market is responding to products and campaigns.
- With the help of these insights, it is possible for companies to adjust their pricing, promotion, and campaign placement to get optimal results.

1.4 TYPES OF BIG DATA

GQ. What are different Types of Big Data ?

There are three types of Big Data Analytics:

1. Unstructured	2. Structured	3. Semi-structured

1.4.1 Type #1 : Unstructured

- Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it.
- Typical example of unstructured data is, a heterogeneous data source containing a combination of simple text files, images, videos like search in Google Engine.
- Now a day organizations have wealth of data available with them but unfortunately they don't know how to derive value out of it since this data is in its raw form or unstructured format.
- Human Generated Data Machine Generated Data.
- Unstructured – Example: The output returned by 'Google Search'

1.4.1(A) Characteristics of Unstructured Data

(1) Data neither conforms to a data model nor has any structure.

(2) Data can not be stored in the form of rows and columns as in Databases.

(3) Data does not follows any semantic or rules.

(4) Data lacks any particular format or sequence.

(5) Data has no easily identifiable structure.

(6) Due to lack of identifiable structure, it can not used by computer programs easily.

1.4.1(B) Sources of Unstructured Data

(1) Web pages
(2) Images (JPEG, GIF, PNG, etc.)
(3) Videos
(4) Memos
(5) Reports
(6) Word documents and PowerPoint presentations
(7) Surveys

1.4.1(C) Advantages and Disadvantages of Unstructured Data

Advantages

1. Its supports the data which lacks a proper format or sequence.
2. The data is not constrained by a fixed schema.
3. Very Flexible due to absence of schema.
4. Data is portable.
5. It is very scalable.
6. It can deal easily with the heterogeneity of sources.
7. These type of data have a variety of business intelligence and analytics applications.

Disadvantages

1. It is difficult to store and manage unstructured data due to lack of schema and structure.
2. Indexing the data is difficult and error prone due to unclear structure and not having pre-defined attributes. Due to which search results are not very accurate.
3. Ensuring security to data is difficult task.

1.4.2 Type #2 : Structured

- Any data that can be stored, accessed and processed in the form of fixed format is termed as a "Structured" data.
- Over the period of time, talent in computer science have achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also determining value out of it.
- When size of such data grows to a huge extent, typical sizes are being in the range of multiple zettabyte. Data stored in a relational database management system in one example of a structured data.
- **Structured data** is the data which conforms to a data model, has a well define structure, follows a consistent order and can be easily accessed and used by a person or a computer program.
- Structured data is usually stored in well-defined schemas such as Databases. It is generally tabular with column and rows that clearly define its attributes.
- SQL (Structured Query language) is often used to manage structured data stored in databases.

1.4.2(A) Characteristics of Structured Data

- Data conforms to a data model and has easily identifiable structure.
- Data is stored in the form of rows and columns.

Example : Database

- Data is well organised so, Definition, Format and Meaning of data is explicitly known.
- Data resides in fixed fields within a record or file.
- Similar entities are grouped together to form relations or classes.
- Entities in the same group have same attributes.
- Easy to access and query, So data can be easily used by other programs.
- Data elements are addressable, so efficient to analyse and process.

1.4.2(B) Sources of Structured Data

(1) SQL Databases
(2) Spreadsheets such as Excel
(3) OLTP Systems
(4) Online forms

(5) Sensors such as GPS or RFID tags (6) Network and Web server logs

(7) Medical devices

1.4.2(C) Advantages of Structured Data

1. Structured data have a well defined structure that helps in easy storage and access of data.
2. Data can be indexed based on text string as well as attributes. This makes search operation hassle-free.
3. Data mining is easy i.e knowledge can be easily extracted from data.
4. Operations such as Updating and deleting is easy due to well structured form of data.
5. Business Intelligence operations such as Data warehousing can be easily undertaken.
6. Easily scalable in case there is an increment of data.
7. Ensuring security to data is easy.

Structured - Example

Employee_Table

Employee_ID	Employee_Name	Gender	Department	Salary_In_lacs
1	XYX	MALE	FINANCE	850000
2	ABC	MALE	ADMIN	250000
3	PQR	FEMALE	SALES	350000
4	MNR	FEMALE	FINANCE	600000

1.4.3 Type #3 : Semi Structured

- Semi structured is the third type of big data. Semi-structured data can contain both the forms of data.
- Semi-structured data pertains to the data containing both the formats mentioned above, that is, structured and unstructured data.
- To be precise, it refers to the data that although has not been classified under a particular repository (database), yet contains vital information or tags that segregate individual elements within the data.

- Web application data, which is unstructured, consists of log files, transaction history files etc.
- Online transaction processing systems are built to work with structured data wherein data is stored in relations (tables).
- Semi-structured data is data that does not conform to a data model but has some structure. It lacks a fixed or rigid schema. It is the data that does not reside in a rational database but that have some organizational properties that make it easier to analyze. With some processes, we can store them in the relational database.

1.4.3(A) Characteristics of Semi-structured Data

1. Data does not conform to a data model but has some structure. Data can not be stored in the form of rows and columns as in Databases
2. Semi-structured data contains tags and elements (Metadata) which is used to group data and describe how the data is stored.
3. Similar entities are grouped together and organized in a hierarchy. Entities in the same group may or may not have the same attributes or properties.
4. Does not contain sufficient metadata which makes automation and management of data difficult.
5. Size and type of the same attributes in a group may differ.
6. Due to lack of a well-defined structure, it can not used by computer programs easily.

1.4.3(B) Sources of semi-structured Data:

(1) E-mails
(2) XML and other markup languages
(3) Binary executables
(4) TCP/IP packets
(5) Zipped files
(6) Integration of data from different sources
(7) Web pages

1.4.3(C) Advantages and Disadvantages of Semi-structured Data

Advantages

1. The data is not constrained by a fixed schema.
2. Flexible i.e Schema can be easily changed.
3. Data is portable.

4. It is possible to view structured data as semi-structured data.
5. Its supports users who can not express their need in SQL.
6. It can deal easily with the heterogeneity of sources.

Disadvantages

1. Lack of fixed, rigid schema make it difficult in storage of the data.
2. Interpreting the relationship between data is difficult as there is no separation of the schema and the data.
3. Queries are less efficient as compared to structured data.

Semi-structured - Example

- User can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS.
- Personal data stored in a XML file:

<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec><rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec><rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>

1.5 DIFFERENCE BETWEEN STRUCTURED, SEMI-STRUCTURED AND UN- STRUCTURED DATA

GQ. What is difference between structured, semi-structured and Un-Structured Data ?

Properties	Structured data	Semi-structured data	Unstructured data
Technology	It is based on Relational database table	It is based on XML/RDF(Resource Description Framework).	It is based on character and binary data
Transaction management	Matured transaction and various concurrency techniques	Transaction is adapted from DBMS not matured	No transaction management and no concurrency
Version management	Versioning over tuples, row, tables	Versioning over tuples or graph is possible	Versioned as a whole

Properties	Structured data	Semi-structured data	Unstructured data
Flexibility	It is schema dependent and less flexible	It is more flexible than structured data but less flexible than unstructured data	It is more flexible and there is absence of schema
Scalability	It is very difficult to scale DB schema	It's scaling is simpler than structured data	It is more scalable.
Robustness	Very robust	New technology, not very spread	–
Query performance	Structured query allow complex joining	Queries over anonymous nodes are possible	Only textual queries are possible

1.6 TRADITIONAL VS. BIG DATA BUSINESS APPROACH

GQ. What is Traditional Data & BigData ?

GQ. Explain in detail Traditional vs. Big Data Business Approach. **(5 Marks)**

1. Traditional Data

- Traditional data is the structured data which is being majorly maintained by all types of businesses starting from very small to big organizations.
- In traditional database system a centralized database architecture used to store and maintain the data in a fixed format or fields in a file. For managing and accessing the data Structured Query Language (SQL) is used.

2. Bigdata

- We can consider big data an upper version of traditional data. Big data deal with too large or complex data sets which is difficult to manage in traditional data-processing application software.
- It deals with large volume of both structured, semi structured and unstructured data. Volume, Velocity and Variety, Veracity and Value refer to the 5'V characteristics of big data.
- Big data not only refers to large amount of data it refers to extracting meaningful data by analyzing the huge amount of complex data sets.

UQ. Compare big data analytics with traditional data mining. **MU - Dec. 18, 5 Marks**

Sr. No.	Traditional Data	Big Data
1.	Traditional data is generated in enterprise level.	Big data is generated in outside and enterprise level.
2.	Its volume ranges from Gigabytes to Terabytes.	Its volume ranges from Petabytes to Zettabytes or Exabytes.
3.	Traditional database system deals with structured data.	Big data system deals with structured, semi structured and unstructured data.
4.	Traditional data is generated per hour or per day or more.	But big data is generated more frequently mainly per seconds.
5.	Traditional data source is centralized and it is managed in centralized form.	Big data source is distributed and it is managed in distributed form.
6.	Data integration is very easy.	Data integration is very difficult.
7.	Normal system configuration is capable to process traditional data.	High system configuration is required to process big data.
8.	The size of the data is very small.	The size is more than the traditional data size.
9.	Traditional data base tools are required to perform any data base operation.	Special kind of data base tools are required to perform any data base operation.
10.	Normal functions can manipulate data.	Special kind of functions can manipulate data.
11.	Its data model is strict schema based and it is static.	Its data model is flat schema based and it is dynamic.
12.	Traditional data is stable and inter relationship.	Big data is not stable and unknown relationship.

Sr. No.	Traditional Data	Big Data
13.	Traditional data is in manageable volume.	Big data is in huge volume which becomes unmanageable.
14.	It is easy to manage and manipulate the data.	It is difficult to manage and manipulate the data.
15.	Its data sources includes ERP transaction data, CRM transaction data, financial data, organizational data, web transaction data etc.	Its data sources includes social media, device data, sensor data, video, images, audio etc.

1.7 CASE STUDY OF BIG DATA SOLUTIONS

GQ. Explain Case Study of BIG Data Solutions

- Undoubtedly Big Data has become a major game change in most part of the cutting edge industries over the last few years.
- As Big Data keeps on going day by day, the number of various organizations that are adopting Big Data keeps on expanding.

Let's discuss example

- An e-commerce site XYZ (having 100 million users) wants to offer a gift voucher of 100$ to its top 10 customers who have spent the most in the previous year.
- Moreover, they want to find the buying trend of these customers so that company can suggest more items related to them.
- Issues: Huge amount of unstructured data which needs to be stored, processed and analyzed.
- Storage: This huge amount of data, Hadoop uses HDFS (Hadoop Distributed File System) which uses commodity hardware to form clusters and store data in a distributed fashion. It works on Write once, read many times principle.
- Processing: Map Reduce paradigm is applied to data distributed over network to find the required output.
- Analyze: Pig, Hive can be used to analyze the data.

- Cost: Hadoop is open source so the cost is no more an issue.
- Where are businesses finding uses for Big Data ?

Walmart

- Biggest retailer in the world and world's biggest organization by revenue. Approx. 2 million workers and 20000 stores in 28+ nations.
- It started to use Big Data concept in earlier stage.
- It used data mining to find designs pattern that can be used to give product suggestions to client, depending on which products were brought together.
- Based on data mining result, it has expanding its conversion rate of customers. Main target of Walmart is to holding customers and enhance their experience.
- Hadoop and NoSQL technologies are used to furnished these customers real time data to gathered from various sources and their effective valuable use.

Uber

- It is the best option for individuals around the globe when moving people and making conveyances.
- It utilizes individuals information of the user to intently monitor which features of services are used.
- To analyze usage pattern and to figure out where the services should be more engaged.
- It focuses around the oraganic market of the services because of which the costs of services gave changes.
- The use of data is surge pricing and its influences the rate of demand.

Netflix

- It is very popular entertainment company work in online on-request web based video streaming for its customers.
- It has been determined to be able to predict what precisely its customers will appreciate viewing with Big Data.
- Recently, Netflix begun positioning itself as a content creator, not simply a distribution medium which is solidly said based on data analytics.

- Data likes are recommendation engines take care of customers watch, regularly playback halted, ratings and so on.
- It has incorporates with Hadoop, Hive and Pig and other traditional business intelligence.

1.8 CONCEPT OF HADOOP

1.8.1 What is Hadoop ?

Hadoop is an open-source software Platform for storing massive volumes of data and running applications on clusters (groups) of commodity software. It gives us the massive data storage capability, massive computational power and the ability to handle different virtually limitless jobs that can be a running job, waiting jobs or tasks. Its main essential component is to support growing big data technologies, thereby support forward-thinking analytics like Predictive analytics, Machine learning and data mining. Hadoop has the capability to handle different modes of data such as structured, unstructured and semi-structured data. It gives us the elasticity to collect, process, and investigate data that the old data warehouses concept failed to do.

1.8.2 History of Hadoop

- The Hadoop was introduced by Doug Cutting and Mike Cafarella in 2002. Its beginning was the Google File System paper, printed by Google.
- In the year 2002, Doug Cutting and Mike Cafarella started to work on a project of Apache Nutch. It is an open source i.e. free web crawler software project.
- While working on Apache Nutch, they were facing some issue with big data. To store that data, they have invested lot of money which becomes the challenging of that project for completion.
- Due to this problem appearance of Hadoop came into existence.
- In 2003, Google presented a file system known as GFS (Google file system). It is a registered distributed file system developed to provide effective access to data.
- In year 2004, Google released the concept of a white paper on Map Reduce.
- This technique makes simpler the data processing on large clusters(groups).
- In 2005, Doug Cutting and Mike Cafarella presented a new file system known as NDFS (Nutch Distributed File System), this file system also contains Map reduce. In 2006, Doug Cutting resign Google and joined Yahoo. Based on the Nutch project,

Dough Cutting announce a new project Hadoop with a file system known as HDFS (Hadoop Distributed File System).

- Hadoop first version 0.1.0 was released and Doug Cutting gave named his project Hadoop after his son's toy elephant. In 2007, Yahoo successfully run two clusters of 1000 machines.
- In 2008, Hadoop became the quickest system to sort 1 terabyte of data on a 900-node cluster in 209 seconds. In 2013, Hadoop 2.2 was released. And Currently In 2017, Hadoop 3.0 was released.

1.8.3 Features of Hadoop

1. **Suitable for Big Data Analysis :** As Big Data manages to be distributed and unstructured in nature, Hadoop clusters are well-matched for analysis of Big Data. Meanwhile it is processing logic (not the actual data) that flows to the computing nodes and less network bandwidth is spent. This concept is called as data locality which helps to increase the productivity of Hadoop based applications.
2. **Scalability :** Hadoop clusters can easily be scaled to any amount by adding extra cluster nodes and thus allows for the growth of Big Data. Also, scaling does not require adjustments to application logic.
3. **Fault Tolerance :** Hadoop network has a facility to duplicate the input data on to other cluster nodes. So, in the event of a cluster node failure the data processing can still process data by using data stored on another cluster node.

1.8.4 Advantages of Hadoop

1. **Fast :** In HDFS the data distributed over the cluster and mapped such a way which helps in faster recovery. Even the tools to process the data are often on the same servers, thus reducing the processing time can be efficient way to manage the data. It also processes terabytes of data in minutes and Peta bytes in hours.
2. **Scalable :** Hadoop cluster can be extended by just adding nodes in the cluster so failure chance can be less.
3. **Cost Effective :** Hadoop is open source and uses commodity hardware to store data, it is cheaper as compared to traditional RDMS.
4. **Tough to failure :** HDFS has the property with which it can duplicate data over the network, so if one node is down or some other network failure happens, then Hadoop takes the backup data and use it. Normally, data are replicated thrice but the replication factor is configurable.

1.8.5 Challenges of Hadoop

1. Hadoop is a complex distributed system with low-level Application programming interface.
2. Specialized skills are required for using Hadoop and prevent most developers from efficiently bringing solutions.
3. Business logic and infrastructure APIs have no clear separation therefore burdening come on app developers.
4. Automated testing of end-to-end solutions is unfeasible or terrible.
5. Common data patterns often require but does not support data steadiness and accuracy.
6. Hadoop is more than just disconnected storage.
7. Hadoop is a various collection of many open source projects.
8. Understanding multiple technologies and hand-coding combination between them is difficult.
9. Significant effort is wasted on simple tasks like data absorptions and ETL (Extract, Transform, Load).
10. Real-time and batch ingestion requires extremely integration numerous components.
11. Different processing models require data to be stored in specific ways so that data can be handle easily.

1.8.6 Architecture of Hadoop

UQ. Explain the physical architecture of Hadoop. **(MU - Dec. 18, 4 Marks)**

Hadoop basically has Master-Slave Architecture for storing data and distributed processing of data by using MapReduce and HDFS methods. In Hadoop, master or slave system can be set up on the cloud or on premise.

1. **NameNode :** NameNode represents all files and directory which is used in the namespace.
2. **DataNode :** DataNode helps you to manage the states of an HDFS node and allows you to cooperate with its blocks.

3. **Master Node :** The master node allows you to conduct parallel processing of data by use Hadoop MapReduce.

4. **Slave node :** The slave nodes are the supplementary machines in the Hadoop cluster which permits you to store data to conduct complex calculations. And all these slave node derives with Task Tracker and DataNode which allows to synchronize the processes with the NameNode and Job Tracker correspondingly.

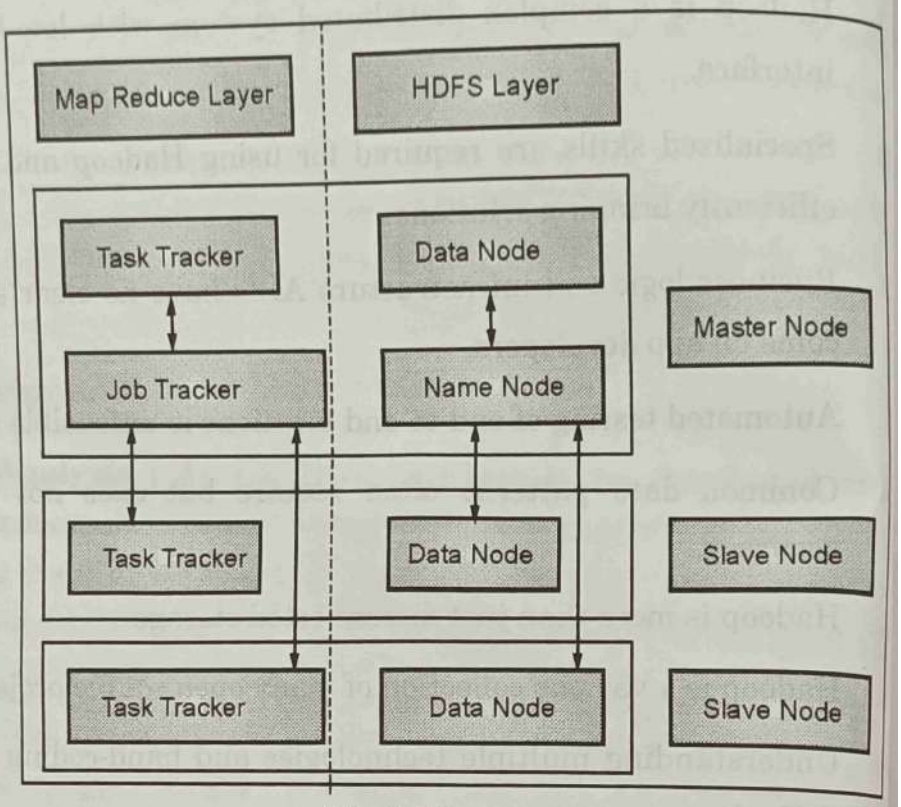


Fig. 1.8.1

1.9 CORE HADOOP COMPONENTS, HADOOP ECOSYSTEM

1.9.1 Core Hadoop Components

UQ. Explain Hadoop ecosystem with core components. **(MU - Dec. 18, 4 Marks)**

Hadoop concept is an open-source big data technology platform which allows computer networks to perform complex processing and gives results that are forever available even when a condition arises when a few nodes are in unavailable state for functional processing. There are a few important Hadoop core components that it can perform through several cloud-based platforms.

The core components are described as follows :

1. **The Distributed File System :** The furthermost important of the Hadoop core components is the idea of the Distributed File System. It permits the platform to access widely storage devices and use the basic tools to read the obtainable data as well perform the essential analysis. This is a unique file system because it lies overhead the individual file system of the network node computers and allows matchless functionality. The DFS of Hadoop can perform the required data

achievements without worrying about the operating system of the individual computers. This allows the network to service superior power and never face the problem of having to observe with the different computer systems available for use. It also allows the connection to other central components, such as MapReduce.

2. **MapReduce :** MapReduce is another of Hadoop core components that trusts two separate functions, which are required for execution of smart big data operations. The first operation is to read the data from a database and doing inserting operation of data it in a suitable format for performing the required analysis. This is the function which is known as a mapping activity. It basically allows a platform to make the data for the analytical requirements in a common format to allow any computer to do further procedures as per requirements. The next method carried out is mathematical operation. This operation is named as reduction (decrease), because it usually reduces the available map to a set of proper values. Together, these functions are existing in a single module and perform the whole operation which delivers information from available different data sources.

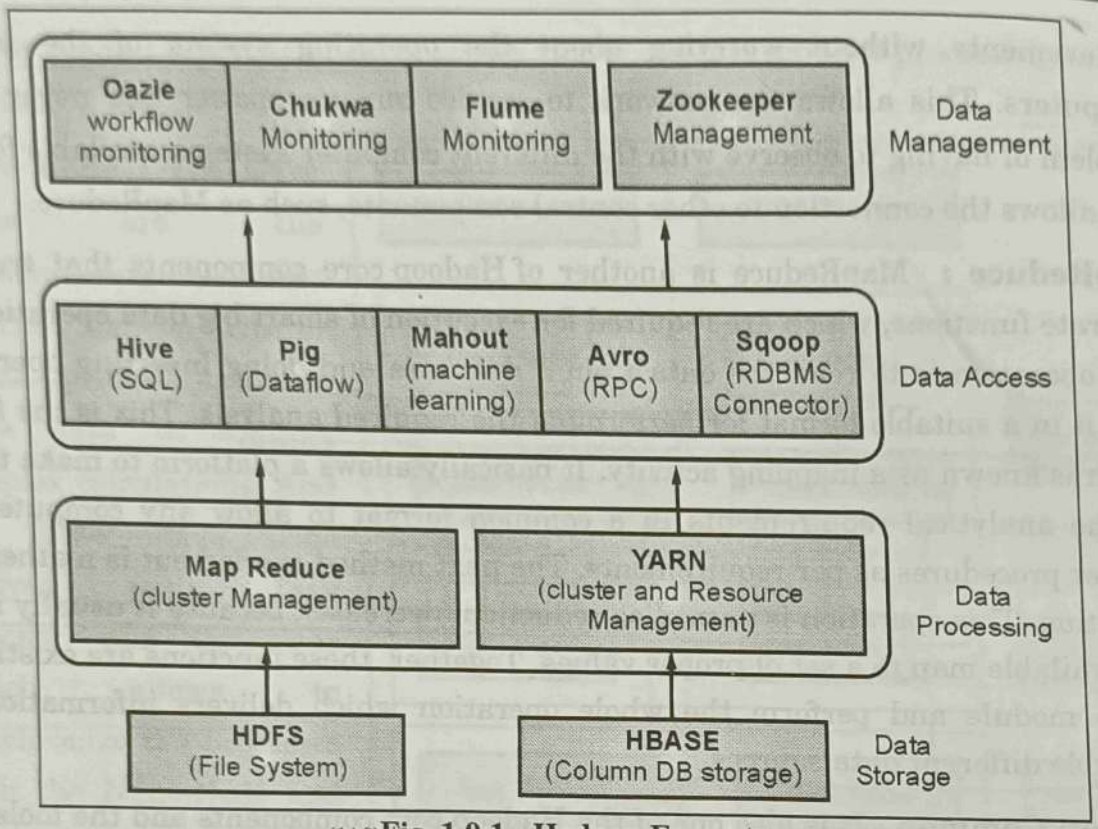
3. **Hadoop Common :** It is also one of the Hadoop core components and the tools which allow any computer to become part of the Hadoop network unrelatedly of the operating system or the present hardware. This module uses Java tools and parts that create a virtual machine and allows the Hadoop platform to store data under its path specific file system. This component named as common as it offers the required common functionality, which removes the difference between the different hardware nodes which may be connected to the network at any time and worldwide.

4. **YARN :** It is the component which accomplishes all the information sources that store the data and then route the required analysis. It is a system which accomplishes the available resources in a network group, as well as schedule the processing tasks to come up with a clever solution for every big data want on the system.

1.9.2 Hadoop Ecosystem Overview

UQ. How big data problems are handled by Hadoop system? **(MU - May 19, 5 Marks)**

Hadoop ecosystem is a platform or framework which assistances in solving the big data problems. It includes different components and services (ingesting, storing, analysing, and maintaining) inside of it. Most of the services available in the Hadoop ecosystem which contains the main four core components of Hadoop which include HDFS, YARN, MapReduce and Common. Hadoop ecosystem contains both Apache Open Source projects and other wide variation of marketable tools and solutions.



(1A9)**Fig. 1.9.1 : Hadoop Ecosystem**

Some of the open source examples are Spark, Hive, Pig, Sqoop and Oozie. As we have got some idea about what Hadoop ecosystem is, what it does, and what are its components, let's discuss each concept in detail.

☞ Components of Hadoop Ecosystem

As we have seen an overview of Hadoop Ecosystem and well-known open-source examples, now we are going to discuss deeply the list of Hadoop Components individually and their specific roles in the big data processing. The components of Hadoop ecosystems are :

1. **HDFS :** Hadoop Distributed File System is the spine of Hadoop which runs on java language and allows to stores data in Hadoop applications. They act as a command interface to relate with Hadoop. There are two components of HDFS that are Data node and Name Node. Name node is the main node which manages file systems and activates all data nodes and maintains records of metadata updating. In some case of deletion of data, they automatically record in its Edit Log file. Data Node requires massive storage space due to the operation of reading and writing. They work according to the orders of the Name Node. The data nodes are hardware in the distributed system.

2. **HBASE :** It is an open-source framework storing all types of data and doesn't support the SQL database. They run on top of HDFS and they are written in java language. Most of the companies use them for capturing the features like supporting all types of data, high security, use of HBase tables etc. They play a dynamic role in analytical processing. The two major components of HBase are HBase master and Regional Server. The HBase master is answerable for load balancing in a Hadoop cluster and controls the failure occurred. They are answerable for performing management role. The role of the regional server would be a worker node and responsible for reading, writing data in the cache.

3. **YARN :** It is an important component in the ecosystem and named as operating system in Hadoop which delivers resource management and job scheduling task. The components are Resource and Node manager, Application manager and container. They also act as protectors across Hadoop clusters. They help in the dynamic allocation of cluster resources, increase in data centre process and allows multiple access engines.

4. **Sqoop :** It is a tool that helps in data transfer between HDFS and MySQL and gives hand-on to import and export of data and as well they have a connector for fetching and linking a data.

5. **Apache Spark :** It is an open-source cluster computing framework for data analytics and an important data processing engine. It is written in Scala and comes with packaged standard libraries. They are also used by many companies for their high processing speed and stream processing.

6. **Apache Flume :** It is a distributed service collecting a huge quantity of data from the source (web server) and moves back to its source and relocated to HDFS. It has its own three components are Source, sink, and channel.

7. **Apache Pig :** Data Handling of Hadoop is performed by Apache Pig and by using of Pig Latin Language. It helps in the reuse of code and easy to read and write code.

8. **Hive :** It is an open-source Platform software for execution of data warehousing ideas, it accomplishes to query for large data sets stored in HDFS. It is built on upper of the Hadoop Ecosystem, the language used is Hive Query language. The user submits the hive queries with metadata which converts SQL into Map-reduce jobs and directs to the Hadoop cluster which consists of one master and many slaves.

9. **Apache Drill :** Apache Drill is an open-source SQL engine which procedures on non-relational databases and File system. They are intended to support Semi-structured databases originate in Cloud storage. They have good Memory management abilities to maintain junk collection

10. **Apache Zookeeper :** It is an API that supports in distributed Organisation. Here a node called Z node which is created by an application in the Hadoop cluster. They do services like Synchronization, Configuration etc. Its categories out the time-consuming coordination in the Hadoop Ecosystem.

11. **Oozie :** Oozie is a java web application which maintains several workflows in a Hadoop cluster. Having Web service APIs controls over a job is done anywhere. It is widespread for handling Multiple jobs successfully.

1.9.3 Examples of Hadoop Ecosystem

Regarding map-reduce, we can see an example and use case. One such case is Skybox which uses Hadoop to analyse a huge volume of data. Hive can find simplicity on Facebook. Frequency of word count in a sentence using map-reduce. MAP performs by taking the count as input and perform functions such as Filtering and sorting and the reduce () consolidates the result. Hive example on taking students from different states from student databases using various DML commands.

1.9.4 Limitations

UQ. State limitations of Hadoop ecosystem. **(MU - Dec. 18, 2 Marks)**

1. Issue with Small Files	2. Slow Processing Speed
3. Support for Batch Processing only	4. No Real-time Data Processing
5. No Delta Iteration	6. Latency
7. Security	8. No Abstraction
9. No Caching	10. Lengthy Line of Code

1. Issue with Small Files

Hadoop is not suitable for the small data. **(HDFS) Hadoop distributed file system** wants the capability to professionally support the arbitrary reading of the small files since of it is high volume design. Small files are the main problematic in HDFS. A small file is expressively minor than the HDFS block size (default 128MB). If we are storing these vast numbers of small files, then HDFS cannot handle these files while HDFS is working accurately with a small unit of large files by storing large data sets rather than storing several small files. If there are several small files, then the **NameNode** will get burden meanwhile it stores the namespace of HDFS.

2. Slow Processing Speed

In Hadoop, with a support of the parallel and distributed algorithm the MapReduce procedure the large data sets. There are some tasks that we need to perform like Map and Reduce and thus the MapReduce needs a lot of time to complete these tasks thus by increasing the time interval. The Data is spread and handled over the cluster in MapReduce which increases the period and decreases the handling and execution speed.

3. Support for Batch Processing only

Hadoop supports the batch processing only and it does not procedure the streamed data and later complete performance is slower. The MapReduce framework of the Hadoop does not influence the memory of the **Hadoop cluster** to the extreme level.

4. No Real-time Data Processing

Apache Hadoop is for the operation of batch processing, which allow it to take a vast amount of data in input and execute it and generate the outcome. Even though batch processing is very well-organized for processing a data of high volume dependent on the size of the data that is being processes and the computational power of the system but basically an output can be delay so the Hadoop is not appropriate for Real-time data processing.

5. No Delta Iteration

Hadoop isn't well-organized for the constant processing basically the Hadoop doesn't support the repeated data flow i.e. sequence of phases during which the respective output of the earlier phase is the input to the succeeding phase.

6. Latency

The Hadoop MapReduce framework is that the comparatively slower so meanwhile its supporting the various format, structure and vast capacity of information or data. In **MapReduce**, Map takes the collection of the data and decodes it into the alternative set of data where the separate elements are fragmented down into **key-value pairs** and reduce the output from the map as input and process extra and MapReduce requires plenty of the time to accomplish these tasks thus by increasing the latency.

7. Security

Hadoop is challenges in handling the compound application. If the user doesn't know the way to enable a platform who is managing the platform that the data can be in the danger.

At storage and network levels, Hadoop is missing the encryption part, which can leads to the major point of concern. Hadoop supports **Kerberos authentication**, which is tough to manage. HDFS **supports access control lists** (ACLs) and a standard file permissions model. However, third-party vendors have enabled an association to influence the **Active Directory Kerberos** and **LDAP** for verification.

8. **No Abstraction**

   Hadoop doesn't have any kind of abstraction; thus, MapReduce creators need to the hand code for each process which makes it difficult to work.

9. **No Caching**

   Hadoop is not well-organized for caching. In Hadoop, MapReduce cannot store the intermediate data in the memory for a additional condition which reduces the performance of the Hadoop.

10. **Lengthy Line of Code**

    Hadoop has approximately about 1,20,000 codes of line, the number of lines generated by the bugs and it will take more period of the time to execute the program.

UQ. Explain how Hadoop goals are covered in Hadoop distributed file system.

(MU - May 19, 10 Marks)

HDFS and MapReduce are two important major components of Hadoop, whereas HDFS is useful for infrastructure point of view and MapReduce is useful for programming concept. To understand the concept behind scalability of Hadoop from single-node to a thousand- nodes cluster, HDFS is very useful. It covered the goals of Hadoop as follows :

1. **Handling of large dataset :** As Hadoop supports distributed storage and processing of large data set, HDFS architecture is designed as it most useful to store and retrieve large data.
2. **Fault tolerance and data replication :** In HDFS, the data files are divided into big blocks of data and for fault tolerance each one block is stored on three nodes from which two nodes are from same rack and one is from a different rack. A block is considered as the amount of data stored on each data node. The redundancy of data leads to robustness, fault detection, quick recovery of data and scalability.
3. **Commodity Hardware :** HDFS consider that the cluster will run on common hardware such as less expensive or ordinary machines. And an important feature of Hadoop is that Hadoop can be installed on any average commodity hardware. For installation and execution of Hadoop It does not require any super computers or high-end hardware. This reduces the overall cost.

4. **Data Locality concept :** Data locality means locating computation logic near to the data, instead of moving data to the computation logic or application space. This reduces the bandwidth utilization in a system. HDFS provide interfaces for applications to relocate themselves closer to the location where data.

UQ. How big data analytics can be useful in the development of Digital India? **(MU - Dec. 18, 5 Marks)**

- "Digital India" program is get introduce in in our country with the vision to make over India into a digitally empowered society and build knowledge economy.

  Its vision is mostly focused on three main areas :

  i) Digital Infrastructure as a basic utility to every citizen,
  ii) Governance and services on demand and
  iii) Digital empowerment of citizens of India.

- Subsequently, to fulfil this vision, availability of different Data resources has got increased.
- In Big data huge amount of data collected and stored whether it is in structured or unstructured form or semi-structured. This data may contains various business related transactions, email, images, audio, surveillance camera videos, logs and unstructured data from blogs or messages from social media, medical data, banks related transaction data, e-Governance data, media data, defence related data, IT sectors.
- If this data get efficiently cleaned and then get analyzed, it can helpful in data visualization for business trade for various enterprises or organizations.
- This digital technology has make the progress enterprises or organizations more easier. Data from collected from tweets, various blogs and other social networks sites can useful to an enterprise or organization to analyze consumer's view. It will helpful for them to understand needs and choices of their customers.
- The big data supports five V"s attributes : Volume, Velocity, Variety, Veracity and Values.

  1. **Volume :** Means the amount of data it contains. Now in this digital world , organization may have huge amount of data and to deal with this huge data big data uses the concept of distributed systems so that they can store their data and analyse it through-out databases that are scattered around the world and linked via internet

2. **Velocity :** In big data refers to the data transfer from various digital platform such as online systems, various sensors, social media, live web capture, etc. As we all known that social media messages get viral in seconds of time. Such scenario of big data represents to its velocity. In some cases, it needs to be analysed the data without storing it.

3. **Variety :** Refers to different types of data from many sources, it may be in structured, unstructured or semi-structured. Big data analytics provides the facility to integrate this data.

4. **Veracity :** Means Complexity which indicate that the data must be able to transfer via different multiple data centers such as the cloud and geographical zones. Managing or analyzing this huge data is very complex task.

5. **Value :** Is measure of visible or invisible benefits gained by organisation by the use of Big Data. It is more useful when the data from Big Data is converted into value then it gets used.

Chapter Ends...

❑❑❑

MODULE II

CHAPTER 2

Hadoop HDFS and MapReduce

University Prescribed Syllabus w.e.f Academic Year 2022-2023

2.1 Distributed File Systems: Physical Organization of Compute Nodes, Large-Scale File-System Organization.

2.2 MapReduce: The Map Tasks, Grouping by Key, The Reduce Tasks,Combiners, Details of MapReduce Execution, Coping With Node Failures.

2.3 Algorithms Using MapReduce: Matrix-Vector Multiplication by MapReduce,Relational-Algebra Operations, Computing Selections by MapReduce,Computing Projections by MapReduce, Union ,Intersection,and Difference by MapReduce

2.4 Hadoop Limitations

2.1 ARCHITECTURE OF HADOOP HDFS

GQ. Explain Architecture of HDFS in detail. **(5 Marks)**

- The HDFS is the primary storage system used by Hadoop applications. HDFS is a distributed file system and a framework provided by Hadoop for the analysis and transformation of huge data sets which uses the MapReduce paradigm. The HDFS is based on Google File System (GFS). It provides high-performance access to data across Hadoop clusters (thousands of computers), HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.
- HDFS is usually deployed on commodity hardware of low-cost where the possibility of server failures is common. The file system is designed to be highly fault-tolerant.

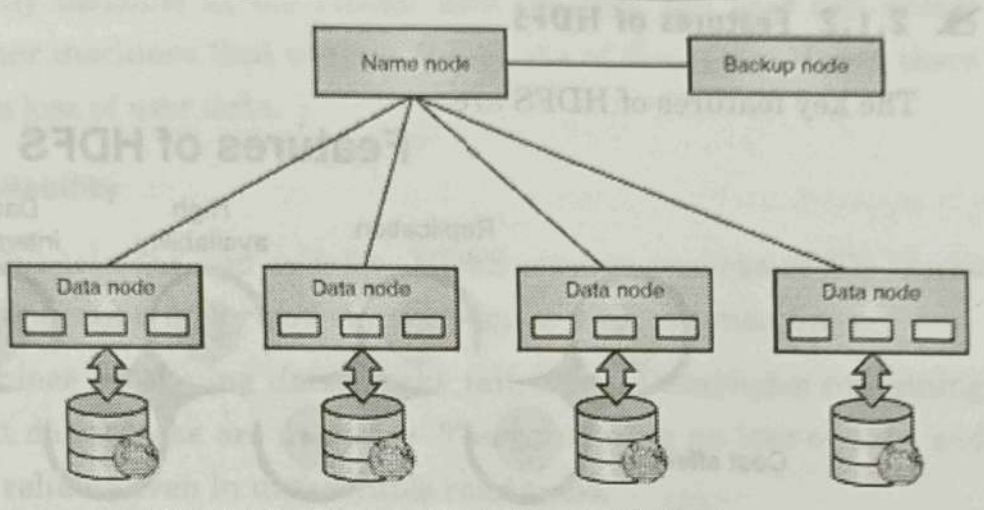


Fig. 2.1.1 : Hadoop Distributed File System

- The HDFS facilitates the rapid transfer of data between different computer nodes and enables Hadoop systems to precede its execution even if one or more nodes gets failed. That decreases the risk of catastrophic failure, even in the event that numerous nodes fail.
- The architecture used by HDFS is known as master/slave architecture.
- NameNode which manages the metadata of file system and DataNode which stores the actual data.
- The HDFS namespace is a hierarchy of files and directories. Inodes are used to represent these file and directories. Inodes are used to record attributes such as permissions, modification and access times etc. The file content is split into large blocks and each block of the file is independently replicated at multiple DataNodes.
- The tree structure of namespace is maintained by the NameNode. It maps the blocks to DataNodes. In a cluster there may be hundreds of DataNodes and thousands of HDFS clients per cluster, as number of application tasks can be executed by each DataNode simultaneously.

2.1.1 Advantages and Disadvantages of HDFS

Advantages of HDFS	Disadvantages of HDFS
1. High scalability	1. Still rough - means software under active development.
2. Low limitation.	2. Programming model is very restrictive.
3. Open source.	3. Cluster management is high.
4. Low cost.	

2.1.2 Features of HDFS

The key features of HDFS are:

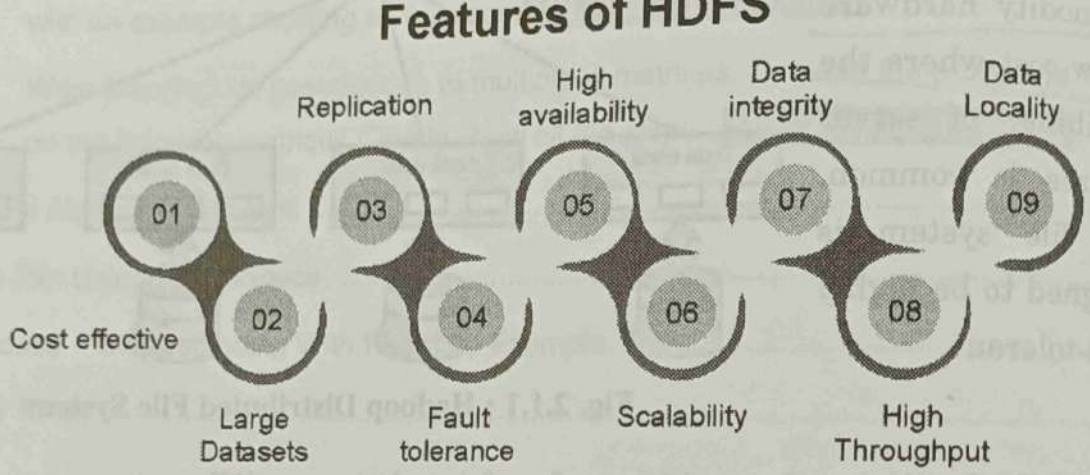


Fig. 2.1.2

1. Cost-effective	2. Large Datasets/ Variety and volume of data
3. Replication	4. Fault Tolerance and reliability
5. High Availability	6. Scalability
7. Data Integrity	8. High Throughput
9. Data Locality	

1. Cost-effective

In HDFS architecture, the DataNodes, which stores the actual data are inexpensive commodity hardware, thus reduces storage costs.

2. Large Datasets/ Variety and volume of data

HDFS can store data of any size (ranging from megabytes to petabytes) and of any formats (structured, unstructured).

3. Replication

- Data Replication is one of the most important and unique features of HDFS. In HDFS replication of data is done to solve the problem of data loss in unfavorable conditions like crashing of a node, hardware failure, and so on.
- The data is replicated across a number of machines in the cluster by creating replicas of blocks. The process of replication is maintained at regular intervals of time by HDFS and HDFS keeps creating replicas of user data on different machines present in the cluster.
- Hence whenever any machine in the cluster gets crashed, the user can access their data from other machines that contain the blocks of that data. Hence there is no possibility of a loss of user data.

4. Fault Tolerance and reliability

- HDFS is highly fault-tolerant and reliable. HDFS creates replicas of file blocks depending on the replication factor and stores them on different machines.
- If any of the machines containing data blocks fail, other DataNodes containing the replicas of that data blocks are available. Thus, ensuring no loss of data and makes the system reliable even in unfavorable conditions.

5. High Availability

- The High availability feature of Hadoop ensures the availability of data even during NameNode or DataNode failure.
- Since HDFS creates replicas of data blocks, if any of the DataNodes goes down, the user can access his data from the other DataNodes containing a copy of the same data block.
- Also, if the active NameNode goes down, the passive node takes the responsibility of the active NameNode. Thus, data will be available and accessible to the user even during a machine crash.

6. Scalability

- As HDFS stores data on multiple nodes in the cluster, when requirements increase, we can scale the cluster.
- There is two scalability mechanism available: Vertical scalability – add more resources (CPU, Memory, Disk) on the existing nodes of the cluster.

- Another way is horizontal scalability – Add more machines in the cluster. The horizontal way is preferred since we can scale the cluster from 10s of nodes to 100s of nodes on the fly without any downtime.

7. Data Integrity

- Data integrity refers to the correctness of data. HDFS ensures data integrity by constantly checking the data against the checksum calculated during the write of the file.
- While file reading, if the checksum does not match with the original checksum, the data is said to be corrupted. The client then opts to retrieve the data block from another DataNode that has a replica of that block. The NameNode discards the corrupted block and creates an additional new replica.

8. High Throughput

Hadoop HDFS stores data in a distributed fashion, which allows data to be processed parallelly on a cluster of nodes. This decreases the processing time and thus provides high throughput.

9. Data Locality

- Data locality means moving computation logic to the data rather than moving data to the computational unit.
- In the traditional system, the data is brought at the application layer and then gets processed.
- But in the present scenario, due to the massive volume of data, bringing data to the application layer degrades the network performance.
- In HDFS, we bring the computation part to the Data Nodes where data resides. Hence, with Hadoop HDFS, we are not moving computation logic to the data, rather than moving data to the computation logic. This feature reduces the bandwidth utilization in a system.

2.2 PHYSICAL ORGANIZATION OF COMPUTER NODES

GQ. What do you mean by Physical Organization of Compute Nodes? **(5 Marks)**

GQ. Write short note on cluster Computing. **(5 Marks)**

GQ. What are the problems faced by parallel computing architecture and state the solution in detail? **(5 Marks)**

- The new parallel-computing architecture, sometimes called cluster computing, is organized as follows. Compute nodes are stored on racks, perhaps 8–64on a rack.
- The nodes on a single rack are connected by a network, typically gigabit Ethernet. There can be many racks of compute nodes, and racks are connected by another level of network or a switch.
- The bandwidth of inter-rack communication is somewhat greater than the intra rack Ethernet, but given the number of pairs of nodes that might need to communicate between racks, this bandwidth may be essential.
- However, there may be many more racks and many more compute nodes per rack. It is a fact of life that components fail, and the more components, such as compute nodes and interconnection networks, a system has, the more frequently something in the system will not be working at any given time. Some important calculations take minutes or even hours on thousands of compute nodes. If we had to abort and restart the computation every time one component failed, then the computation might never complete successfully.
- The solution to this problem takes two forms:
  1. Files must be stored redundantly. If we did not duplicate the file at several compute nodes, then if one node failed, all its files would be unavailable until the node is replaced. If we did not back up the files at all, and the disk crashes, the files would be lost forever.
  2. Computations must be divided into tasks, such that if any one task failsto execute to completion, it can be restarted without affecting other tasks. This strategy is followed by the map-reduce programming system

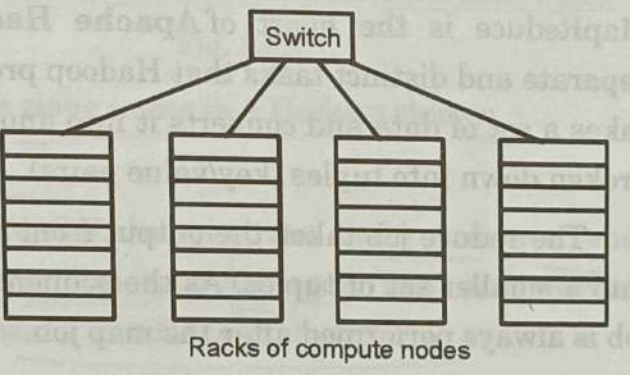


Fig. 2.2.1 : Architecture of a large scale computing system

2.3 LARGE SCALE FILE SYSTEM ORGANIZATION

To exploit cluster computing, files must look and behave somewhat differently from the conventional file systems found on single computers. This new file system, often called a distributed file system or DFS (although this term had other meanings in the past), is typically used as follows :

- Files can be enormous, possibly a terabyte in size. If you have only smallfiles, there is no point using a DFS for them.

- Files are rarely updated. Rather, they are read as data for some calculation, and possibly additional data is appended to files from time to time.
- Files are divided into chunks, which are typically 64 megabytes in size. Chunks are replicated, perhaps three times, at three different compute nodes. Moreover, the nodes holding copies of one chunk should be located on different racks, so we don't lose all copies due to a rack failure. Normally, both the chunk size and the degree of replication can be decided by the user.
- To find the chunks of a file, there is another small file called the master node or name node for that file. The master node is itself replicated, and a directory for the file system as a whole knows where to find its copies. The directory itself can be replicated, and all participants using the DFS know where the directory copies are.

2.4 INTRODUCTION TO MAPREDUCE

UQ. What is MapReduce? Explain the role of Combiner with help of an example. **(MU - Dec. 18, 10 Marks)**

UQ. Explain Concept of MapReduce using an example. **(MU - Dec. 17, 5 Marks)**

MapReduce is a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. As the processing component, MapReduce is the heart of **Apache Hadoop.** The term "MapReduce" refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

2.4.1 How MapReduce Works

- At a high level, MapReduce breaks input data into fragments and distributes them across different machines.
- The input fragments consist of key-value pairs. Parallel map tasks process the chunked data on machines in a cluster. The mapping output then serves as input for the reduce stage. The reduce task combines the result into a particular key-value pair output and writes the data to HDFS.
- The Hadoop Distributed File System usually runs on the same set of machines as the MapReduce software. When the framework executes a job on the nodes that also store the data, the time to complete the tasks is reduced significantly.

2.4.2 Basic Terminology of Hadoop MapReduce

As we mentioned above, MapReduce is a processing layer in a Hadoop environment. MapReduce works on tasks related to a job. The idea is to tackle one large request by slicing it into smaller units.

JobTracker and TaskTracker

- In the early days of Hadoop (version 1), JobTracker and TaskTracker daemons ran operations in MapReduce. At the time, a Hadoop cluster could only support MapReduce applications.
- A JobTracker controlled the distribution of application requests to the compute resources in a cluster. Since it monitored the execution and the status of MapReduce, it resided on a master node.
- A TaskTracker processed the requests that came from the JobTracker.

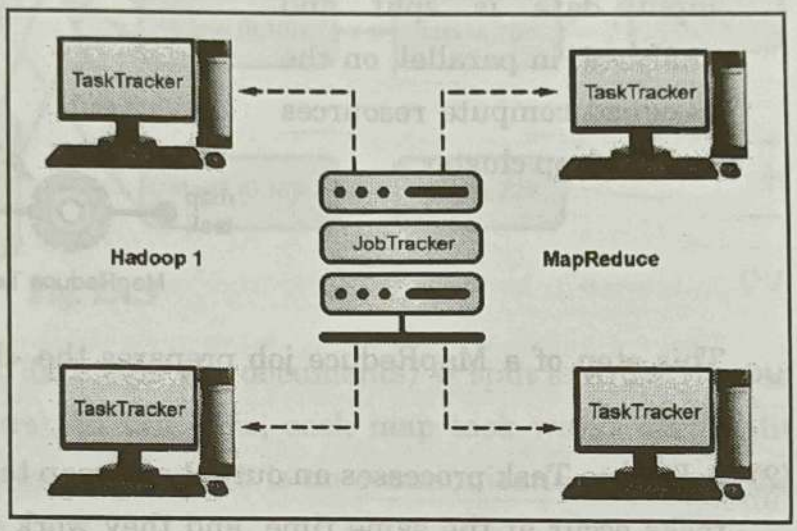


Fig. 2.4.1

- All task trackers were distributed across the slave nodes in a Hadoop cluster.

YARN

Later in Hadoop version 2 and above, YARN became the main resource and scheduling manager. Hence the name Yet Another Resource Manager. Yarn also worked with other frameworks for the distributed processing in a Hadoop cluster.

MapReduce Job

- A MapReduce job is the top unit of work in the MapReduce process. It is an assignment that Map and Reduce processes need to complete. A job is divided into smaller tasks over a cluster of machines for faster execution.
- The tasks should be big enough to justify the task handling time. If you divide a job into unusually small segments, the total time to prepare the splits and create tasks may outweigh the time needed to produce the actual job output.

MapReduce Task

MapReduce jobs have two types of tasks.

(1) A Map Task is a single instance of a MapReduce app. These tasks determine which records to process from a data block. The input data is split and analyzed, in parallel, on the assigned compute resources in a Hadoop cluster.

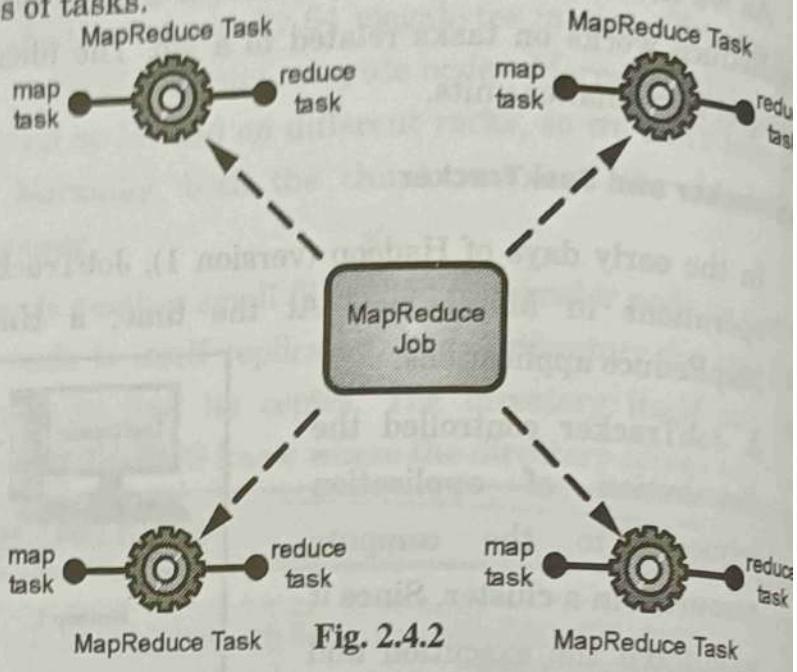


Fig. 2.4.2

This step of a MapReduce job prepares the <key, value> pair output for the reduce step.

(2) A Reduce Task processes an output of a map task. Similar to the map stage, all reduce tasks occur at the same time, and they work independently. The data is aggregated and combined to deliver the desired output. The final result is a reduced set of <key, value> pairs which MapReduce, by default, stores in HDFS.

2.4.3 How Hadoop Map and Reduce Work Together

- As the name suggests, MapReduce works by processing input data in two stages - Map and Reduce. To demonstrate this, we will use a simple example with counting the number of occurrences of words in each document.
- The final output we are looking for is: How many times the words Apache, Hadoop, Class, and Track appear in total in all documents.
- For illustration purposes, the example environment consists of three nodes. The input contains six documents distributed across the cluster. We will keep it simple here, but in real circumstances, there is no limit. You can have thousands of servers and billions of documents.

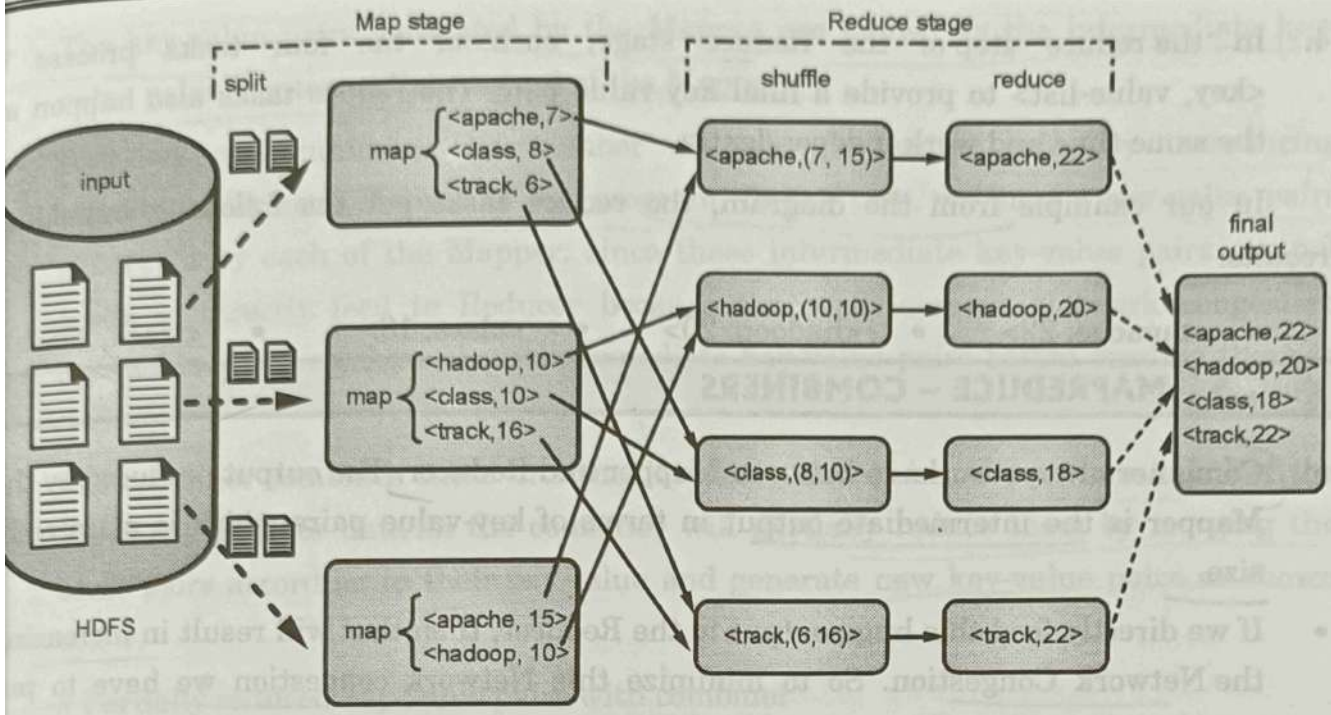


Fig. 2.4.3

First, in the map stage, the input data (the six documents) is split and distributed across the cluster (the three servers). In this case, each map task works on a split containing two documents. During mapping, there is no communication between the nodes. They perform independently.

Then, map tasks create a <key, value> pair for every word. These pairs show how many times a word occurs. A word is a key, and a value is its count. For example, one document contains three of four words we are looking for: Apache 7 times, Class 8 times, and Track 6 times. The key-value pairs in one map task output look like this:

- <apache, 7>
- <class, 8>
- <track, 6>

This process is done in parallel tasks on all nodes for all documents and gives a unique output.

After input splitting and mapping completes, the outputs of every map task are shuffled. This is the first step of the Reduce stage. Since we are looking for the frequency of occurrence for four words, there are four parallel Reduce tasks. The reduce tasks can run on the same nodes as the map tasks, or they can run on any other node.

The shuffle step ensures the keys Apache, Hadoop, Class, and Track are sorted for the reduce step. This process groups the values by keys in the form of <key, value-list> pairs.

4. In the reduce step of the Reduce stage, each of the four tasks process a <key, value-list> to provide a final key-value pair. The reduce tasks also happen at the same time and work independently.

In our example from the diagram, the reduce tasks get the following individual results:

- <apache, 22>
- <hadoop, 20>
- <class, 18>
- <track, 22>

2.5 MAPREDUCE – COMBINERS

- Combiner always works in between Mapper and Reducer. The output produced by the Mapper is the intermediate output in terms of key-value pairs which is massive in size.
- If we directly feed this huge output to the Reducer, then that will result in increasing the Network Congestion. So to minimize this Network congestion we have to put combiner in between Mapper and Reducer.
- These combiners are also known as semi-reducer. It is not necessary to add a combiner to your Map-Reduce program, it is optional.
- Combiner is also a class in our java program like Map and Reduce class that is used in between this Map and Reduce classes.
- Combiner helps us to produce abstract details or a summary of very large datasets. When we process or deal with very large datasets using Hadoop Combiner is very much necessary, resulting in the enhancement of overall performance.

2.5.1 How does combiner works

- In the above example, we can see that two Mappers are containing different data. the main text file is divided into two different Mappers. Each mapper is assigned to process a different line of our data. in our above example, we have two lines of data so we have two Mappers to handle each line.
- Mappers are producing the intermediate key-value pairs, where the name of the particular word is key and its count is its value. For example, for the data Geeks For Geeks For the key-value pairs are shown below.

// Key Value pairs generated for data

(Geeks,1)

(For,1)

(Geeks,1)

(For,1)

- The key-value pairs generated by the Mapper are known as the intermediate key-value pairs or intermediate output of the Mapper.
- Now we can minimize the number of these key-value pairs by introducing a combiner for each Mapper in our program. In our case, we have 4 key-value pairs generated by each of the Mapper. since these intermediate key-value pairs are not ready to directly feed to Reducer because that can increase Network congestion so Combiner will combine these intermediate key-value pairs before sending them to Reducer.
- The combiner combines these intermediate key-value pairs as per their key. For the above example for data for the combiner will partially reduce them by merging the same pairs according to their key value and generate new key-value pairs as shown below.

  // Partially reduced key-value pairs with combiner

  (Geeks,2)

  (For,2)
- With the help of Combiner, the Mapper output got partially reduced in terms of size(key-value pairs) which now can be made available to the Reducer for better performance.
- Now the Reducer will again Reduce the output obtained from combiners and produces the final output that is stored on **HDFS(Hadoop Distributed File System)**.

2.5.2 Advantage of combiners

1. Reduces the time taken for transferring the data from Mapper to Reducer.
2. Reduces the size of the intermediate output generated by the Mapper.
3. Improves performance by minimizing Network congestion.

2.5.3 Disadvantage of combiners

- The intermediate key-value pairs generated by Mappers are stored on Local Disk and combiners will run later on to partially reduce the output which results in expensive Disk Input-Output.
- The map-Reduce job cannot depend on the function of the combiner because there is no such guarantee in its execution.

2.6 MATRIX VECTOR MULTIPLICATIONBY MAPREDUCE

UQ. Write pseudo code for Matrix vector Multiplication by MapReduce. Illustrate with an example showing all thesteps. **(MU - May 19, 10 Marks)**

GQ. What happen when vector does not fit in memory in matrix vector multiplication? **(5 Marks)**

UQ. Write MapReduce pseudocode to multiply to matrices. Illustrate the procedure on the following matrices.Clearly show all the steps. **(MU - May 18, 10 Marks)**

- MapReduce is a technique in which a huge program is subdivided into small tasks and run parallelly to make computation faster, save time, and mostly used in distributed systems. It has 2 important parts:
- **Mapper:** It takes raw data input and organizes into key, value pairs. For example, In a dictionary, you search for the word "Data" and its associated meaning is "facts and statistics collected together for reference or analysis". Here the Key is Data and the Value associated with is facts and statistics collected together for reference or analysis.
- **Reducer:** It is responsible for processing data in parallel and produce final output.

Algorithm 1 : The map function

1. **For** each element m_{ij} of M **do**
2. product (key value) pair as ((i, k), (M, j, m_{ij})), for k = 1, 2,3,...up to the number of columns of N
3. for each element n_{jk} of N do
4. product (key value) pair as ((i, k), (N, j, n_{jk})), for i = 1, 2,3,...up to the number of rows of M.
5. **return** set of (key value) pairs that each key, (i, k), has a list with values (M, j, m_{ij}) and (N, j, n_{jk}) for all possible values of j

Algorithm 2 : The reduce function

1. For each key (i, k) **do**
2. sort value begin with M by j in $list_M$
3. sort value begin with N by j in $list_N$
4. multiply m_{ij} and n_{jk} for j_{th} value of each list
5. sum up $m_{ij} * m_{jk}$
6. **return** (i, k), $\sum_{j=1} m_{ij} * n_{jk}$

Let us consider the matrix multiplication example to visualize MapReduce

Consider the following matrix

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \times \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix}$$

Here matrix A is a 2×2 matrix which means the number of rows(i) = 2 and the number of columns(j)=2. Matrix B is also a 2×2 matrix where number of rows(j) = 2 and number of columns(k)=2. Each cell of the matrix is labelled as A_{ij} and B_{jk}. Ex. element 3 in matrix A is called A_{21} i.e. 2nd- row 1stcolumn. Now one step matrix multiplication has 1 mapper and 1 reducer. The Formula is:

Mapper for Matrix A (k, v) = ((i, k), (A, j, A_{ij})) for all k

Mapper for Matrix B (k, v) = ((i, k), (B, j, B_{jk})) for all i

Therefore, ecomputing the mapper for Matrix A:

k, i, j computes the number of times it occurs .

Here all are 2, therefore when k = l, i can have

2 values 1 & 2, each case can have 2 further

values of j = 1 and j = 2. Substituting all values

in formula

k = 1 i = 1 j = 1 ((1, 1), (A, 1, 1))

j = 2 ((1, 1), (A, 2, 2))

i = 2 j = 1 ((2, 1), (A, 1, 3))

j = 2 ((2, 1), (A, 2, 4))

k = 2 i = 1 j = 1 ((1, 2), (A, 1, 1))

j = 2 ((1, 2), (A, 2, 2))

i = 2 j = 1 ((2, 2), (A, 1, 3))

j = 2 ((2, 2), (A, 2, 4))

Matrix-Vector Multiplication by Map-Reduce

Computing the mapper for matrix B

i = 1 j = 1 k = 1 ((1, 1), (B, 1, 5))

k = 2 ((1, 2), (B, 1, 6))

j = 2 k = 1 ((1, 1), (B, 2, 7))

j = 2 ((1, 2), (B, 2, 8))

i = 2 j = 1 k = 1 ((2, 1), (B, 1, 5))

k = 2 ((2, 2), (B, 1, 6))

j = 2 k = 1 ((2, 1), (B, 2, 7))

k = 2 ((2, 2), (B, 2, 8))

- Matrix-Vector Multiplication by Map-Reduce

Reducer (k, v) = (i, k)=>Make sorted A_{list} and B_{list}

(i,k) => Summation (A_{ij} * B_{jk})) for J

Output =>((i k),sum)

Therefore, computing the reducer:

We can observe from Mapper computation

that 4 pairs are common (1, 1), (1, 2),

(2, 1) and (2, 2)

Make a list separate for Matrix A &

B with adjoining values taken from

Mapper step above :

Matrix-Vector Multiplication by Map-Reduce

(1, 1) =>A_{list}={(A, 1, 1), (A, 2, 2)}

B_{list}={(B, 1, 5), (B, 2, 7)}

Now $A_{ij} \times B_{jk}$: [(1 * 5) + (2*7)] = 19 ...(i)

(1, 2) =>A_{list}={(A, 1, 1), (A, 2, 2)}

B_{list}={(B, 1, 6), (B, 2, 8)}

Now $A_{ij} \times B_{jk}$: [(1*6) + (2*8)] :22 ...(ii)

(2, 1) =>A_{list}={(A, 1, 3), (A, 2, 4) }

B_{list}={(B, 1, 5), {B, 2, 7)}

Now $A_{ij} \times B_{jk}$: [(3* 5) + (4*7)] =43 ...(iii)

(2, 2) =>A_{list}={(A, 1, 3), (A, 2, 4)}

B_{list}={(B, 1, 6), (B, 2, 8)}

Now $A_{ij} \times B_{jk}$: [(3*6) + (4*8)] =50 ...(iv)

From (i), (ii), (iii) and (iv) we conclude that

((1, 1), 19)

((1, 2), 22)

((2, 1), 43)

((2, 2), 50)

Therefore, the final matrix is

$$\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

2.7 RELATIONAL ALGEBRA OPERATIONS

1. Selection.
2. Projection.
3. Union & Intersection
4. Natural Join.
5. Grouping & Aggregation.

1. Selection

- Apply a condition c to each taple in the relation and produce as output only those tuples that satisfy c.
- The result of this selection is denoted by 6c(R)6c(R)
- Selection really does not need the full power of MapReduce.
- They can be done most conveniently in the map portion alone, although they could also be done in the reduce portion also.
- The pseudo code is as follows :

```
Map (key, valve)
for tuple in valve :
if tuple satisfies C :
emit (tuple, tuple)
Reduce (key, valves)
emit (key, key)
```

2. Projection

- for some subset s of the attribute of the relation, produce from each tuple only the components for the attributes in S.
- The result of this projection is denoted TTs (R)
- Projection is performed similarly to selection.
- As projection may cause the same tuple to appear several times, the reduce function eliminate duplicates.

- The pseudo code for projection is as follows :

```
Map (key, valve)
for tuple in valve :
ts = tuple with only the components forthe attributes in S.
emit (ts, ts)
Reduce (key, values)
emit (key, key)
```

3. Union Using Map Reduce

- Both selection and projection are operations that are applied on a single table, whereas Union, intersection and difference are among the operations that are applied on two or more tables.
- Let's consider that schemas of the two tables are the same, and columns are also ordered in same order.
- **Map Function:** For each row r generate key-value pair (r, r) .
- **Reduce Function:** With each key there can be one or two values (As we don't have duplicate rows), in either case just output first value.

This operations has the **map function of the selection and reduce function of projection**. Let's see the working using an example. Here yellow colour represents one table and green colour is used to represent the other one stored at two map workers.

Map worker 1

Table 1	
A	**B**
1	2
2	3
5	6

Table 2	
A	**B**
2	3
4	4
6	1

Map worker 2

Table 1	
A	**B**
6	1
6	3
7	6

Table 2	
A	**B**
9	8
3	3
0	1

Initial data at map workers

After applying the map function and grouping the keys we will get output as:

Map worker 1

Key	Value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1)]

Map worker 2

Key	Value
(6, 1)	[(6, 1)]
(6, 3)	[(6, 3)]
(7, 6)	[(7, 6)]
(9, 8)	[(9, 8)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Map and grouping the keys

The data to be sent to reduce workers will look like:

Map worker 1

RW 1	
Key	**value**
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]

RW 2	
Key	**value**
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1)]

Map worker 2

RW1	
Key	**value**
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

RW2	
Key	**Value**
(6, 1)	[(6, 1)]
(7, 8)	[(7, 8)]
(9, 8)	[(9, 8)]

Files to be sent to reduce workersData at reduce workers after will be:

Reduce worker 1

RW 1	
Key	**value**
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]

RW 1	
Key	**value**
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Reduce worker 2

Key	Value
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1)]

Key	Value
(6, 1)	[(6, 1)]
(7, 8)	[(7, 8)]
(9, 8)	[(9, 8)]

Files At reduce workers

At reduce workers aggregation on keys will be done.

Reduce worker 1

Key	Value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Reduce worker 2

Key	Value
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1), (6, 1)]
(7, 6)	[(7, 6)]
(9, 8)	[(9, 8)]

Aggregated data at reduce workers

The final output after applying the reduce function which takes only the first value and ignores everything else is as follows:

Reduce worker 1

A	B
1	2
2	3
5	6
6	3
3	3
0	1

Reduce worker 2

A	B
4	4
6	1
7	6
9	8

Final table after union

Here we note that in this case same as projection we can this done without moving data around in case we are not interested in removing duplicates. And hence this operation is also efficient it terms of data shuffle across machines.

Intersection Using Map Reduce

For intersection, let's consider the same data we considered for union and just change the map and reduce functions

- **Map Function:** For each row r generate key-value pair (r, r) (Same as union).
- **Reduce Function:** With each key there can be one or two values (As we don't have duplicate rows), in case we have length of list as 2 we output first value else we output nothing.

As the map function is same as union and we are considering the same data lets skip to the part before reduce function is applied.

Reduce worker 1

Key	Value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Reduce worker 2

Key	Value
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1), (6, 1)]
(7, 6)	[(7, 6)]
(9, 8)	[(9, 8)]

Data at reduce workers

Now we just apply the reduce operation which will output only rows if list has a length of 2.

Reduce worker 1

File 1	
A	B
2	3

Reduce worker 2

File 1	
A	B
6	1

Output of intersection

Difference Using Map Reduce

Let's again consider the same data. The difficulty with difference arises with the fact that we want to output a row only if it exists in the first table but not the second one. So the reduce function needs to keep track on which tuple belongs to which relation. To

visualize that easier, we will keep those rows green which come from 2nd table and yellow for which come from 1st table and **purple** which comes from both tables.

- **Map Function:** For each row r create a key-value pair (r, T1) if row is from table 1 else product key-value pair (r, T2).
- **Reduce Function:** Output the row if and only if the value in the list is T1 , otherwise output nothing.

The data taken initially is the same as it was for union

Map worker 1

Table 1	
A	**B**
1	2
2	3
5	6

Table 2	
A	**B**
2	3
4	4
6	1

Map worker 2

Table 1	
A	**B**
6	1
6	3
7	6

Table 2	
A	**B**
9	8
3	3
0	1

Initial Data

After applying the map function and grouping the keys the data looks like the following figure

Reduce worker 1

Key	Value
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]
(4, 4)	[T2]
(6, 1)	[T2]

Reduce worker 2

Key	Value
(6, 3)	[T1]
(7, 6)	[T1]
(9, 8)	[T2]
(6, 1)	[T1]
(3, 3)	[T2]
(0, 1)	[T2]

Data after applying map function and grouping keys

After applying map function files for reduce workers will be created based on hashing keys as has been the case so far.

Reduce worker 1

RW1	
Key	**Value**
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]

RW2	
Key	**Value**
(4, 4)	[T2]
(6, 1)	[T2]

Reduce worker 2

RW1	
Key	**Value**
(6, 3)	[T1]
(7, 6)	[T1]
(9, 8)	[T2]

RW2	
Key	**Value**
(6, 1)	[T1]
(3, 3)	[T2]
(0, 1)	[T2]

Files for reduce workers

The data at the reduce workers will look like

Reduce worker 1

RW1	
Key	**Value**
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]

RW1	
Key	**Value**
(6, 3)	[T1]
(7, 6)	[T1]
(9, 8)	[T2]

Reduce worker 2

RW1	
Key	**Value**
(4, 4)	[T2]
(6, 1)	[T2]

RW1	
Key	**Value**
(6, 1)	[T1]
(3, 3)	[T2]
(0, 1)	[T2]

Files at reduce workers

After aggregation of the keys at reduce workers the data looks like:

Reduce worker 1

Key	Value
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]
(6, 3)	[T1]
(9, 8)	[T2]

Key	Value
(4, 4)	[T2]
(6, 1)	[T1, T2]
(3, 3)	[T2]
(0, 1)	[T2]
(7, 6)	[T1]

Data after aggregation of keys at reduce workers

The final output is generated after applying the reduce function over the output.

File 1	
A	B
1	2
5	6
6	3

File 1	
A	B
7	6

Output of difference of the tables

For the difference operation we notice that we cannot get rid of the reduce part and hence have to send data across the workers as the context of from which table the value came is needed. Hence it will be **more expensive** operation as compared to selection, projection, union and intersection.

Grouping and Aggregation Using Map Reduce

Usually understanding grouping and aggregation takes a bit of time when we learn SQL, but not in case when we understand these operations using map reduce. The logic is already there in the working of the map. Map workers implicitly group keys and the reduce function acts upon the aggregated values to generate output.

- **Map Function:** For each row in the table, take the attributes using which grouping is to be done as the key, and value will be the ones on which aggregation is to be performed. For example, If a relation has 4 columns A, B, C, D and we want to group by A, B and do an aggregation on C we will make (A, B) as the key and C as the value.
- **Reduce Function:** Apply the aggregation operation (sum, max, min, avg, ...) on the list of values and output the result.

For our example lets group by (A, B) and apply sum as the aggregation.

Map worker1							
File 1				File 2			
A	B	C	D	A	B	C	D
1	2	3	1	4	2	1	3
2	2	3	2	6	8	4	4
1	2	1	3	3	2	2	4

Map worker 2							
File 1				File 2			
A	B	C	D	A	B	C	D
1	2	5	2	3	2	5	2
2	3	2	4	2	3	9	2
1	3	1	3	3	4	2	1

Initial data at the map workers

The data after application of map function and grouping keys will creates (A, B) as key and C as value and D is discarded as if it doesn't exist.

Reduce worker 1	
Key	Value
(1, 2)	[3, 1]
(2, 2)	[3]
(4, 2)	[1]
(6, 8)	[4]
(3, 2)	[2]

Reduce worker 2	
Key	Value
(1, 2)	[5]
(2, 3)	[2, 9]
(1, 3)	[1]
(3, 2)	[1]
(3, 4)	[2]

Data at map workers

Applying partitioning using hash functions, we get

Files for the reduce workers

Map worker 1			
RW1		RW2	
Key	Value	Key	Value
(1, 2)	[3, 1]	(6, 8)	[4]
(2, 2)	[3]	(3, 2)	[2]
(4, 2)	[1]		

Map worker 2			
RW1		RW2	
Key	Value	Key	Value
(1, 2)	[5]	(3, 2)	[1]
(2, 3)	[2, 9]	(3, 4)	[2]
		(1, 3)	[1]

Files for reduce workers

The data at the reduce workers will look like

Reduce worker 1			
RW 1		RW1	
Key	Value	Key	Value
(1, 2)	(3, 1)	(1, 2)	[5]
(2, 2)	[3]	(2, 3)	[2, 9]
(4, 2)	[1]		

Reduce worker 1			
RW 2		RW 2	
Key	Value	Key	Value
(6, 8)	[4]	(3, 2)	[1]
(3, 2)	[2]	(3, 4)	[2]
		(1, 3)	[1]

Data at reduce workers

The data is aggregated based on keys before applying the aggregation function (sum in this case).

Reduce worker 1	
Key	Value
(1, 2)	[3, 1, 5]
(2, 2)	[3]
(4, 2)	[1]
(2, 3)	[(2, 9)]

Reduce worker 2	
Key	Value
(6, 8)	[4]
(3, 2)	[1, 2]
(3, 4)	[2]
(1, 3)	[1]

Aggregated data based on keys

After applying the sum over the value lists we get the final output

Reduce worker 1		
A	B	Sum
1	2	9
2	2	3
4	2	1
2	3	11

Reduce worker 2		
A	B	Sum
6	8	4
3	2	3
3	4	2
1	3	1

Output of group by (A, B) sum(C)

Here also like difference operation we can't get rid of the reduce stage. The context of tables isn't wanted here but the aggregation function makes it necessary for the values to be in one place for a single key. This operation is also inefficient as compared to selection, projection, union, and intersection. The column that is not in aggregation or grouping clause is ignored and isn't required. So, if the data be stored in a columnar format, we can save cost of loading a lot of data. Usually there are only a few columns involved in grouping and aggregation it does save up a lot of cost both in terms of data that is sent over the network and the data that needs to be loaded to main memory for execution.

2.8 NATURAL JOIN USING MAP REDUCE

The natural join will keep the rows that matches the values in the common column for both tables. To perform natural join, we will have to keep track of from which table the value came from. If the values for the same key are from different tables we need to form pairs of those values along with key to get a single row of the output. Join can explode the number of rows as we have to form each and every possible combination of the values for both tables.

- **Map Function:** For two relations Table 1(A, B) and Table 2(B, C) the map function will create key-value pairs of form b: [(T1, a)] for table 1 where T1 represents the fact that the value a came from table 1, for table 2 key-value pairs will be of the form b: [(T2, c)].
- **Reduce Function:** For a given key b construct all possible combinations for the values where one value is from table T1 and the other value is from table T2. The output will consist of key-value pairs of form b: [(a, c)] which represent one row a, b, c for the output table.

For an example let's consider joining Table 1 and Table 2, where B is the commo column.

Map worker 1			
Table 1		Table 2	
A	B	B	C
1	2	2	3
2	3	4	4
5	6	6	1

Map worker 2			
Table 1		Table 2	
A	B	B	C
6	1	9	8
6	3	3	4
7	6	2	1

Initial data at map workers

The data after applying the map function and grouping at the map workers will loo like:

Map worker 1	
Key	Value
2	[T1, 1), (T2, 3)]
3	[T(1, 2)]
6	[(T1, 5), (T2, 1)]
4	[(T1, 4)]

Map worker 2	
Key	Value
1	[T1, 6)
3	[T1, 6), (T2, 4)]
6	[(T1, 7)
9	[(T2, 8)]
2	[(T2, 1)]

Data at map workers after applying map function and grouping the keys

As has been the case so far files for reduce workers will be created at the map worker

Map worker 1			
RW 1		RW 2	
Key	Value	Key	Value
2	[(T1, 1), (T2, 3), (T2, 1)]	6	[(T1, 5), (T2, 1)]
3	[(T1, 2)]	4	[(T1, 4)]

Map worker 2			
RW 1		RW 2	
Key	Value	Key	Value
1	[(T1, 6)	6	[(T1, 7)]
3	[(T1, 6), (T2, 4)]	9	[(T2, 8)]

Files constructed for reduce workers

The data at the reduce workers will be:

Reduce worker 1				Reduce worker 1			
RW1		RW1		RW2		RW2	
Key	Value	Key	Value	Key	Value	Key	Value
2	[(T1, 1), (T2, 3), (T2, 1)]	1	[(T1, 6)]	6	[(T1, 5), (T2, 1)]	6	[(T1, 7)]
3	[(T1, 2)]	3	[(T1, 6), (T2, 4)]	4	[(T1, 4)]	9	[(T2, 8)]

Data at reduce workers

Applying aggregation of keys at the reduce workers we get:

Reduce workers 1			Reduce workers 2	
Key	value		Key	value
2	[(T1, 1), (T2, 3), (T2, 1)]		6	[(T1, 5), (T1, 7), (T2, 1)]
3	[(T1, 2), (T1, 6), (T2, 4)]		4	[(T1, 4)
1	[(T1, 6)]		9	[(T2, 8)]

Data after aggregation of keys at the reduce workers

After applying the reduce function which will create a row by taking one value from table T1 and other one from T2. If there are only values from T1 or T2 in the values list that won't constitute a row in output.

Reduce worker 1		
B	A	C
2	1	3
2	1	1
3	2	4
3	6	4

Reduce worker 1		
B	A	C
6	5	1
6	7	1

Output of the join

- As we need to keep context from which table a value came from, we can't get rid of the data that needs to be sent across the workers for application of reduce task, this operation also becomes costly as compared to others we discussed so far.
- The fact that for each list of values we need to create pairs also plays a major factor in the computation cost associated with this operation.

2.9 MAPREDUCE – UNDERSTANDING WITH REAL-LIFE EXAMPLE

MapReduce is a programming model used to perform distributed processing in parallel in a Hadoop cluster, which Makes Hadoop working so fast. When you are dealing with Big Data, serial processing is no more of any use. MapReduce has mainly two tasks which are divided phase-wise:

- Map Task
- Reduce Task

Let us understand it with a real-time example, and the example helps you understand Mapreduce Programming Model in a story manner:

- Suppose the Indian government has assigned you the task to count the population of India. You can demand all the resources you want, but you have to do this task in 4 months. Calculating the population of such a large country is not an easy task for a single person(you). So what will be your approach?.
- One of the ways to solve this problem is to divide the country by states and assign individual in-charge to each state to count the population of that state.
- Task Of Each Individual: Each Individual has to visit every home present in the state and need to keep a record of each house members as:

  State_Name Member_House1

  State_Name Member_House2

  State_Name Member_House3

  State_NameMember_House n

  .For Simplicity, we have taken only three states.

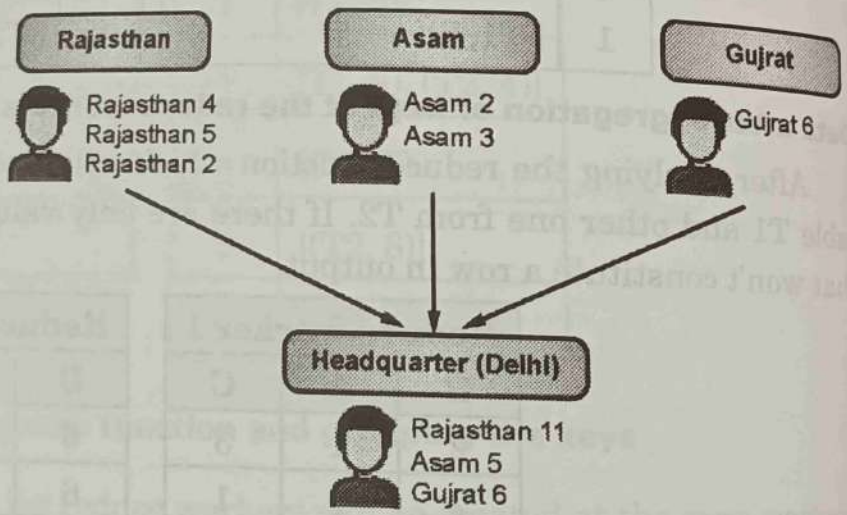


Fig. 2.9.1

- This is a simple Divide and Conquer approach and will be followed by each individualto count people in his/her state.
- Once they have counted each house member in their respective state. Now they need to sum up their results and need to send it to the Head-quarter at New Delhi.
- We have a trained officer at the Head-quarter to receive all the results from each state and aggregate them by each state to get the population of that entire state. and

Now, with this approach, you are easily able to count the population of India by summing up the results obtained at Head-quarter.

- The Indian Govt. is happy with your work and the next year they asked you to do the same job in 2 months instead of 4 months. Again you will be provided with all the resources you want.
- Since the Govt. has provided you with all the resources, you will simply double the number of assigned individual in-charge for each state from one to two. For that divide each state in 2 division and assigned different in-charge for these two divisions as:
  - State_Name_Incharge_division1
  - State_Name_Incharge_division2
- Similarly, each individual in charge of its division will gather the information about members from each house and keep its record.
- We can also do the same thing at the Head-quarters, so let's also divide the Head-quarter in two division as:
  - Head-qurter_Division1
  - Head-qurter_Division2

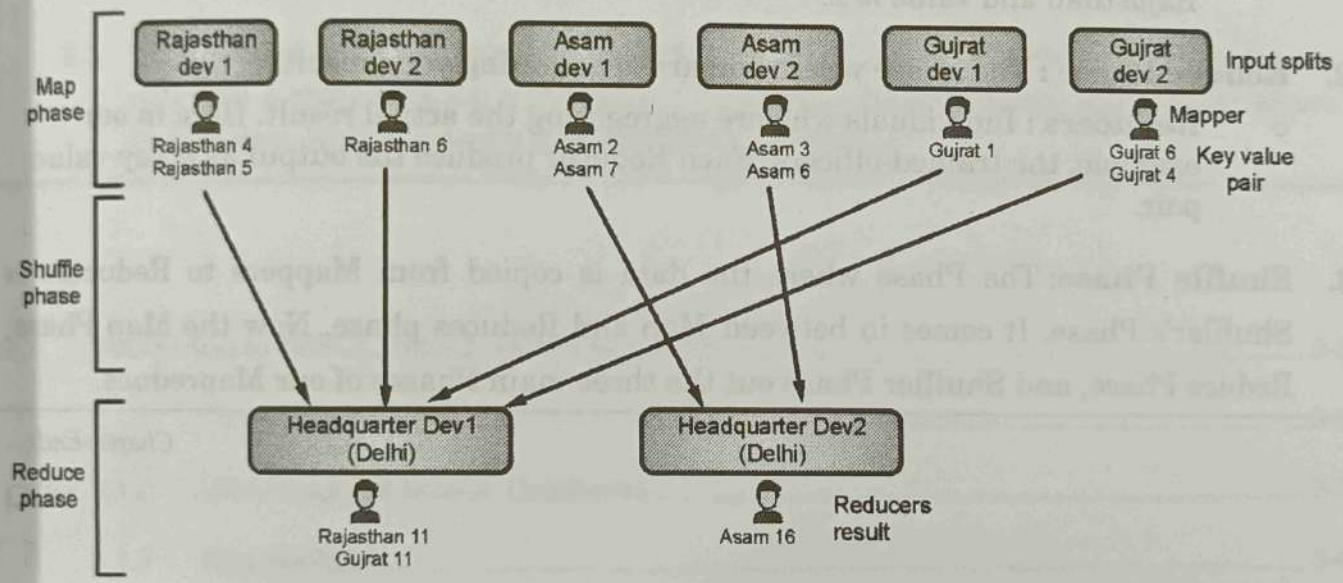


Fig. 2.9.2

- Now with this approach, you can find the population of India in two months. But there is a small problem with this, we never want the divisions of the same state to send their result at different Head-quarters then, in that case, we have the partial population of that state in Head-quarter_Division1 and Head-quarter_Division2 which is inconsistent because we want consolidated population by the state, not the partial counting.
- One easy way to solve is that we can instruct all individuals of a state to either send

there result to Head-quarter_Division1 or Head-quarter_Division2. Similarly, for all the states.

- Our problem has been solved, and you successfully did it in two months.
- Now, if they ask you to do this process in a month, you know how to approach the solution.
- Great, now we have a good scalable model that works so well. The model we have seen in this example is like the MapReduce Programming model. so now you must be aware that MapReduce is a programming language.

Now let's discuss the phases and important things involved in our model.

1. **Map Phase:** The Phase where the individual in-charges are collecting the population of each house in their division is Map Phase.
   - **Mapper :**Involved individual in-charge for calculating population.
   - **Input Splits :** The state or the division of the state.
   - **Key-Value Pair :** Output from each individual Mapper like the key is Rajasthan and value is 2.
2. **Reduce Phase :** The Phase where you are aggregating your result.
   - ***Reducers :*** *Individuals who are aggregating the actual result. Here in our example, the trained-officers. Each Reducer produce the output as a key-value pair.*
3. **Shuffle Phase:** The Phase where the data is copied from Mappers to Reducers is Shuffler's Phase. It comes in between Map and Reduces phase. Now the Map Phase, Reduce Phase, and Shuffler Phase out the three main Phases of our Mapreduce.

ChapterEnds...

❑❑❑

MODULE III

CHAPTER 3

NoSQL

University Prescribed Syllabus w.e.f AcademicYear 2022-2023

3.1 Introduction to NoSQL, NoSQL Business Drivers

3.2 NoSQL Data Architecture Patterns: Key-value stores, Graph stores, Column family (Bigtable)stores, Document stores, Variations of NoSQL architectural patterns, NoSQL Case Study

3.3 NoSQL solution for big data, Understanding the types of big data problems; Analyzing big data with a shared-nothing architecture; Choosing distribution models: master-slave versus peer-to-peer; NoSQL systems to handle big data problems.

3.1 INTRODUCTION TO NoSQL, NoSQL BUSINESS DRIVERS

3.1.1 Introduction to NoSQL

A database is a systematic collection of data. And a database management system supports storage and manipulation of data which makes data management easy. For example, an online telephone directory uses a database to store data of people like phone numbers and other contact details that can be used by service provider to manage billing client related issues and handle fall data etc. That means A database management system provides the mechanism to store and retrieve the data.

There are different kinds of management systems:

RDBMS	OLAP	NoSQL
(Relational Database Management System)	(Online Analytical Processing)	(Not only SQL)

NoSQL refers to all databases and data stores that are not based on the relational database management system or RDBMS principles. NoSQL are the new set of databases that has emerged recent past as an alternative solution to relational databases.

Carl Strozzi introduced the term NoSQL to name his file-based database in 1998.

NoSQL does not represent single product or technology but it represents a group products and various related data concepts for storage and management. NoSQL is an approach to database management that can accommodate a wide variety of data models including key-value, document, column and graph formats. NoSQL database generally means that it is non-relational, distributed, flexible and scalable.So, we can bind it up as-

- NoSQL an approach to database design that provides flexible schemas for the storage and retrieval of data beyond the traditional table structures found in relational databases.
- It relates to large data sets accessed and manipulated on a Web scale.

3.1.2 Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database.
- 2000- Graph database Neo4j is launched.

- 2004- Google BigTable is launched.
- 2005- CouchDB is launched.
- 2007- The research paper on Amazon Dynamo is released.
- 2008- Facebooks open sources the Cassandra project.
- 2009- The term NoSQL was reintroduced.

3.1.3 Why NoSQL?

The concept of NoSQL databases became popular with internet giants like Google, Facebook, Amazon etc. Who deals with huge volumes of data the system response time becomes slow when we use RDBMS for massive volumes of data so to resolve this problem, we could scale up our system by upgrading our existing hardware but this process is an expensive. So alternative for this issue is to distribute database load on multiple hosts whenever the load increases this method known as scaling out.

NoSQL databases are non-relational so they scale-out better than relational databases. As they designed with the web applications in mind. Now NoSQL database is exactly type pf database that can handle all sorts of semi structured data, unstructured data, rapidly changing data and bigdata. So, to resolve the problems related to large volume and semi structured data. NoSQL databases have emerged.

3.1.4 CAP Theorem

GQ. What is CAP Theorem? How it is applicable to NOSQL systems? **(4 Marks)**

It plays important role in NoSQL databases. CAP theorem is also called brewer's theorem which states that it is impossible for a distributed data store to offer more than two out of three guarantees:

So basically, some NoSQL databases offer consistency and partition tolerance. While some offer availability and partition tolerance. But partition tolerance is common as NoSQL databases are distributed in nature so based on requirement, we can choose NoSQL database has to be used. Different types of NoSQL databases are available based on data models.

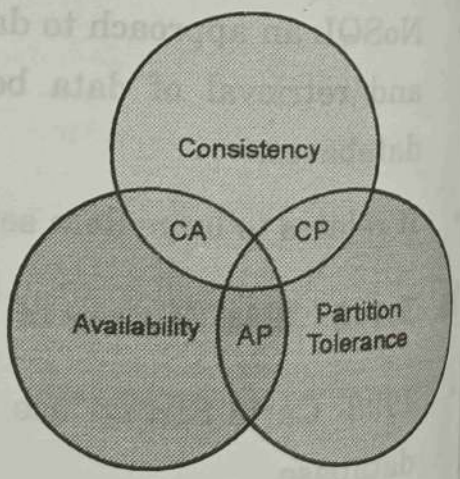


Fig 3.1.1 :CAP Property

Consistency

- This means that the data in the database remains consistent after the execution of an operation.
- For example, after an update operation all clients see the same data.

Availability

- This means that the system is always on (service guarantee availability), no downtime.

Partition Tolerance

- This means that the system continues to function even the communication among the servers is unreliable, i.e., the servers may be partitioned into multiple groups that cannot communicate with one another.

In theoretically it is impossible to fulfil all 3 requirements. CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore, all the current NoSQL database follow the different combinations of the C, A, P from the CAP theorem.

Here is the brief description of **three combinations CA, CP, AP** :

- CA - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.
- CP -Some data may not be accessible, but the rest is still consistent/accurate.
- AP - System is still available under partitioning, but some of the data returned may be inaccurate.

The use of the word consistency in CAP and its use in ACID do not refer to the same identical concept.

In CAP, the term consistency refers to the consistency of the values in different copies of the same data item in a replicated distributed system. In ACID, it refers to the fact that a transaction will not violate the integrity constraints specified on the database schema

3.1.5 Characteristics / Features of NoSQL

UQ. Describe characteristics of a NoSQL database. **MU- Dec 17, 10 Marks**

1. Non-relational

- NoSQL databases never follow the relational model

Consistency

- This means that the data in the database remains consistent after the execution of an operation.
- For example, after an update operation all clients see the same data.

Availability

- This means that the system is always on (service guarantee availability), no downtime.

Partition Tolerance

- This means that the system continues to function even the communication among the servers is unreliable, i.e., the servers may be partitioned into multiple groups that cannot communicate with one another.

In theoretically it is impossible to fulfil all 3 requirements. CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore, all the current NoSQL database follow the different combinations of the C, A, P from the CAP theorem.

Here is the brief description of **three combinations CA, CP, AP** :

- CA - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.
- CP -Some data may not be accessible, but the rest is still consistent/accurate.
- AP - System is still available under partitioning, but some of the data returned may be inaccurate.

The use of the word consistency in CAP and its use in ACID do not refer to the same identical concept.

In CAP, the term consistency refers to the consistency of the values in different copies of the same data item in a replicated distributed system. In ACID, it refers to the fact that a transaction will not violate the integrity constraints specified on the database schema

3.1.5 Characteristics / Features of NoSQL

UQ. Describe characteristics of a NoSQL database. **MU- Dec 17, 10 Marks**

1. Non-relational

- NoSQL databases never follow the relational model

- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID

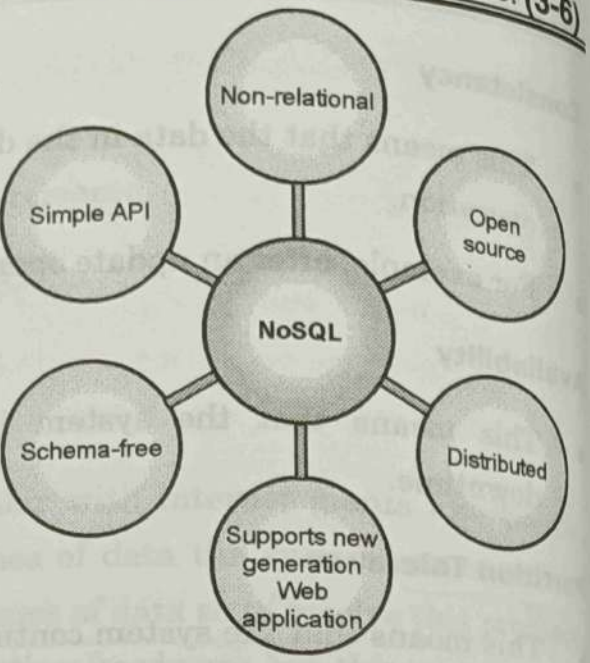


Fig. 3.1.2

2. Open-source

NoSQL databases don't require expensive licensing fees and can run on inexpensive hardware, rendering their deployment cost-effective.

3. Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain

4. Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based query language
- Web-enabled databases running as internet-facing services

5. Distributed

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Often ACID concept can be sacrificed for scalability and throughput
- Mostly no synchronous replication between distributed nodes Asynchronous MultiMaster Replication, peer-to-peer, HDFS Replication
- Only providing eventual consistency

- Shared Nothing Architecture. This enables less coordination and higher distribution.

3.1.6 Advantages and Disadvantagesof NoSQL

Advantages of NoSQL	Disadvantages of NoSQL
1. Scale(horizontal) 2. SQL databases are vertically scalable. This means that you can increase the load on a single server by increasing things like RAM, CPU or SSD. But on the other hand, NoSQL databases are horizontally scalable. This means that you handle more traffic by sharding, or adding more servers in your NoSQL database 3. Simple data model (fewer joins) 4. Streaming/ volume 5. Reliability 6. Schema-less (no modelling or prototyping) 7. Rapid development 8. Flexible as it can handle semi-structured, unstructured and structured data. 9. Cheaper than relational database 10. Creates a caching layer 11. Wide data type variety 12. Uses large binary objects for storing large data 13. Bulk upload 14. Lower administration 15. Distributed storage 16. Real-time analysis	1. ACID transactions 2. Cannot use SQL 3. Cannot perform searches 4. Data loss 5. No referential integrity 6. Lack of availability of expertise

3.1.7 Difference between RDBMS and NoSQL

UQ. Differentiate between a RDBMS and NoSQL database. **MU- Dec-18, 10 Marks**

Sr. No.	RDBMS	NoSQL
1.	Have fixed or static or predefined schema	Have dynamic schema
2.	Not suited for hierarchical data storage	Best suited for hierarchical data storage
3.	Vertically scalable (Scale - up)	Horizontally scalable (Scale-out)
4.	Follow ACID property	Follows CAP (consistency, availability, partition tolerance)
5.	Relational Database supports transactions (also complex transactions with joins).	NoSQL databases don't support transactions (support only simple transactions).
6.	Relational database manages only structured data.	NoSQL database can manage structured, unstructured and semi-structured data.
7.	Relational databases have a single point of failure with failover.	NoSQL databases have no single point of failure.
8.	Relational Database supports a powerful query language.	NoSQL Database supports a very simple query language.
9.	It gives only read scalability.	It gives both read and write scalability.
10.	Transactions written in one location.	Transactions written in many locations.
11.	It supports complex transactions.	It supports simple transactions.
12.	It is used to handle data coming in low velocity.	It is used to handle data coming in high velocity.
13.	Examples- MySQL, Oracle, Sqlite, PostgreSQL and MS-SQL etc.	Examples- MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc.

3.1.8 NoSQL Business Drivers

The scientist-philosopher Thomas Kuhn coined the term paradigm shift to identify a recurring process. He observed that in science, where innovative ideas came in bursts and impacted the world in nonlinear ways. We'll use Kuhn's concept of the paradigm shift as a way to think about and explain the NoSQL movement and the changes in thought patterns, architectures, and methods emerging today.

Many organizations are supporting to the single-CPU relational systems that have fulfilled the needs of theirorganizations as per the requirements.

Businesses have initiated the value in fast catching and examining huge quantityof adjustable data and making direct changes in their businesses based on the data that they obtain. As all of thesedrivers applies burden on single-processor relational model, its basis suits less steady and in time no extendedencounters the organization's needs.

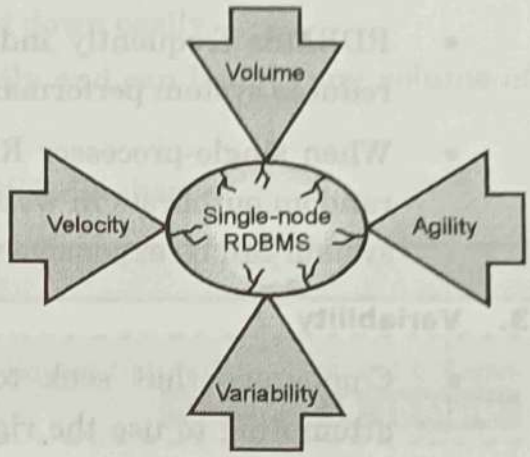


Fig. 3.1.3

Fig. 3.1.3 In this, we see how the business drivers volume, speed, variability, and agility create pressure on a single CPU system, resulting in cracks. Volume and velocity state to the capability to handle and managethe big datasets that appears early. Variability states to how various data types do not fit within the structured tables,and agility states to how much fast an organization replies to the business modification.

There are 4 major business drivers for NoSQL as:

(1) Volume (2) Velocity (3) Variability (4) Agility.

1. Volume

- Undoubtedly, the main factor forcing organizations to look at alternatives to their existing RDBMS is to investigate big data using clusters of commodity processors.
- Until around 2005, performance problems were eliminated by purchasing faster processors. Over time, the ability to speed up the process is no longer an option.
- As the chip density increases, the heat can no longer dissolve rapidly enough without chip overheating. This phenomenon, known as the power wall, forced system designers to shift their focus from increasing speeds on a single chip to using more processors working together.
- The need to scale out (also known as horizontal scaling), rather than scale up (faster processors), moved organizations from serial to parallel processing where data problems are split into separate paths and sent to separate processors to divide and conquer the work.

2. Velocity

- While large data issues are considered for many organizations moving away from RDBMSs, the ability of a single processor system to read and write data quickly is also important.
- Many single-processor RDBMS cannot meet the demand for online queries on databases created by real-time insert and public-facing websites.
- RDBMSs frequently index multiple columns of each new row, a process that reduces system performance.
- When single-processor RDBMSs are used as a back-end in front of a web store, random outbursts in web traffic reduce the response for everyone, and tuning the system can be expensive when both high read and write throughput is required.

3. Variability

- Companies that seek to capture and report exceptional data conflicts when attempting to use the rigorous database schema structure imposed by RDBMSs. For example, if a business unit wants to capture some custom fields for a specific customer, it must store this information even if it does not apply to all customer rows in the database.
- Adding new columns to RDBMS requires shutting down the system and running the ALTER TABLE command. When the database is large, this process can affect the availability of the system, costing time and money.

4. Agility

- Agility is ability to accept change easily and quickly.
- Among the variety of agility dimensions such as model agility (ease and speed of changing data modules), operational agility (ease models), operational agility (ease and speed of changing operational aspects), and programming agility (ease and speed of application development) – one of the most important is the ability to quickly and seamlessly scale an application to accommodate large amounts of data, users and connections.
- The most complex part of building applications using RDBMS is the process of putting data into and getting data out of the database.
- If your data has nested and repeated subgroups of data structures, you need to include an object-relational mapping layer.
- The responsibility of this layer is to generate the correct combination of Insert, update, delete and select SQL statement to move object data to and from the RDBMS persistence layer.

- This process is not simple and is associated with the largest barrier to rapid change when developing new or modifying existing applications.
- Generally, object relational mapping requires experienced software developers.
- Even with experienced staff, small change requests can cause slowdowns in development and testing schedules.
- All these hurdles are best overcome by NOSQL database.
- These databases are schematic and can be scaled down easily.
- They can accommodate application changes easily and can handle any volume of data efficiently.
- This agility has become business driver for NOSQL databases.

3.2 NoSQL DATA ARCHITECTURE PATTERNS

UQ. What are the different architectural patterns in NoSQL? Explain Graph data store and Column Family Store patterns with relevant examples. **(MU - May 19, 10 Marks)**

NoSQL databases were born out of the rigidity of traditional relational or SQL databases, which use tables, columns, and rows to establish relationships across data. Developers welcomed NoSQL databases because they didn't require an upfront schema design; they were able to go straight to development. And it's this flexibility, this "ad-hoc" approach to organizing data, that has arguably been NoSQL's greatest selling point, which continues to appeal to organizations that need to store, retrieve, and analyze either unstructured or rapidly changing data.

The data stored in NoSQL follows any of the four data architecture patterns.

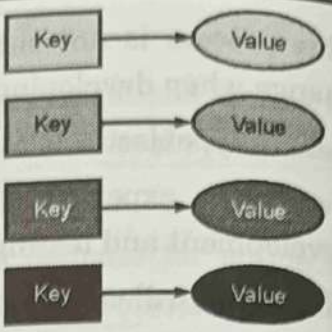
(A)	Key-Value Stores	(B)	Column family (Bigtable)Stores
(C)	Document Stores	(D)	Graph Stores

3.2.1 Key-Value Stores

UQ. Explain in detail key-value store NoSQL architectural pattern. Identify two applications that can use thispattern. **(MU - May 18, 5 Marks)**

- One of the most basic NoSQL database models is this model. The data is collected in the pattern of Key-Value Pairs, as the name implies. A series of strings, integers or characters is typically the key, but it can also be a more advanced form of data.
- Usually, the value is connected or co-related to the key. The databases for key-value pair storage typically store information as a hash table where each key is unique.

- The value may be of any form (JavaScript Object Notation (JSON), Binary Large Object (BLOB), strings, etc.).
- **Application :** This style of architecture is commonly used in shopping websites or e-commerce applications and its important assets is its ability for wide management of data volumes, heavy loads and the ease with which keys are used to retrieve data.



Key	Value
user-123	"John Doe"
image-123.jpg	<binary image file>
http://webpage-123.html	<web page html>
file-123.pdf	<pdf document>

Fig 3.2.1: An example of Key-Value

Keys and values are flexible. Keys can be image names, web page URLs, or file path names that point to values like binary images, HTML web pages, and PDF documents

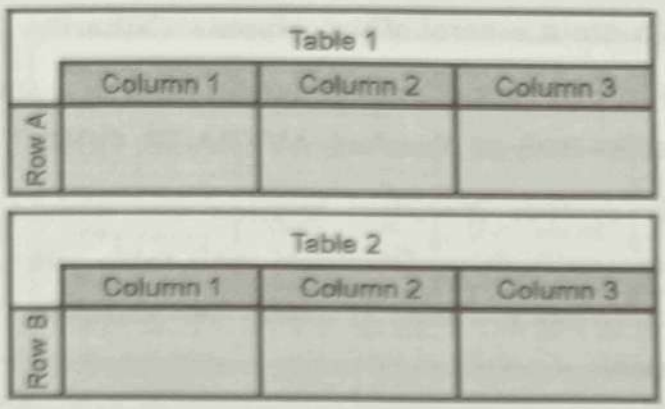
Constraints associated with the key-value store databases is its complexity in handling queries which will attempt to include many key-value pairs that may delay output and may cause data to clash with many-to-many relationships.

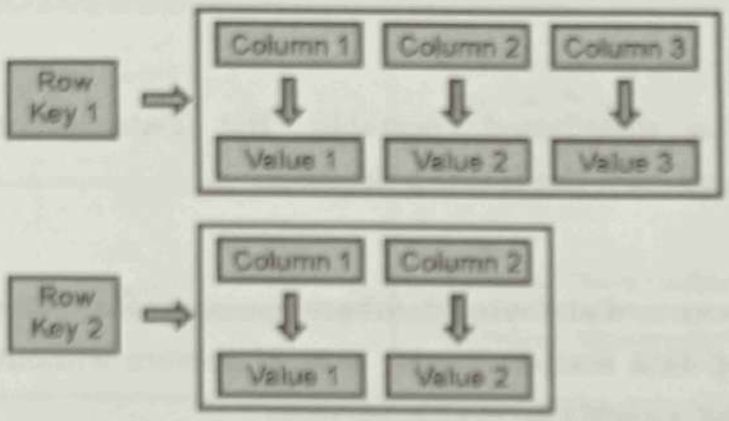
GQ. State example of any two key value databases **(2 Marks)**

Examples here are:

- DynamoDB (developed by Amazon)
- Berkeley DB (developed by Oracle)
- **REDIS:** An advanced open-source key-value store, also referred to as a data structure server because keys can include strings, hashes, lists, sets and sorted sets. This product, written in C/C++, is searingly quick, which makes it perfect for data collection in real time.
- **Riak:** An open source that is powerful, distributed database that predictably scalescapability and simplifies creation by prototyping, developing, and deployingapplications quickly. Written in Erlang and C this technology gives transparent fault-tolerant/fail-over functionality, a comprehensive and versatile API perfect for point-of-sale and factory control systems.
- **VoltDB:** scalable database in memory that offers complete transactional ACID consistency and ultra-high throughput, self-referred to as the NewSQL. This technology relies on segmentation and replication to achieve high-availability data snapshots and durable command logging using Java stored processes (for crash recovery), making it ideal for capital markets, digital networks, network services, and for online gaming.

3.2.2 Column Store Database





	company				super column family
	name	address		website	
1	DataX	city	San Francisco	protocol	https
		state	California	domain	datax.com
		street num	135	subdomain	www
		street	Kearny St		column
2	Process-One	city	Arlington	protocol	https
		state	Virginia	domain	process1.com
		street num	3500	subdomain	www
		street	Wilson St		

row key column family

Fig 3.2.2 : An Example of Column Store

- This pattern employs data storage in individual cells that is further divided into columns, rather than storing data in relational tuples.
- Databases thatare column-oriented operate only on columns. They together store vast quantities of data in columns. The column format and titles will diverge fromone rowto another.

- Each column is handled differently, but still, like conventional databases, each individual column will contain several other columns. (Niharika ,2020)
- Basically, columns are in this sort of storage mode. Data is readily available and it is possible to perform queries such as Number, AVERAGE, COUNT on columns easily.

The setbacks for this system includes: transactions should be avoided or not supported, queries can decrease high performance with table joins, record updates and deletes reduce storage efficiency, and it can be difficult to design efficient partitioning/indexing schemes.

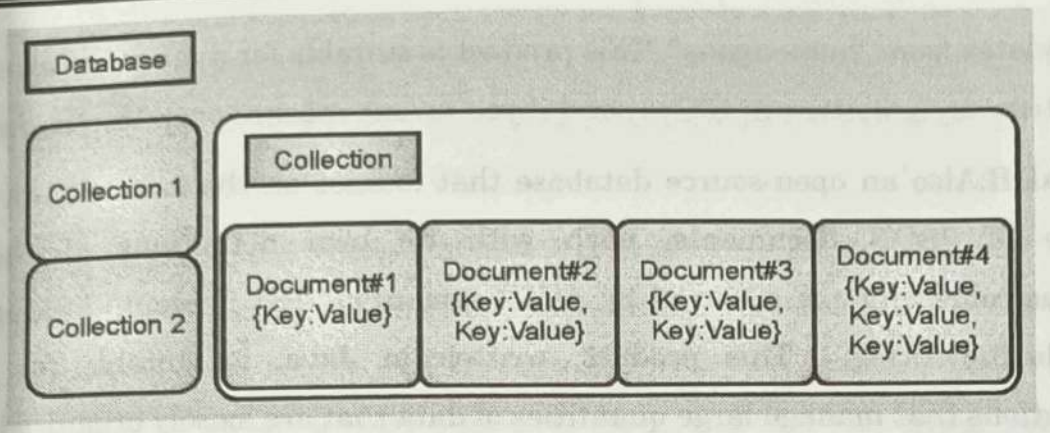
GQ. State example of any two column store databases **(2 Marks)**

Examples here are:

- **HBase:** HBase is a distributed, portable, Big Data Store modelled after Google's BigTable technology, the Hadoop database.
- Google's BigTable
- Cassandra: An open-source distributed database management system built to manage very large volumes of data scattered over several servers without a single point of failure while delivering a highly accessible service.
- Written in Java, this product is best for non-transactional real-time data analysis with linear scalability and proven fault-tolerance combined with column indexes.

3.2.3 Document Database

- In the form of key-value pairs, the record database fetches and accumulates information, but here the values are called documents. A complicated data structure can be represented as a text.
- It is hierarchical version of key-value databases.
- The document can be in text form, arrays, strings, JSON (JavaScript Object Notation), XML (Extensible Markup Language) or any other format.
- The use of nested documents is immensely popular. It is highly efficient since most of the generated information is generally in the form of JSONs and is unstructured.



Key	Document
document-1	{ "id":"1", "name":"John Smith", "isactive":"true", "birthdate":"08/30/1984" }
document-2	{ "id":"2", "fullname":"Sara Walker", "isactive":"false", "birthdate":"02/15/1971" }
document-3	{ "id":"3", "fullname": { "firstname":"Max", "lastname":"Johnson" "middleinitial":"B" } "isactive":"true", "birthdate":"04/02/1964" }

Fig 3.2.3 : An Example of Document

This format is extremely useful and appropriate for semi-structured data, and it is simple to retrieve and handle documents from storage. The drawbacks associated with this system includes the challenging factor of handling multiple documents and the inaccurate working of aggregation operations.

GQ. State example of any two document store databases **(2 Marks)**

Examples of such databases are:

- **MongoDB:** This scalable, high-performance, open-source NoSQL database features document-oriented (JSON-like) storage, full index support, replication, and fast on-

site updates from "humongous". This product is suitable for dynamic queries, dynamic data structures, written in C/C++, and if you favour indexes over Map/Reduce.

- **CouchDB:**Also an open-source database that focuses on the ease of data storage in aseries of JSON documents, each with its own definitions of the schema. Eventualconsistency is enforced by ACID semantics that prevents locking database files duringwriting. This product, written in Java, is suitable for web-based applications that manage large quantities of data that are loosely organized.

3.2.4 Graph Database

GQ. Write a short note on Graph Databases. **(4Marks)**

- This pattern of architecture clearly deals with information storage and management in graphs.
- Graphs are essentially structures that represent relations between two or more objects in some data.
- Objects or entities are referred to as nodes and are connected with relationships known as edges. There is a unique identifier on each edge.

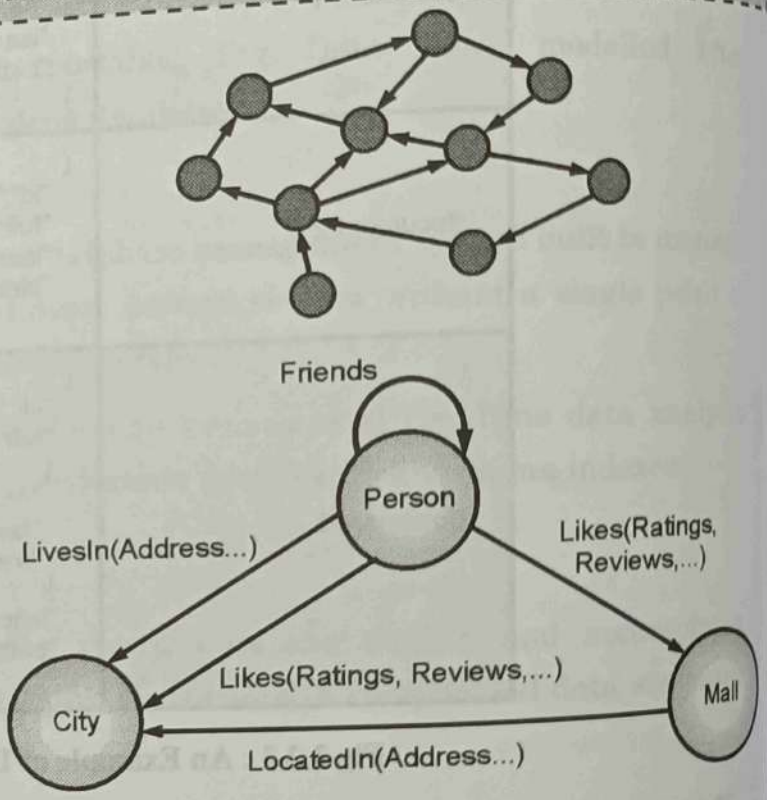


Fig 3.2.4 : An Example of Graph

- For the graph, each node serves as a point of touch. In social networks where there are many and large numbers of entities, this pattern is very widely used and each entity has one or many characteristics that are linked by edges.

There are loosely connected tables in the relational database pattern, whereas graphs are often strong and rigid in nature, have a faster traversal due to connections, and allow spatial data to be easily handled, but incorrect connections can lead to infinite loops. (Jan. 2016)

GQ. State example of any two graph databases **(2 Marks)**

Examples of such databases are:

- **Neo4J:** The leading native graph database and graph platform is Neo4J: Neo4j. For enterprise levels of security and high performance and reliability by clustering, it is available both as open source and through a commercial license. Cypher, the graph query language of Neo4j, is very simple to learn and can use newly released open source toolkits, "Cypher on Apache Spark (CApS) and Cypher for Gremlin to operate across Neo4j, Apache Spark and Gremlin-based products."
- **FlockDB (Used by Twitter):** FlockDB is easier than other graph databases since it attempts to solve less problems. It fits horizontally and is optimized for on-line, low-latency, high throughput environments such as websites.
- **ArangoDB:** this type of graph database requires one database, One Query language, Three models for data. The Limitless Possibilities. ArangoDB is a fast-growing native multi-model NoSQL database, with more than one million downloads.
- **OrientDB:** OrientDB is the first Distributed DBMS multi-model with a True Graph Driver. Multi-Model means NoSQL 2nd generation that is capable of managing complex domains with amazing efficiency.
- **Titan:** Titan is a scalable graph database designed for storing and querying graphs spread over a multi-machine cluster comprising hundreds of billions of vertices and edges. Titan is a transactional database that can facilitate the real-time execution of complex graph traversals by thousands of concurrent users.
- **DataStax:** In a rapidly changing environment where aspirations are strong, DataStax helps businesses thrive and new technologies occur daily.
- **Amazon Neptune:** Amazon Neptune with a highly connected datasets to create and run applications is secure, fast and has a fully managed graph database service that is easy. (Niharika, 2020).

NoSQL databases features

Types Features	Key-value store	Document store	Column-oriented store	Graph store
Characteristics	A simple hash tableindexed by key	Multiple key/value pairsform a document. Document stored generally in JSON format.	Store data in columnar format. Each key is associated with multiple attributes.	Focused on modelling the structure of the data.
Pros	Very fast and scalable. Simple model.	Schema free: Unstructured data can be stored easily. Simple, powerful data model. Horizontal scalability.	Better for complex read queries. Fast querying of data. Storage of very large quantities of data. Better analytic performance. Improved data compression.	Powerful data model. Locally connected data. Indexed data. Easy to query. Handling complex relational information.
Cons	Stored data have no schema. Poor for complex data. All joins must be done in code. No foreign key constraints. No triggers.	Query model limited to keys and indexes. No standard query syntax. Map Reduce for larger queries. Poor for interconnected data.	Very low-level API. Undefined data usage pattern. Increased disk seek time. Increased cost of inserts. Poor for interconnected data.	Travers entire graph to give correct results. Sharding.

Types Features	Key-value store	Document store	Column-oriented store	Graph store
Suitable for	Storing session's information, user profiles, preferences, shopping cart data.	Content management systems, blogging platforms, web analytics, real-time analytic's, e-commerce applications.	Content management systems, blogging platforms, maintaining counters, expiring usage, heavy write volume such as log aggregation	Space problem and connected data, such as social networks, spatial data, routing information for goods and money, recommendation engines.
Examples	Riak Redis MemcacheDB Dynamo Voldemort	MongoDB CouchDB ArangoDB MarkLogic RzthinkDB	BigTable Habse Casandra Accumulo Hypertable	Neo4j OrientDB Allegro Virtuoso InfiniteGraph

3.2.5 Variations of NoSQL Architectural Patterns

GQ. How NoSQL data architecture patterns varies as you move from a single processor to multiple processors? **(10 Marks)**

The key-value store, graph store, Bigtable store, and document store patterns can bemodified by focusing on a different aspect of system implementation. Variations on the architectures that use RAM or solid state drives (SSDs), and the patterns can be used on distributed systems or modified to createenhanced availability. Finally, we'll look at how database items can be groupedtogether in different ways to make navigation over many items easier.

Customization for RAM or SSD stores

Some NoSQL products are designed to specifically work with one type of memory; for example, Memcache, a key-value store, was specifically designed to see ifitems are in RAM on multiple servers. A key-value store that only uses RAM is called a RAM cache; it's flexible and has general tools that application developers can use tostore global variables, configuration files, or intermediate results of document transformations.

A RAM cache is fast and reliable, and can be thought of as another programmin construct like an array, a map, or a lookup system. There are several things about ther you should consider:

- Simple RAM resident key-value stores are generally empty when the server startsu and can only be populated with values on demand.
- You need to define the rules about how memory is partitioned between theRAM cach and the rest of your application.
- RAM resident information must be saved to another storage system if you want itt persist between server restarts.

The key is to understand that RAM caches must be re-created from scratch each time a server restart. A RAM cache that has no data in it is called a cold cache and is why some systems get faster the more they're used after a reboot.

SSD systems provide permanent storage and are almost as fast as RAM for read operations. The Amazon DynamoDB key-value store service uses SSDs for all its storage resulting in high-performance read operations. Write operations to SSDs can often be buffered in large RAM caches, resulting in fast write times until the RAM becomes full.

Distributed Stores

- NoSQL data architecture patterns vary from a single processor to multiple processors that are distributed over data centers in different geographic regions. The ability to elegantly and transparently scale to a large number of processors is a core property of most NoSQL systems. Ideally, the process of data distribution is transparent to the user, meaning that the API doesn't require you toknow how or where your data is stored. But knowing that your NoSQL software can scale and how it does this is critical in the software selection process.
- If your application uses many web servers, each caching the result of a long-running query, it's most efficient to have a method that allows the servers to worktogether to avoid duplication. This mechanism, known as memcache. Whether we use NoSQL or traditional SQL systems, RAM continues to be the most expensive and precious resource in anapplication server's configuration. If you don't have enough RAM, your application won't scale.
- The solution used in a distributed key-value store is to create a simple, lightweightprotocol that checks whether any other server has an item in its cache. If it does, this item is quickly returned to the requester and no additional searching is

required. The protocol is simple: each memcache server has a list of the other memcache servers it's working with. Whenever a memcache server receives a request that's not in its own cache, it checks with the other peer servers by sending them the key.

- The memcache protocol shows that you can create simple communication protocols between distributed systems to make them work efficiently as a group. This type of information sharing can be extended to other NoSQL data architectures such as Big table stores and document stores. You can generalize the key-value pair to other patterns by referring to them as cached items.
- Cached items can also be used to enhance the overall reliability of a data service by replicating the same items in multiple caches. If one server goes down, other servers quickly fill in so that the application gives the user the feeling of service without interruption. To provide a seamless data service without interruption, the cached items need to be replicated automatically on multiple servers. If the cached items are stored on two servers and the first one becomes unavailable, the second server can quickly returnthe value; there's no need to wait for the first server to be rebooted or restored from backup.

Grouping Items

- In the key-value store section, we looked at how web pages can be stored in a key-value store using a website URL as the key and the web page as the value. We can extend this construct to file systems as well. In a filesystem, the key is the directory or folder path, and the value is the file content. But unlike web pages, file systems have the ability to list all the files in a directory without having to open the files. If the file content is large, it would be inefficient to load all of the files into memory each time you want a listing of the files.
- To make this easier and more efficient, a key-value store can be modified to include additional information in the structure of the key to indicate that the key-value pair is associated with another key-value pair, creating a collection, or general-purpose structures used to group resources. Though each key-value store system might call it something different (such as folders, directories, or buckets), the concept is the same.
- The implementation of a collection system can also vary dramatically based onwhat NoSQL data pattern you use. Key-value stores have several methods to group similar items based on attributes in their keys. Graph stores associate one or moregroup identifiers with each triple. Big data systems use column families to group similar columns. Document stores use a concept of a document collection.

- One approach to grouping items is to have two key-value data types, the first called resource keys and the second collection keys. We can use collection keys to store a list ofkeys that are in a collection. This structure allows you to store a resource in multiple collections and also to store collections within collections. Using this design poses some complex issues that require careful thought and planning about what should bedone with a resource if it's in more than one collection and one of the collections is deleted.
- To simplify this process and subsequent design decisions, key-value systems can include the concept of creating collection hierarchies and require that a resource bein one and only one collection. The result is that the path to a resource is essentially a distinct key for retrieval. Also known as a simple document hierarchy, the familiar concept of folders and documents resonates well with end users.

These are some examples where we can use the concept of a collection hierarchy in a key, use it to perform many functions on groups of key-value pairs:

- Associate metadata with a collection (who created the collection, when it was created, the last time it was modified, and who last modified the collection).
- Give the collection an owner and group, and associate access rights with the owner group and other users in the same way UNIX file systems use permissions.
- Create an access control permission structure on a collection, allowing only users with specific privileges the ability to read or modify the items within the collection.
- Create tools to upload and/or download a group of items into a collection.
- Set up systems that compress and archive collections if they haven't been accessed for a specific period of time.

3.2.6 NoSQL case studies

LiveJournal's Memcache

- Engineers working on the blogging system Live Journal started to look at how their systems were using their most precious resource: the RAM in each web server. Live-Journal had a problem. Their website was so popular that the number of visitors using the site continued to increase on a daily basis. The only way they could keep up with demand was to continue to add more web servers, each with its own separate RAM.

- To improve performance, the LiveJournal engineers found ways to keep the results of the most frequently used database queries in RAM, avoiding the expensive cost of rerunning the same SQL queries on their database. But each web server had its own copy of the query in RAM; there was no way for any web server to know that the server next to it in the rack already had a copy of the query sitting in RAM.
- So the engineers at LiveJournal created a simple way to create a distinct "signature" of every SQL query. This signature or hash was a short string that represented a SQL SELECT statement. By sending a small message between web servers, any web server could ask the other servers if they had a copy of the SQL result already executed. If one did, it would return the results of the query and avoid an expensive round trip to the already overwhelmed SQL database. They called their new system Memcache because it managed RAM memory cache.
- Many other software engineers had come across this problem in the past. The concept of large pools of shared-memory servers wasn't new. What was different this time was that the engineers for LiveJournal went one step further. They not only made this system work (and work well), they shared their software using an open source license, and they also standardized the communications protocol between the web front ends (called the memcached protocol). Now anyone who wanted to keep their database from getting overwhelmed with repetitive queries could use their front end tools.

Google's MapReduce—use commodity hardware to create search indexes

- Google shared their process for transforming large volumes of web data content into search indexes using low-cost commodity CPUs. Though sharing of this information was significant, the concepts of map and reduce weren't new. Map and reduce functions are simply names for two stages of a data transformation
- The initial stages of the transformation are called the map operation. They're responsible for data extraction, transformation, and filtering of data. The results of the map operation are then sent to a second layer: the reduce function. The reduce function is where the results are sorted, combined, and summarized to produce the final result.
- The core concepts behind the map and reduce functions are based on solid computer science work that dates back to the 1950s when programmers at MIT implemented these functions in the influential LISP system. LISP was different than other programming languages because it emphasized functions that transformed isolated lists of data. This focus is now the basis for many modern functional programming languages that have desirable properties on distributed systems.

- Google extended the map and reduce functions to reliably execute on billions of web pages on hundreds or thousands of low-cost commodity CPUs. Google made map and reduce work reliably on large volumes of data and did it at a low cost. It was Google's use of MapReduce that encouraged others to take another look at the power of functional programming and the ability of functional programming systems to scale over thousands of low-cost CPUs. Software packages such as Hadoop have closely modeled these functions.
- The use of MapReduce inspired engineers from Yahoo! and other organizations to create open source versions of Google's MapReduce. It fostered a growing awareness of the limitations of traditional procedural programming and encouraged others to use functional programming systems.

Google's Bigtable—a table with a billion rows and a million columns

Google also influenced many software developers when they announced their Bigtable system white paper titled A Distributed Storage System for Structured Data. The motivation behind Bigtable was the need to store results from the web crawlers that extract HTML pages, images, sounds, videos, and other media from the internet. The resulting dataset was so large that it couldn't fit into a single relational database, so Google built their own storage system. Their fundamental goal was to build a system that would easily scale as their data increased without forcing them to purchase expensive hardware. The solution was neither a full relational database nor a filesystem, but what they called a "distributed storage system" that worked with structured data.

By all accounts, the Bigtable project was extremely successful. It gave Google developers a single tabular view of the data by creating one large table that stored all the data they needed. In addition, they created a system that allowed the hardware to be located in any data center, anywhere in the world, and created an environment where developers didn't need to worry about the physical location of the data they manipulated.

Amazon's Dynamo—accept an order 24 hours a day, 7 days a week

- Google's work focused on ways to make distributed batch processing and reporting easier, but wasn't intended to support the need for highly scalable web storefronts that ran 24/7. This development came from Amazon. Amazon published another significant NoSQL paper: Amazon's 2007 Dynamo: A Highly Available Key-Value Store. The business motivation behind Dynamo was Amazon's need to create a highly reliable web storefront that supported transactions from around the world 24 hours a day, 7 days a week, without interruption.

- Traditional brick-and-mortar retailers that operate in a few locations have the luxury of having their cash registers and point-of-sale equipment operating only during business hours. When not open for business, they run daily reports, and perform backups and software upgrades. The Amazon model is different. Not only are their customers from all corners of the world, but they shop at all hours of the day, every day. Any downtime in the purchasing cycle could result in the loss of millions of dollars. Amazon's systems need to be iron-clad reliable and scalable without a loss in service.
- In its initial offerings, Amazon used a relational database to support its shopping cart and checkout system. They had unlimited licenses for RDBMS software and a consulting budget that allowed them to attract the best and brightest consultants for their projects. In spite of all that power and money, they eventually realized that a relational model wouldn't meet their future business needs.
- Many in the NoSQL community cite Amazon's Dynamo paper as a significant turning point in the movement. At a time when relational models were still used, it challenged the status quo and current best practices. Amazon found that because key-value stores had a simple interface, it was easier to replicate the data and more reliable. In the end, Amazon used a key-value store to build a turnkey system that was reliable, extensible, and able to support their 24/7 business model, making them one of the most successful online retailers in the world.

MarkLogic

- In 2001 a group of engineers in the San Francisco Bay Area with experience in document search formed a company that focused on managing large collections of XML documents. Because XML documents contained markup, they named the company MarkLogic.
- MarkLogic defined two types of nodes in a cluster: query and document nodes. Query nodes receive query requests and coordinate all activities associated with executing a query. Document nodes contain XML documents and are responsible for executing queries on the documents in the local filesystem.
- Query requests are sent to a query node, which distributes queries to each remote server that contains indexed XML documents. All document matches are returned to the query node. When all document nodes have responded, the query result is then returned.

- The MarkLogic architecture, moving queries to documents rather than moving documents to the query server, allowed them to achieve linear scalability with petabytes of documents.
- MarkLogic found a demand for their products in US federal government systems that stored terabytes of intelligence information and large publishing entities that wanted to store and search their XML documents. Since 2001, MarkLogic has matured into a general-purpose highly scalable document store with support for ACID transactions and fine-grained, role-based access control. Initially, the primary language of MarkLogic developers was XQuery paired with REST; newer versions support Java as well as other language interfaces.
- MarkLogic is a commercial product that requires a software license for any datasets over 40 GB. NoSQL is associated with commercial as well as open-source products that provide innovative solutions to business problems.

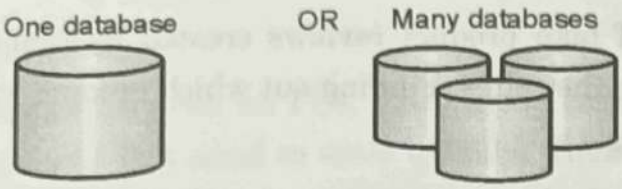
3.3 NOSQL SOLUTION FOR BIG DATA

A big data class problem is any business problem that's so large that it can't be easily managed using a single processor. Big data problems force you to move away from a single-processor environment toward the more complex world of distributed computing. Though great for solving big data problems, distributed computing environments come with their own set of challenges.

Some of the challenges you face when you move from a single processor to a distributed computing system. Moving to a distributed environment is a nontrivial endeavour and should be done only if the business problem really warrants the need to handle large data volumes in a short period of time. This is why platforms like Hadoop are complex and require a complex framework to make things easier for the application developer.

These are challenges for one or many databases:

NoSQL includes concepts and use cases that can be managed by a single processor and have a positive impact on agility and data quality. But we consider big data problems a primary use case for NoSQL.



- Easy to understand
- Easy to setup and configure
- Easy to administer
- Single source of truth
- Limited scalability

- Data partitioning
- Replication
- Clustering
- Query distribution
- Load balancing
- Consistency/Suncing
- Latency/Concurrency
- Clock synchronization
- Network bottlenecks/failures
- Multiple data centers
- Distributed backup
- Node failure
- Voting algorithms for error detection
- Administration of many systems
- Monitoring
- Scalable if designed correctly

Fig. 3.3.1

Here are some typical big data use cases:

(1) Bulk Image Processing	(2) Public Web Page Data
(3) Remote Sensor Data	(4) Event Log Data
(5) Mobile Phone Data	(6) Social Media Data
(7) Game Data	(8) Open Linked Data

(1) Bulk Image Processing

- Organizations like NASA regularly receive terabytes of incoming data from satellites or even rovers on Mars. NASA uses a large numberof servers to process these images and perform functions like image enhancement and photo stitching.
- Medical imaging systems like CAT scans and MRIs need to convert raw image data into formats that are useful to doctors and patients. Custom imaging hardware has been found to be more expensive thanrenting a large number of processors on the cloud when they're needed.
- For example, the New York Times converted 3.3 million scans of old newspaper articles into web formats using tools like Amazon EC2 and Hadoop for a few hundred dollars.

(2) Public Web Page Data

- Publicly accessible pages are full of information that organizations can use to be more competitive. They contain news stories, RSS feeds, new product information, product reviews, and blog postings. Not all of the information is authentic.

- There are millions of pages of fake product reviews created by competitors third parties paid to disparage other sites. Finding out which product reviews valid is a topic for careful analysis.

(3) Remote Sensor Data

- Small, low-power sensors can now track almost any aspect of our world. Devi installed on vehicles track location, speed, acceleration, and fuel consumpti and tell your insurance company about your driving habits.
- Road sensors can warn about traffic jams in real time and suggest alterna routes. You can even track the moisture in your garden, lawn, and indoor pla to suggest a watering plan for your home.

(4) Event Log Data

- Computer systems create logs of read-only events from web page hits (also call clickstreams), email messages sent, or login attempts.
- Each of these events can help organizations understand who's using wh resources and when systems may not be performing according to specification.
- Event log data can be fed into operational intelligence tools to send alerts to use when key indicators fall out of acceptable ranges.

(5) Mobile Phone Data

- Every time users move to new locations; applications can track these events. Y can see when your friends are around you or when customers walk through yo retail store.
- Although there are privacy issues involved in accessing this data, it's forming new type of event stream that can be used in innovative ways to give companies competitive advantage.

(6) Social Media Data

- Social networks such as Twitter, Facebook, and LinkedIn provide a continuou real-time data feed that can be used to see relationships andtrends.
- Each site creates data feeds that you can use to look at trends in customer moo or get feedback on your own as well as competitor products.

(7) Game Data

- Games that run on PCs, video game consoles, and mobile deviceshave back-end datasets that need to scale quickly. These games store and sharehigh scores for all users as well as game data for each player.
- Game site backends must be able to scale by orders of magnitude if viral marketing campaign scatch on with their users.

(8) Open Linked Data

- Not only is this data large, butit may require complex tools to reconcile, remove duplication, and find invaliditems.
- This includes problems like image and signal processing. Theirfocus is on the efficient and reliable data transformation at scale. These use cases don't need the query or transactions support provided by many NoSQL systems.

They read and write to key-value stores or distributed filesystems like *Amazon's Simple StorageService (S3)* or *Hadoop Distributed File System (HDFS)* and may not need the advanced features of a document store or an RDBMS. Other use cases are more demanding and need more features. Big data problems like event log data and game data do need to store their data directly into structures that can be queried and analysed, so they will need different NoSQL solutions.

To be a good candidate for a general class of big data problems, NoSQL solutionsshould :

1. Be efficient with input and output and scale linearly with growing data size.
2. Be operationally efficient. Organizations can't afford to hire many people to run the servers.
3. Require that reports and analyses be performed by nonprogrammers using simple toolsnot every business can afford a full-time Java programmer to write on-demand queries.
4. Meet the challenges of distributed computing, including consideration of latency between systems and eventual node failures.
5. Meet both the needs of overnight batch processing economy-of-scale and time-critical event processing.

6. RDBMS can, with enough time and effort, be customized to solve some big data problems. Applications can be rewritten to distribute SQL queries to many processors and merge the results of the queries. Databases can be redesigned to remove joins between tables that are physically located on different nodes. SQL systems can be configured to use replication and other data synchronization processes. Yet these steps all take considerable time and money.

3.4 UNDERSTANDING THE TYPES OF BIG DATA PROBLEMS

There are many types of big data problems, each requiring a different combination of NoSQL systems. After you've categorized your data and determined its type, you'll find there are different solutions.

Fig.3.4.1 is a good example of a high-level big data classification system.

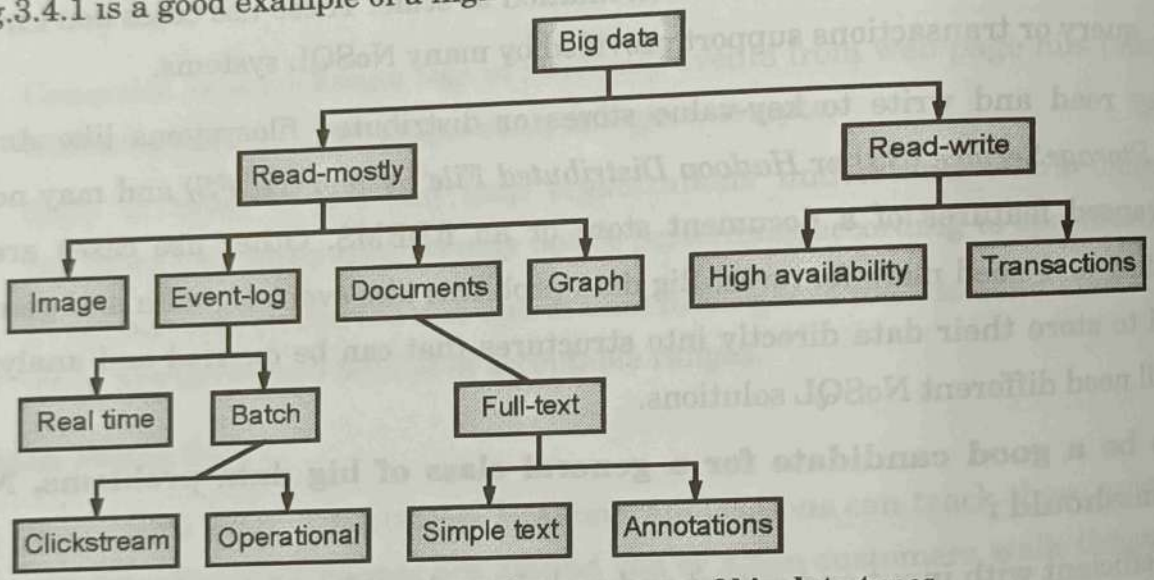


Fig.3.4.1:A sample of a taxonomy of big data types

Some ways you classify big data problems and see how NoSQL systems are changing the way organizations use data.

(1) **Read-mostly :** Read-mostly data is the most common classification. It includes data that's created once and rarely altered. This type of data is typically found in data warehouse applications but is also identified as a set of non-RDBMS items like images or video, event-logging data, published documents, or graph data. Event data includes things like retail sales events, hits on a website, system logging data, or real-time sensor data.

(2) **Log events :** When operational events occur in your enterprise, you can record it in a log file and include a timestamp so you know when the event occurred. Log events

may be a web page click or an out-of-memory warning on a disk drive. In the past, the cost and amount of event data produced were so large that many organizations opted not to gather or analyse it. Today, NoSQL systems are changing companies' thoughts on the value of log data as the cost to store andanalyse it is more affordable.

The ability to cost-effectively gather and store log events from all computers in your enterprise has lead to BI operational intelligence systems. Operational intelligence goes beyond analysing trends in your web traffic or retail transactions. It can integrate information from network monitoring systems so you can detect problems before they impact your customers. Cost-effective NoSQL systems can be part of good operations management solutions.

(3) **Full-text documents :** This category of data includes any document that contains natural-language text like the English language. An important aspect of document stores is that you can query the entire contents of your office document inthe same way you would query rows in your SQL system.

This means that you can create new reports that combine traditional data inRDBMSs as well as the data within your office documents. For example, youcould create a single query that extracted all the authors of titles of PowerPoint slides that contained the keywords NoSQL or big data. The result of this list of authors could then be filtered with a list of titles in the HR database to show which people had the title of Data Architect or Solution Architect.

This is a good example of how organizations are trying to tap into the hidden skills that already exist within an organization for training and mentorship. Integrating documents into what can be queried is opening new doors in knowledge management and efficient staff utilization.

3.5 ANALYZING BIG DATA WITH A SHARED-NOTHING ARCHITECTURE

GQ. Explain Shared Nothing architecture in detail. **(10 Marks)**

There are three ways that resources can be shared between computer systems: shared RAM, shared disk, and shared-nothing. Fig.3.5.1 shows a comparison of these three distributed computing architectures. Of the three alternatives, a shared-nothing architecture is most cost effective interms of cost per processor when you're using commodity hardware.

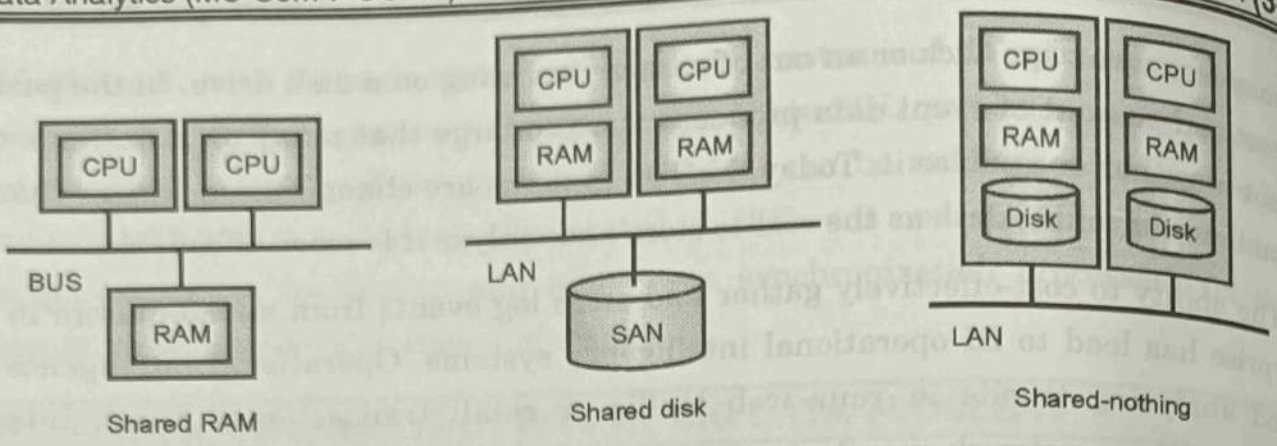


Fig.3.5.1 :Three ways to share resources

In Fig 3.5.1, The left panel shows a shared RAM architecture, where many CPUs access a single shared RAM over a high-speed bus. This system is ideal for large graph traversal. The middle panel shows a shared disk system, where processors have independent RAM but share disk using a storage area network (SAN). The right panel shows an architecture used in big data solutions: cache-friendly, using low-cost commodity hardware, and a shared-nothing architecture.

Of the architectural data patterns row store, key-value store, graph store, document store, and Bigtable store, only two (key-value store and document store) lend themselves to cache-friendliness. Bigtable stores scale well on shared-nothing architectures because their row-column identifiers are similar to key-value stores. But row stores and graph stores aren't cache-friendly since they don't allow a large BLOB to be referenced by a short key that can be stored in the cache. For graph traversals to be fast, the entire graph should be in main memory. This is why graph stores work most efficiently when you have enough RAM to hold the graph.

3.6 CHOOSING DISTRIBUTION MODELS: MASTER-SLAVE VERSUS PEER-TO-PEER

GQ. Explain Distributing models in details. **(10 Marks)**

- From a distribution perspective, there are two main models: master-slave and peer-to-peer. Distribution models determine the responsibility for processing data when arequest is made.
- Understanding the pros and cons of each distribution model is important when you're looking at a potential big data solution. Peer-to-peer models may be more resilient to failure than master-slave models. Some master-slave distribution models havesingle points of failure that might impact your system availability, so you might need to take special care when configuring these systems.

- In the master-slave model, one node is in charge (master). When there's no single node with a special role in taking charge, you have a peer-to-peer distribution model.
- Fig. 3.6.1 shows how these models each work.

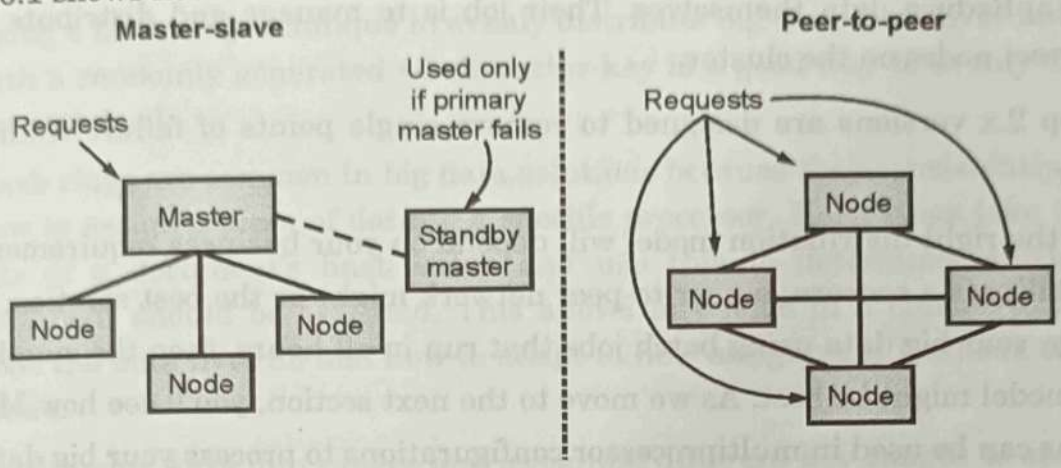


Fig.3.6.1 : Master-slave versus peer-to-peer

- In Fig.3.6.1 on the left illustrates a master-slave configuration where all incoming database requests (reads or writes) are sent to a single master node and redistributed from there. The master node is called the NameNode in Hadoop.
- This node keeps a database of all the other nodes in the cluster and the rules for distributing requests to each node. The panel on the right shows how the peer-to-peer model stores all the information about the cluster on each node in the cluster. If any node crashes, the other nodes can take over and processing can continue.
- With a master-slave distribution model, the role of managing the cluster is done on a single master node. This node can run on specializedhardware such as RAID drives to lower the probability that it crashes. The cluster canalso be configured with a standby master that's continually updated from the masternode.
- The challenge with this option is that it's difficult to test the standby master without jeopardizing the health of the cluster. Failure of the standby master to takeover from the master node is a real concern for high-availability operations.
- Peer-to-peer systems distribute the responsibility of the master to each node in the cluster. In this situation, testing is much easier since you can remove any node in the cluster and the other nodes will continue to function.
- The disadvantage of peer-to-peer networks is that there's an increased complexity and communication overhead that must occur for all nodes to be kept up to date with the cluster status.

- The initial versions of Hadoop (frequently referred to as the 1.x versions) were designed to use a master-slave architecture with the NameNode of a cluster being responsible for managing the status of the cluster. NameNodes usually don't deal with any MapReduce data themselves. Their job is to manage and distribute queries to thecorrect nodes on the cluster.
- Hadoop 2.x versions are designed to remove single points of failure from a Hadoop cluster.
- Using the right distribution model will depend on your business requirements: if high availability is a concern, a peer-to-peer network might be the best solution. If you can manage your big data using batch jobs that run in off hours, then the simpler master-slave model might be best. As we move to the next section, you'll see how Map-Reduce systems can be used in multiprocessor configurations to process your big data.

3.7 NOSQL SYSTEMS TO HANDLE BIG DATA PROBLEMS

1. Moving queries to the data, not data to thequeries

- With the exception of large graph databases, most NoSQL systems use commodity processors that each hold a subset of the data on their local shared-nothing drives.
- When a client wants to send a general query to all nodes that hold data, it's more efficient to send the query to each node than it is to transfer large datasets to a central processor. This may seem obvious, but it's amazing how many traditional databases still can't distribute queries and aggregate query results.
- This simple rule helps you understand how NoSQL databases can have dramatic performance advantages over systems that weren't designed to distribute queries to the data nodes.
- Consider an RDBMS that has tables distributed over two different nodes. In order for the SQL query to work, information about rows on one table must all be moved across the network to the other node.
- Larger tables result in more data movement, which results in slower queries. Think of all the steps involved. The tables can be extracted, serialized, sent through the network interface, transmitted over networks, reassembled, and then compared on the server with the SQL query.
- Keeping all the data within each data node in the form of logical documents means that only the query itself and the final result need to be moved over a network. This keeps your big data queries fast.

2. Using hash rings to evenly distribute data on a cluster

- One of the most challenging problems with distributed databases is figuring out a consistent way of assigning a document to a processing node.
- Using a hash ring technique to evenly distribute big data loads over many servers with a randomly generated 40-character key is a good way to evenly distribute a network load.
- Hash rings are common in big data solutions because they consistently determine how to assign a piece of data to a specific processor. Hash rings take the leading bits of a document's hash value and use this to determine which node the document should be assigned. This allows any node in a cluster to know what node the data lives on and how to adapt to new assignment methods as your data grows.
- Partitioning keys into ranges and assigning different key ranges to specific nodes is known as keyspace management. Most NoSQL systems, including MapReduce, use keyspace concepts to manage distributed computing problems.
- The concept of a hash ring can also be extended to include the requirement that an item must be stored on multiple nodes.
- When a new item is created, the hash ring rules might indicate both a primary and a secondary copy of where an item is stored. If the node that contains the primary fails, the system can look up the node where these secondary item is stored.

3. Using replication to scale reads

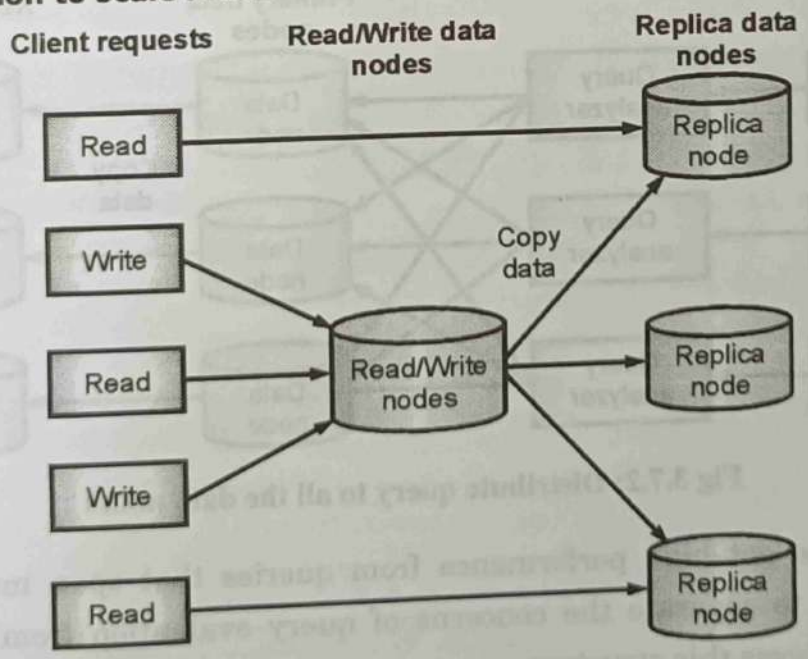


Fig 3.7.1 : Duplication of the Data to increase the performance of NoSQL system

- The Fig 3.7.1 show how you can replicate data to speed read performance in NoSQL systems. All incoming client requests enter from the left. All reads can be directed to any node, either a primary read/write node or a replica node. All write transactions can be sent to a central read/write node that will update the data and then automatically send the updates to replica nodes. The time between the write to the primary and the time the update arrives on the replica nodes determines how long it takes for reads to return consistent results.
- There are only a few times when you must be concerned about the lag time between a write to the read/write node and a client reading that same record from a replica. One of the most common operationsafter a write is a read of that same record. If a client does a write and then an immediate read from that same node, there's no problem. The problem occurs if a readoccurs from a replica node before the update happens. This is an example of an inconsistent read.
- The best way to avoid this type of problem is to only allow reads to the same write node after a write has been done. This logic can be added to a session or state management system at the application layer. Almost all distributed databases relax database consistency rules when a large number of nodes permit writes. If your application needs fast read/write consistency, you must deal with it at the application layer.

4 Letting the database distribute queries evenly to data nodes

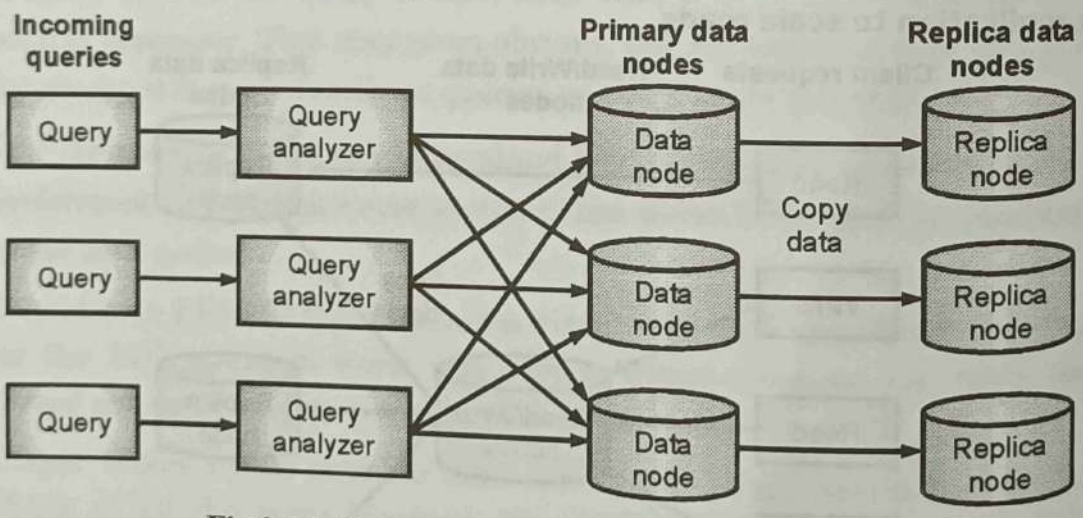


Fig 3.7.2: Distribute query to all the data nodes

- In order to get high performance from queries that span multiple nodes, it's important to separate the concerns of query evaluation from query execution. Fig.3.7.2 shows this structure.

- In this figure NoSQL systems move the query to a data node, but don't move data to a query node. In this example, all incoming queries arrive at query analyzer nodes. These nodes then forward the queries to each data node. If they have matches, the documents are returned to the query node.
- The query won't return until all data nodes (or a response from a replica) have responded to the original query request. If the data node is down, a query can be redirected to a replica of the data node.
- The approach shown in Fig. 3.7.2 is one of moving the query to the data rather than moving the data to the query. This is an important part of NoSQL big data strategies. In this instance, moving the query is handled by the database server, and distribution of the query and waiting for all nodes to respond is the sole responsibility of the database, not the application layer.
- This approach is somewhat similar to the concept of federated search. Federated search takes a single query and distributes it to distinct servers and then combines the results together to give the user the impression they're searching a single system. In some cases, these servers may be in different geographic regions. In this case, sending query to a single cluster that's not only performing search queries on a single local cluster but also performing update and delete operations.

ChapterEnds...

❑❑❑

MODULE IV

CHAPTER 4

Mining Big Data Streams

University Prescribed Syllabus w.e.f Academic Year 2022-2023

4.1 **The Stream Data Model :** A Data-Stream-Management System, Examples of Stream Sources, Stream Queries, Issues in Stream Processing.

4.2 Sampling Data techniques in a Stream

4.3 **Filtering Streams :** Bloom Filter with Analysis.

4.4 Counting Distinct Elements in a Stream, Count - Distinct Problem, Flajolet-Martin Algorithm, Combining Estimates, Space Requirements

4.5 **Counting Ones in a Window :** The Cost of Exact Counts,The Datar-Gionis-Indyk-Motwani Algorithm, Query Answering in the DGIM Algorithm, Decaying Windows.

4.1 THE STREAM MODEL

- As data arrives in various streams, it's important to stream the data as the data which arrives cannot be stored immediately and the probability of losing the data is more.
- To mine the data, we need to create the sample of stream data and to filter the stream to eliminate most of the "undesirable" data elements.
- To summarize the large stream of data, consider the fixed length window consisting of last n elements.
- The data stream model is considered for the management of data, along with the stream queries and issues in stream processing
- In recent years, advances in hardware technology have facilitated the ability to collect data continuously. Simple transactions of everyday life such as using a credit card, a phone or browsing the web lead to automated data storage. Similarly, advances in information technology have lead to large flows of data across IP networks.
- In many cases, these large volumes of data can be mined for interesting and relevant information in a wide variety of applications. When the volume of the underlying data is very large, it leads to a number of computational and mining challenges:
- With increasing volume of the data, it is no longer possible to process the data efficiently by using multiple passes. Rather, one can process a data item at most once.
- This leads to constraints on the implementation of the underlying algorithms. Therefore, stream mining algorithms typically need to be designed so that the algorithms work with one pass of the data.
- In most cases, there is an inherent temporal component to the stream mining process. This is because the data may evolve over time.
- This behavior of data streams is referred to as temporal locality. Therefore, a straightforward adaptation of one-pass mining algorithms may not be an effective solution to the task. Stream mining algorithms need to be carefully designed with a clear focus on the evolution of the underlying data.
- Another important characteristic of data streams is that they are often mined in a distributed fashion. Furthermore, the individual processors may have limited processing and memory.

- Common examples of streaming data sources include:
  - IoT sensors
  - Real-time advertising platforms
  - Server and security logs
  - Click-stream data from apps and websites

4.1.1 Data Stream Mining Characteristics

- **Continuous Stream of Data**. High amount of data in an infinite stream. we do not know the entire dataset
- **Concept Drifting**. The data change or evolves over time
- **Volatility of data**. The system does not store the data received (Limited resources). When data is analysed it's discarded or summarised

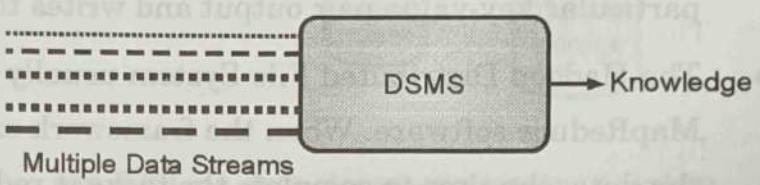


Fig. 4.1.1 : Data stream management system

Next, is discussed how to analyse Data Streams. Data-Based Techniques rely on analysing a representative subset of data. This technique also is used as pre-processing for Data Stream algorithms. On the Other Hand, Mining Techniques are enhanced versions of traditional Data Mining Algorithms.

4.2 DATA BASED TECHNIQUES

Sampling is based on selecting subset of data uniformly distributed.

- **Reservoir Sampling** (Fixed Size Sample)This algorithm sample a subset m of the data from the stream. After the i^{th} item is chosen with probability 1 / i and if the i element is already sampled its randomly replace a sampled item.
- **Min-Wise Sampling** is based on assigning a random γ in range 0 to 1 to a subset of samples m. When the system retrieves m elements, we select the sample with the minimum γ.
- **Sketching** is based on reducing the dimensionality of the dataset. Sketch is based on performing a linear transformation of the data delivering a summarization of the Stream. See also Count-min Sketch.
- **Approximation Techniques** are based on maintaining only a subset of data and discarding previous data (sliding windows).

- **Sequence based :** The size of the window depends on the number of observations. The window store γ elements and when a new element arrives, the last element is removed if the window is full.
- **Timestamp based:** The size of the window depends on time. The window is bounded in instant T_n and T_{n+1} and holds the elements received in this period.
- At a high level, MapReduce breaks input data into fragments and distributes them across different machines. The input fragments consist of key-value pairs. Parallel map tasks process the chunked data on machines in a cluster. The mapping output then serves as input for the reduce stage. The reduce task combines the result into a particular key-value pair output and writes the data to HDFS.
- The Hadoop Distributed File System usually runs on the same set of machines as the MapReduce software. When the framework executes a job on the nodes that also store the data, the time to complete the tasks is reduced significantly.

4.2.1 Data Stream Management System (DSMS)

UQ. Explain with block diagram architecture of Data stream Management System. **(MU - Dec. 19, 10 Marks)**

UQ. Explain abstract architecture of Data Stream Management System (DSMS). **(MU - Dec. 16, 10 Marks)**

UQ. What is Data Stream Management System? Explain with block diagram. **(MU - May 17, 10 Marks)**

- We can view a stream processor as a kind of data-management system, the high-level organization which is shown in Fig. 4.2.1.
- Any number of streams can enter the system. Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform.
- The fact that the rate of arrival of stream elements is not under the control of the system distinguishes stream processing from the processing of data that goes on within a database-management system. T
- he latter system controls the rate at which data is read from the disk, and therefore never has to worry about data getting lost as it attempts to execute queries.
- Streams may be archived in a large archival store, but we assume it is not possible to answer queries from the archival store.

- It could be examined only under special circumstances using time-consuming retrieval processes.
- There is also a working store, into which summaries or parts of streams may be placed, and which can be used for answering queries.
- The working store might be disk, or it might be main memory, depending on how fast we need to process queries. But either way, it is of sufficiently limited capacity that it cannot store all the data from all the streams.

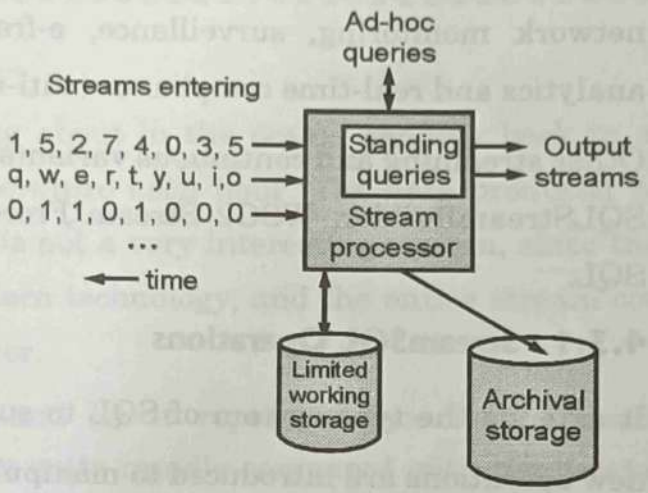


Fig. 4.2.1 : A data-stream-management system

- A Data Stream Management System (DSMS) is a computer software system to manage continuous **Data Streams**. It is similar to a **Database Management System** (DBMS), which is, however, designed for static data in conventional **Databases**.
- A DBMS also offers a flexible query processing so that the information needed can be expressed using queries. However, in contrast to a DBMS, a DSMS executes a continuous query that is not only performed once, but is permanently installed.
- Therefore, the query is continuously executed until it is explicitly uninstalled. Since most DSMS are data-driven, a continuous query produces new results as long as new data arrive at the system. This basic concept is similar to **Complex Event Processing** so that both technologies are partially coalescing.

4.3 StreamSQL

- StreamSQL is a query language that extends SQL with the ability to process real-time data streams. SQL is primarily intended for manipulating relations (also known as tables), which are finite bags of tuples (rows).
- StreamSQL adds the ability to manipulate streams, which are infinite sequences of tuples that are not all available at the same time.
- Because streams are infinite, operations over streams must be monotonic.
- Queries over streams are generally "continuous", executing for long periods of time and returning incremental results.

- The StreamSQL language is typically used in the context of a Data Stream Management System (DSMS), for applications including market data analytics, network monitoring, surveillance, e-fraud detection and prevention, clickstream analytics and real-time compliance (anti-money laundering, RegNMS, MiFID).
- Other streaming and continuous variants of SQL include StreamSQL.io, Kafka KSQL, SQLStreamBuilder, WSO2 Stream Processor, SQLStreams, SamzaSQL, and Storm SQL.

4.3.1 StreamSQL Operations

- It extends the type system of SQL to support streams in addition to tables. Several new operations are introduced to manipulate streams.
- **Selecting from a stream** : A standard SELECT statement can be issued against a stream to calculate functions (using the target list) or filter out unwanted tuples (using a WHERE clause). The result will be a new stream.
- **Stream-Relation Join** : A stream can be joined with a relation to produce a new stream. Each tuple on the stream is joined with the current value of the relation based on a predicate to produce 0 or more tuples.
- **Union and Merge** : Two or more streams can be combined by unioning or merging them. Unioning combines tuples in strict FIFO order. Merging is more deterministic, combining streams according to a sort key.
- **Windowing and Aggregation** : A stream can be windowed to create finite sets of tuples. For example, a window of size 5 minutes would contain all the tuples in a given 5-minute period. Window definitions can allow complex selections of messages, based on tuple field values. Once a finite batch of tuples is created, analytics such as count, average, max, etc., can be applied.
- **Windowing and Joining** : A pair of streams can also be windowed and then joined together. Tuples within the join windows will combine to create resulting tuples if they fulfill the predicate.

4.3.2 Examples of Stream Sources

GQ. Explain different types of Stream sources.

Sensor Data

- Imagine a temperature sensor bobbing about in the ocean, sending back to a base station a reading of the surface temperature each hour. The data produced by this sensor is a stream of real numbers. It is not a very interesting stream, since the data rate is so low. It would not stress modern technology, and the entire stream could be kept in main memory, essentially forever.
- Now, give the sensor a GPS unit, and let it report surface height instead of temperature. The surface height varies quite rapidly compared with temperature, so we might have the sensor send back a reading every tenth of a second. If it sends a 4-byte real number each time, then it produces 3.5 megabytes per day. It will still take some time to fill up main memory, let alone a single disk.
- But one sensor might not be that interesting. To learn something about ocean behavior, we might want to deploy a million sensors, each sending back a stream, at the rate of ten per second. A million sensors isn't very many; there would be one for every 150 square miles of ocean. Now we have 3.5 terabytes arriving every day, and we definitely need to think about what can be kept in working storage and what can only be archived.

Image Data

- Satellites often send down to earth streams consisting of many terabytes of images per day. Surveillance cameras produce images with lower resolution than satellites, but there can be many of them, each producing a stream of images at intervals like one second.
- London is said to have six million such cameras, each producing a stream.

Internet and Web Traffic

- A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs. Normally, the job of the switch is to transmit data and not to retain it or query it. But there is a tendency to put more capability into the switch, e.g., the ability to detect denial-of-service attacks or the ability to reroute packets based on information about congestion in the network.

- Web sites receive streams of various types. For example, Google receives several hundred million search queries per day. Yahoo! accepts billions of "clicks" per day on its various sites. Many interesting things can be learned from these streams. For example, an increase in queries like "sore throat" enables us to track the spread of viruses. A sudden increase in the click rate for a link could indicate some news connected to that page, or it could mean that the link is broken and needs to be repaired.

4.3.3 Stream Queries

UQ. With respect to data stream querying, give example of Ad-hoc queries and standing queries. **(MU - May 17, 5 Marks)**

- There are two ways that queries get asked about streams. We show in Fig. 4.1 a place within the processor where standing queries are stored. These queries are, in a sense, permanently executing, and produce outputs at appropriate times.
- **Example 4.1 :** The stream produced by the ocean-surface-temperature sensor. It might have a standing query to output an alert whenever the temperature exceeds 25 degrees centigrade. This query is easily answered, since it depends only on the most recent stream element.
- Alternatively, we might have a standing query that, each time a new reading arrives, produces the average of the 24 most recent readings. That query also can be answered easily, if we store the 24 most recent stream elements. When a new stream element arrives, we can drop from the working store the 25th most recent element, since it will never again be needed.
- Another query we might ask is the maximum temperature ever recorded by that sensor. We can answer this query by retaining a simple summary: the maximum of all stream elements ever seen. It is not necessary to record the entire stream. When a new stream element arrives, we compare it with the stored maximum, and set the maximum to whichever is larger.
- We can then answer the query by producing the current value of the maximum. Similarly, if we want the average temperature over all time, we have only to record two values: the number of readings ever sent in the stream and the sum of those readings. We can adjust these values easily each time a new reading arrives, and we can produce their quotient as the answer to the query.

- The other form of query is ad-hoc, a question asked once about the current state of a stream or streams. If we do not store all streams in their entirety, as normally we cannot, then we cannot expect to answer arbitrary queries about streams. If we have some idea what kind of queries will be asked through the ad-hoc query interface, then we can prepare for them by storing appropriate parts or summaries of streams as in Example 4.1.
- If we want the facility to ask a wide variety of ad-hoc queries, a common approach is to store a sliding window of each stream in the working store. A sliding window can be the most recent n elements of a stream, for some n, or it can be all the elements that arrived within the last t time units, e.g., one day. If we regard each stream element as a tuple, we can treat the window as a relation and query it with any SQL query. Of course the stream-management system must keep the window fresh, deleting the oldest elements as new ones come in.
- **Example 4.2 :** Web sites often like to report the number of unique users over the past month. If we think of each login as a stream element, we can maintain a window that is all logins in the most recent month. We must associate the arrival time with each login, so we know when it no longer belongs to the window. If we think of the window as a relation Logins(name, time), then it is simple to get the number of unique users over the past month. The SQL query is: SELECT COUNT(DISTINCT(name)) FROM Logins WHERE time >= t; Here, t is a constant that represents the time one month before the current time. Note that we must be able to maintain the entire stream of logins for the past month in working storage. However, for even the largest sites, that data is not more than a few terabytes, and so surely can be stored on disk.

4.4 KEY ISSUES IN BIG DATA STREAM ANALYSIS

UQ. What are the challenges of querying on large data stream? **(MU - May 18, 5 Marks)**

Scalability

- One of the main challenges in big data streaming analysis is the issue of scalability. The big data stream Is experiencing exponential growth in a way much faster than computer resources.
- The processors follow Moore's law, but the size of data is exploding. Therefore, research efforts should be geared towards developing scalable frameworks and algorithms that will accommodate data stream computing mode, effective resource allocation strategy and parallelization issues to cope with the ever-growing size and complexity of data.

☞ **Integration**

- Building a distributed system where each node has a view of the data flow, th is, every node performing analysis with a small number of sources, the aggregating these views to build a global view is non-trivial.
- An integration technique should be designed to enable efficient operations acro different datasets.

☞ **Fault-tolerance**

- High fault-tolerance is required in life-critical systems.
- As data is real-time and infinite in big data stream computing environments, a good scalable high fault-tolerance strategy is required that allows an application to continue working despite component failure without interruption.

☞ **Timeliness**

- Time is of the essence for time-sensitive processes such as mitigating security threats, thwarting fraud, or responding to a natural disaster.
- There is a need for scalable architectures or platforms that will enable continuous processing of data streams which can be used to maximize the timeliness of data.
- The main challenge is implementing a distributed architecture that will aggregate local views of data into global view with minimal latency between communicating nodes.

☞ **Consistency**

- Achieving high consistency (i.e. stability) in big data stream computing environments is non-trivial as it is difficult to determine which data are needed and which nodes should be consistent.
- Hence a good system structure is required.

☞ **Heterogeneity and incompleteness**

- Big data streams are heterogeneous in structure, organisations, semantics, accessibility and granularity. The challenge here is how to handle an always ever-increasing data, extract meaningful content out of it, aggregate and correlate streaming data from multiple sources in real-time.
- A competent data presentation should be designed to reflect the structure, diversity and hierarchy of the streaming data.

☞ **Load balancing**

- A big data stream computing system is expected to be self-adaptive to data streams changes and avoid load shedding.
- This is challenging as dedicating resources to cover peak loads 24/7 is impossible and load shedding is not feasible when the variance between the average load and the peak load is high.
- As a result, a distributing environment that automatically streams partial data streams to a global centre when local resources become insufficient is required.

☞ **High throughput**

- Decision with respect to identifying the sub-graph that needs replication, how many replicas are needed and the portion of the data stream to assign to each replica is an issue in big data stream computing environment.
- There is a need for good multiple instances replication if high throughput is to be achieved.

☞ **Privacy**

- Big data stream analytics created opportunities for analyzing a huge amount of data in real-time but also created a big threat to individual privacy.
- According to the International Data Cooperation (IDC), not more than half of the entire information that needs protection is effectively protected.
- The main challenge is proposing techniques for protecting a big data stream dataset before its analysis.

☞ **Accuracy**

- One of the main objectives of big data stream analysis is to develop effective techniques that can accurately predict future observations.
- However, as a result of inherent characteristics of big data such as volume, velocity, variety, variability, veracity, volatility, and value, big data analysis strongly constrain processing algorithms spatio-temporally and hence stream-specific requirements must be taken into consideration to ensure high accuracy.

4.5 SAMPLING TECHNIQUES FOR EFFICIENT STREAM PROCESSING

UQ. Describe any two sampling techniques for big data with the help of examples. **(MU - May 16, 10 Marks)**

4.5.1 Sliding Window

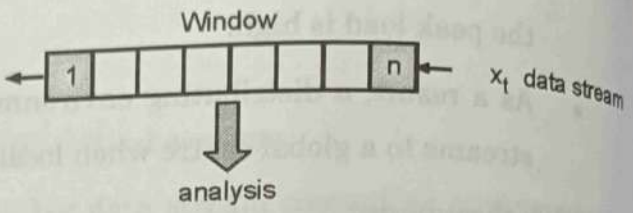


Fig. 4.5.1 : Sliding Window

- This is the simplest and most straightforward method. A first-in, first-out (FIFO) queue with size n and a skip / sub-sampling factor k ≥1 is maintained.
- In addition to that, a stride factor s ≥ 1 describes by how many time-steps the window is shifted before analyzing it.

Advantage

(1) Simple to implement.

(2) Deterministic — reservoir can be filled very fast from the beginning.

Drawbacks

The time history represented by the reservoir R is short; long-term concept drifts cannot be detected easily — outliers can create noisy analyses.

4.5.2 Unbiased Reservoir Sampling

- A reservoir R is maintained such that at time t > n the probability of accepting point x(t) in the reservoir is equal to n/t.
- The algorithm [1] is as follows:
  - Fill the reservoir R with the first n points of the stream.
  - At time t > n replace a randomly chosen (equal probability) entry in the reservoir R with acceptance probability n/t.
- This leads to a reservoir R(t) such that each point x(1)...x(t) is contained in R(t) with equal property n/t.

Advantages

(1) The reservoir contains data points from all history of the stream with equal probability.

(2) Very simple implementation; adding a point requires only O(1)

☞ **Drawbacks**

A concept drift cannot be compensated; the oldest data point x(1) is equal important in this sampling technique as the latest data point x(t).

4.5.3 Biased Reservoir Sampling

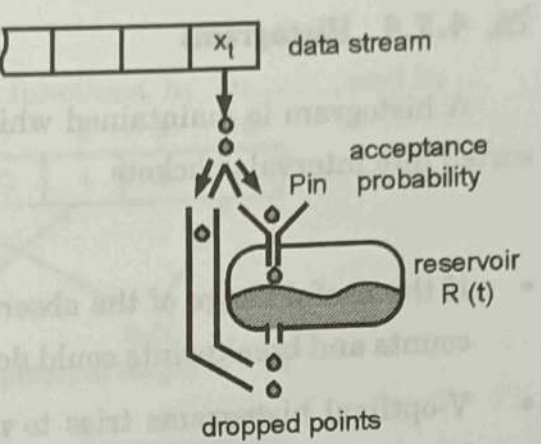


Fig. 4.5.2

- In biased reservoir sampling Alg. 3.1, [2] the probability of a data point x(t) being in the reservoir is a decreasing function of its lingering time within R.
- So the probability of finding points of the sooner history in R is high. Very old data points will be in R with very low probability.

☞ **Illustration of the Biased Reservoir Sampling**

The probability that point x(r) is contained in R(t) equals to

$$P(r, t) = e^{-\lambda (t-r)}$$

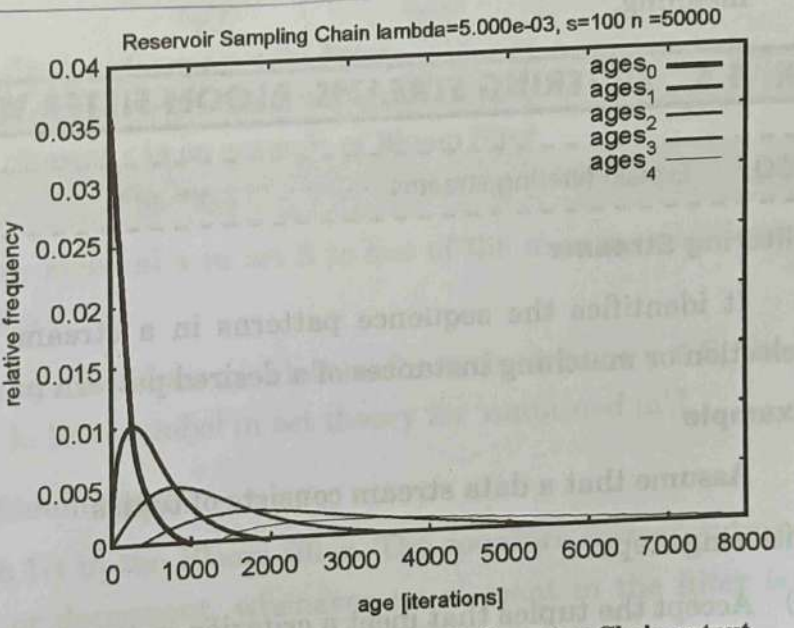


Fig. 4.5.3 : Reservoir Sampling Chain output

- So this is an exponential forgetting. For details of the algorithm, see [2] and the goreservoir package.
- The example from the github.com\andremueller\goservoir package shows the lingering time of a stack of unbiased reservoir samplers.

☞ **Advantages**

(1) O(1) algorithm for adding a new data point.

(2) Slowly moving concept drifts can be compensated.

(3) An adjustable forgetting factor can be tuned for the application of interest.

Drawbacks

It is a randomized technique. So the algorithm is non-deterministic. However, the variance might be estimated by running an ensemble of independent reservoirs [3]

$$R_1 \ldots R_B$$

4.5.4 Histograms

A histogram is maintained while observing the data stream. Hereto, data points are sorted into intervals/buckets

$$[li, u_i]$$

- If the useful range of the observed values is known in advance, a simple vector with counts and breakpoints could do the job.
- V-optimal histograms tries to minimize the variance within each histogram bucket. [4] proposes an algorithm for efficiently maintaining an approximate V-optimum histogram from a data stream. This is of relevance for interval data, such as a time-series of temperature values; i.e., absolute value and distance between values have a meaning.

4.6 FILTERING STREAMS: BLOOM FILTER WITH ANALYSIS

GQ. Explain filtering streams.

Filtering Streams

It identifies the sequence patterns in a stream. Stream filtering is the process of selection or matching instances of a desired pattern in a continuous stream of data.

Example

Assume that a data stream consists of tuples

Filtering steps:

(i) Accept the tuples that meet a criterion in the stream,

(ii) Pass the accepted tuples to another process as a stream and

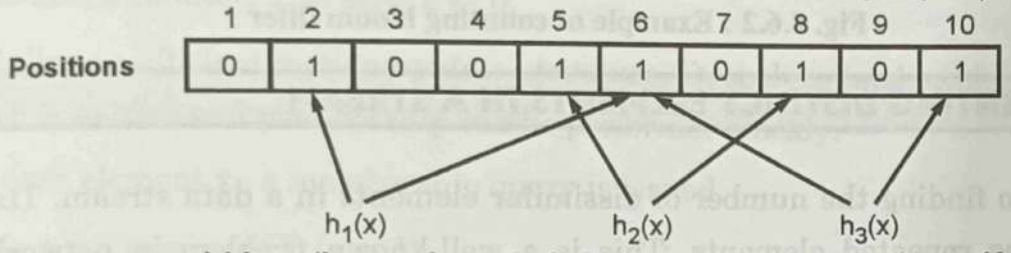
(iii) discard remaining tuples

UQ. How bloom filter is useful for big data analytics? Explain with one example. (MU - Dec. 18, 10 Marks)

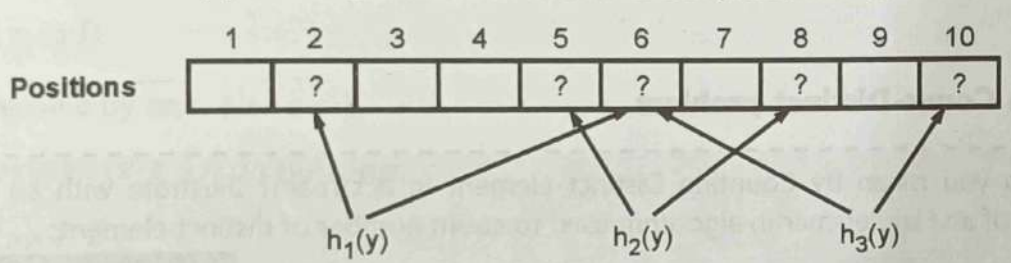
UQ. Explain the concept of Blooms Filter using an example. (MU - Dec. 17, 10 Marks)

Bloom Filter with Analysis

- A simple space-efficient data structure introduced by Burton Howard Bloom in 1970. The filter matches the membership of an element in a dataset.
- The filter is basically a bit vector of length m that represent a set $S = \{x_1, x_2, \ldots, x_m\}$ of m elements,
- Initially all bits 0. Then, define k independent hash functions, $h_1, h_2, \ldots,$ and h_k.



(a) Inserting an element x in bit vector (filter) of length m = 10



(b) Finding element y is the Filter

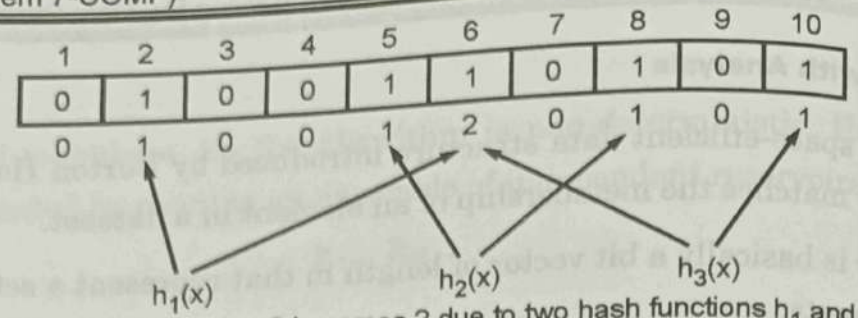
(a)Inserting an element x in bit vector(filter) of length m =10,
(b) finding an element y in an example of Bloom filter

Fig. 4.6.1

- Each of which maps (hashes) some element x in set S to one of the m array positions with a uniform random distribution.
- Number k is constant, and much smaller than m. That is, for each element $x \in S$, the bits $h_i(x)$ are set to 1 for $1 \leq i \leq k$. [$\in$ is symbol in set theory for 'contained in'.]

Counting Bloom Filter: A Variant of Bloom Filter

- It mmaintains a counter for each bit in the Bloom filter. The counters corresponding to the k hash values increment or decrement, whenever an element in the filter is added or deleted, respectively.
- As soon as a counter changes from 0 to 1, the corresponding bit in the bit vector is set to 1. When a counter changes from 1 to 0, the corresponding bit in the bit vector is set to 0. The counter basically maintains the number of elements that hashed



Counter at position 6 becomes 2 due to two hash functions h_1 and h_3, when an element x is inserted in bit vector (filter) of lenght m = 10

Fig. 4.6.2 : Example of counting bloom filter

4.7 COUNTING DISTINCT ELEMENTS IN A STREAM

It relates to finding the number of dissimilar elements in a data stream. The stream of data contains repeated elements. This is a well-known problem in networking and databases.

4.7.1 The Count-Distinct problem

UQ. What do you mean by Counting Distinct Element in a stream? Illustrate with an example working of an Flajolet-martin algorithm used to count number of distinct elements. **(MU - Dec. 19, 10 Marks)**

- The count-distinct problem[1] (also known in applied mathematics as the cardinality estimation problem) is the problem of finding the number of distinct elements in a data stream with repeated elements.
- This is a well-known problem with numerous applications.
- The elements might represent IP addresses of packets passing through a router, unique visitors to a web site, elements in a large database, motifs in a DNA sequence, or elements of RFID/sensor networks.

Formal definition

Instance

A stream of elements $x_1, x_2, \ldots x_n$ with repetitions, and an integer m. Let n be the number of distinct elements, namely $n = \{x_1, x_2, \ldots x_n\}$ and let these elements by $\{e1, e_2, \ldots e_n\}$.

Objective

Find an estimate n of n using only m storage units, where $m << n$

An example of an instance for the cardinality estimation problem is the steam a,b,a,c,d,b,d. For this instance.

$$n = |\{a, b, c, d\}| = 4$$

Naive solution [edit]

The naive solution to the problem is as follows :

- Inititalize a counter, c, to zero, $c \leftarrow 0$,
- Initialize an efficient dictionary data structure, D, such as hash table or search tree in which insertion and membership can be performet quickly.

For each element x_i, a membership query is issued.

If x_i is not a memer of D ($x_i \notin D$)

Add x_i to D

Increase c by one, $c \leftarrow c + 1$

Otherwise ($x_i \in D$) do nothing.

Output n = c.

4.7.2 Flajolet Martin Algorithm

UQ. What do you mean by Counting Distinct Elements in a stream? Illustrate with an example working of a Flajolet – Martin Algorithm used to count number of distinct elements. **(MU - Dec. 19, 10 Marks)**

UQ. Give problem in Flajolet-Martin (FM) algorithm to count distinct element in a stream **(MU - Dec. 16, 5 Marks)**

- Flajolet Martin Algorithm, also known as FM algorithm, is used to approximate the number of unique elements in a data stream or database in one pass.
- The highlight of this algorithm is that it uses less memory space while executing.
- Pseudo Code-Stepwise Solution:

1. Selecting a hash function h so each element in the set is mapped to a string to at least $\log_2 n$ bits.
2. For each element x, r(x)= length of trailing zeroes in h(x)
3. R= max(r(x)) = > Distinct elements = 2_R

Reasons for using Flajolet Martin algorithm

- Let us compare this algorithm with our conventional algorithm using python code. Assume we have an array (stream in code) of data of length 20 with 8 unique elements.
- Using the brute force approach to find the number of unique elements in the array, each element is taken into consideration. Another array (st_unique in code) is formed for unique elements.
- Initially, the new array is empty (st_unique length equals zero), so naturally, the first element is not present in it.
- The first element is considered to be unique as it does not exist in the new array and thus a copy of the first element is inserted into the new array (1 is appended to st_unique)
- Similarly, all the elements are checked, if they are already present in the new array, they are not considered to be unique, else a copy of the element is inserted into the new array.
- Running the brute force algorithm for our array, we will get 8 elements in the new array.
- If each element takes 20 bytes of data, the new array will take 8*20 = 160 bytes memory to run the algorithm.

```
stream=[1,2,3,4,5,6,4,2,5,9,1,6,3,7,1,2,2,4,2,1]
print('Using conventional Algorithm:')
start_time = time.time()
st_unique=[]
for i in stream:
 if i in st_unique:
 continue
 else:
 st_unique.append(i)
 print('distinct elements',len(st_unique))
print("--- %s seconds ---" % (time.time() - start_time))
```

- For the same array, if use the FM algorithm for the same array, we define a variable (maxnum in code) that stores the maximum number of zeroes at the end.
- For each value in the array(stream in code), we run a loop to covert its hash function in the form ax + b mod c,(a = 1, b = 6 and c = 32 in this case) into binary (we place [2:] at the end because in python when converted to binary, the number starts with '0b').

- We run another loop to find if the number of zeroes at the end exceeds the maximum number of zeroes.
- In this case, if each variable occupies 2 bytes of data, the whole program takes 4*20= 80 bytes of data, i.e half of the memory used in the above case.
- Here, we have considered the variables maxnum, sum, val, and j. We have not considered i, time, and stream as they are common in both of the codes.

```
stream=[1,2,3,4,5,6,4,2,5,9,1,6,3,7,1,2,2,4,2,1]
print('Using Flajolet Martin Algorithm:')
import time
start_time = time.time()
maxnum=0
for i in range(0,len(stream)):
 val= bin((1*stream[i] + 6) % 32)[2:]

 sum=0
 for j in range(len(val)-1,0,-1):

 if val[j]=='0':
 sum+=1
 else:
 break
 if sum>maxnum:
 maxnum=sum
print('distict elements', 2**maxnum)
print("--- %s seconds ---" % (time.time() - start_time))
```

- For small values of m(where m is the number of unique elements), the brute force approach can work, but for large data sets or data streams, where m is very large, a lot of space is required. The compiler may not let us run the algorithm in some cases.
- This is where the Flajolet Martin Algorithm can be used.
- Not only does it occupy less memory, but it also shows better results in terms of time in seconds when the python code is run which can be shown in our output as we calculated seconds taken by both algorithms by using time.time() in python.
- As shown in the output, it can clearly be said that the FM algorithm takes very little time as compared to the conventional algorithm.

```
Using  Flajolet Martin Algorithm :
distict elements 8
- - - 0.00011801719665527344 seconds - -
Using conventional Algorithm
distinct elements 8
---
7.295608520507812e-05 seconds - - -
```

4.8 COMBINING ESTIMATES

GQ. Explain the combining estimates method.

- The strategy for combining the estimates of m, the number of distinct elements, that we obtain by using many different hash functions.
- Our first assumption would be that if we take the average of the values 2^R that we get from each hash function, we shall get a value that approaches the true m, the more hash functions we use. Consider a value of r such that 2^r is much larger than m. There is some probability p that we shall discover r to be the largest number of 0's at the end of the hash value for any of the m stream elements. Then the probability of finding r + 1 to be the largest number of 0's instead is at least p/2. However, if we do increase by 1 the number of 0's at the end of a hash value, the value of 2^R doubles.
- Consequently, the contribution from each possible large R to the expected value of 2^R grows as R grows, and the expected value of 2^R is actually infinite.
- Another way to combine estimates is to take the median of all estimates. The median is not affected by the occasional outsized value of 2^R, so the worry described above for the average should not carry over to the median. Unfortunately, the median suffers from another defect: it is always a power of 2. Thus, no matter how many hash functions we use, should the correct value of m be between two powers of 2, say 400, then it will be impossible to obtain a close estimate.
- There is a solution to the problem, however. We can combine the two methods. First, group the hash functions into small groups, and take their average. Then, take the median of the averages.
- It is true that an occasional outsized 2^R will bias some of the groups and make them too large. However, taking the median of group averages will reduce the influence of this effect almost to nothing.

- Moreover, if the groups themselves are large enough, then the averages can be essentially any number, which enables us to approach the true value m as long as we use enough hash functions. In order to guarantee that any possible average can be obtained, groups should be of size at least a small multiple of $\log_2 m$.

4.8.1 Space Requirements

- As we read the stream it is not necessary to store the elements seen. The only thing we need to keep in main memory is one integer per hash function; this integer records the largest tail length seen so far for that hash function and any stream element.
- If we are processing only one stream, we could use millions of hash functions, which is far more than we need to get a close estimate. Only if we are trying to process many streams at the same time would main memory constrain the number of hash functions we could associate with any one stream.
- In practice, the time it takes to compute hash values for each stream element would be the more significant limitation on the number of hash functions we use.

UQ. Give two applications for counting the number of 1's in a long stream of binary values. Using a stream of binary digits, illustrate how the DGIM algorithm will find the number of 1's?

(MU - May 18, 5 Marks)

4.9 COUNTING ONES IN A WINDOW

- The Cost of Exact Counts Sliding window model - data elements arrive at every instant; each data element expires after exactly N time steps; and the portion of data that is relevant to gathering statistics or answering queries is the set of the last N elements to arrive.
- Able to count exactly the number of 1s in the last k bits for any k<=N, it is necessary to store all N bits of the window. Problem:
- Most counting applications N is still so large that it cannot be stored on disk or there are so many streams that windows for all cannot be stored.
- Another method: Random Sampling – this will fail when the 1s may not arrive in a uniform manner.

4.9.1 Datar-Gionis-Indyk-Motwani

UQ. Explain DGIM algorithm for counting ones in stream with example. **(MU - May 19, Dec. 18, 10 Marks)**

UQ. Using an example bit stream explain the working of the DGIM algorithm to count number of 1's in a data stream. **(MU - May 16, 10 Marks)**

The simplest case of an algorithm called DGIM. This version of the algorithm uses $O(\log^2 N)$ bits to represent a window of N bits and enables the estimation of the number of 1s in the window with an error of no more than 50%.

Divide the window into buckets, consisting of – The timestamp of its right 1 – The number of 1s in the bucket. This number must be power of 2, and we refer to the number of 1s as the size of the bucket.

The right end of a bucket always starts with a position with a 1.

- Number of 1s must be power of 2.
- Either one or two buckets with the same power of 2 number of 1s exists.
- Buckets do not overlap in timestamps
- Buckets are stored by size.
- Buckets disappear when their end-time is N time units in the past.

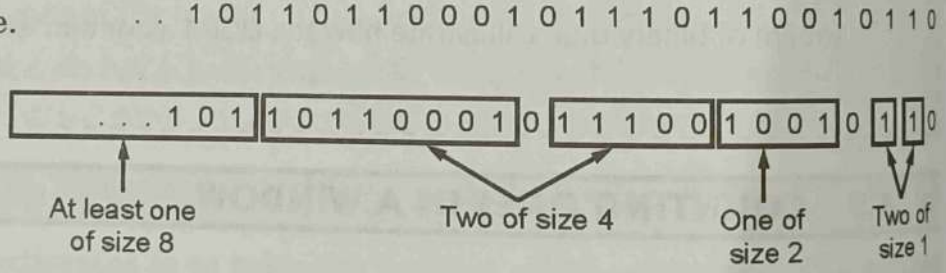


Fig. 4.9.1 : A bit-stream divided into buckets following the DGIM rules

4.9.2 Query Answering in the DGIM Algorithm

- Suppose we are asked how many 1's there are in the last k bits of the window, for some $1 \leq k \leq N$. Find the bucket b with the earliest timestamp that includes at least some of the k most recent bits. Estimate the number of 1's to be the sum of the sizes of all the buckets to the right (more recent) than bucket b, plus half the size of b itself
- **Example :** Suppose the stream is that of Fig. 4.9.1, and k = 10. Then the query asks for the number of 1's in the ten rightmost bits, which happen to be 0110010110. Let the current timestamp (time of the rightmost bit) be t. Then the two buckets with one 1, having timestamps t – 1 and t – 2 are completely included in the answer.

- The bucket of size 2, with timestamp t – 4, is also completely included. However, the rightmost bucket of size 4, with timestamp t–8 is only partly included. We know it is the last bucket to contribute to the answer, because the next bucket to its left has timestamp less than t – 9 and thus is completely out of the window.
- On the other hand, we know the buckets to its right are completely inside the range of the query because of the existence of a bucket to their left with timestamp t – 9 or greater.
- Our estimate of the number of 1's in the last ten positions is thus 6. This number is the two buckets of size 1, the bucket of size 2, and half the bucket of size 4 that is partially within range. Of course the correct answer is 5. 2 Suppose the above estimate of the answer to a query involves a bucket b of size 2j that is partially within the range of the query.
- Let us consider how far from the correct answer c our estimate could be. There are two cases: the estimate could be larger or smaller than c.
- **Case 1:** The estimate is less than c. In the worst case, all the 1's of b are actually within the range of the query, so the estimate misses half bucket b, or 2j–1 1's. But in this case, c is at least 2j; in fact it is at least 2j+1 – 1, since there is at least one bucket of each of the sizes 2j – 1, 2j – 2, . . . , 1. We conclude that our estimate is at least 50% of c.
- **Case 2:** The estimate is greater than c. In the worst case, only the rightmost bit of bucket b is within range, and there is only one bucket of each of the sizes smaller than b. Then $c = 1 + 2j - 1 + 2j - 2 + \cdots + 1 = 2j$ and the estimate we give is $2j-1 + 2j-1 + 2j-2 + \cdots + 1 = 2j + 2j-1 - 1$. We see that the estimate is no more than 50% greater than c.

4.9.3 Decaying Windows

GQ. Explain Decaying windows method.

It useful in applications which need identification of most common elements. Decaying window concept assigns more weight to recent elements. The technique computes a smooth aggregation of all the 1's ever seen in the stream, with decaying weights. When element further appears in the stream, less weight is given. The effect of exponentially decaying weights is to spread out the weights of the stream elements as far back in time as the stream flows.

The Problem of Most-Common Elements

- Suppose we have a stream whose elements are the movie tickets purchased all over the world, with the name of the movie as part of the element. We want to keep a summary of the stream that is the most popular movies "currently." While the notion of "currently" is imprecise, intuitively, we want to discount the popularity of a movie like Star Wars–Episode 4, which sold many tickets, but most of these were sold decades ago.
- On the other hand, a movie that sold n tickets in each of the last 10 weeks is probably more popular than a movie that sold 2n tickets last week but nothing in previous weeks. One solution would be to imagine a bit stream for each movie.
- The i^{th} bit has value 1 if the i^{th} ticket is for that movie, and 0 otherwise. Pick a window size N, which is the number of most recent tickets that would be considered in evaluating popularity. Then, use the method of Section 4.6 to estimate the number of tickets for each movie, and rank movies by their estimated counts.
- This technique might work for movies, because there are only thousands of movies, but it would fail if we were instead recording the popularity of items sold at Amazon, or the rate at which different Twitter-users tweet, because there are too many Amazon products and too many tweeters. Further, it only offers approximate answers

Definition of the Decaying Window

- An alternative approach is to redefine the question so that we are not asking for a count of 1's in a window. Rather, let us compute a smooth aggregation of all the 1's ever seen in the stream, with decaying weights, so the further back in the stream, the less weight is given.
- Formally, let a stream currently consist of the elements a1, a2, . . . , at, where a1 is the first element to arrive and at is the current element. Let c be a small constant, such as 10–6 or 10–9. Define the exponentially decaying window for this stream to be the sum

$$\sum_{i=0}^{t=1} a_{t-i}(1-c)^i$$

- The effect of this definition is to spread out the weights of the stream elements as far back in time as the stream goes.

- In contrast, a fixed window with the same sum of the weights, 1/c, would put equal weight 1 on each of the most recent 1/c elements to arrive and weight 0 on all previous elements. The distinction is suggested by Fig. 4.9.2.

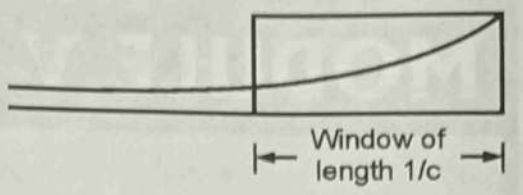


Fig. 4.9.2 : A decaying window and a fixed-length window of equal weight

It is much easier to adjust the sum in an exponentially decaying window than in a sliding window of fixed length. In the sliding window, we have to worry about the element that falls out of the window each time a new element arrives. That forces us to keep the exact elements along with the sum, or to use an approximation scheme such as DGIM. However, when a new element at+1 arrives at the stream input, all we need to do is:

1. Multiply the current sum by 1 – c.
2. Add a_{t+1}.

The reason this method works is that each of the previous elements has now moved one position further from the current element, so its weight is multiplied by 1 – c. Further, the weight on the current element is $(1 - c)^0 = 1$, so adding a_{t+1} is the correct way to include the new element's contribution.

Finding the Most Popular Elements

- The problem of finding the most popular movies in a stream of ticket sales. We shall use an exponentially decaying window with a constant c, which you might think of as 10–9 . That is, we approximate a sliding window holding the last one billion ticket sales.
- For each movie, we imagine a separate stream with a 1 each time a ticket for that movie appears in the stream, and a 0 each time a ticket for some other movie arrives. The decaying sum of the 1's measures the current popularity of the movie.
- We imagine that the number of possible movies in the stream is huge, so we do not want to record values for the unpopular movies. Therefore, we establish a threshold, say 1/2, so that if the popularity score for a movie goes below this number, its score is dropped from the counting. For reasons that will become obvious, the threshold must be less than 1, although it can be any number less than 1.

When a new ticket arrives on the stream, do the following:

1. For each movie whose score we are currently maintaining, multiply its score by (1 – c).
2. Suppose the new ticket is for movie M. If there is currently a score for M, add 1 to that score. If there is no score for M, create one and initialize it to 1.
3. If any score is below the threshold 1/2, drop that score.

Chapter Ends...

❑❑❑

MODULE V

CHAPTER 5

Real-Time Big Data Models

University Prescribed Syllabus w.e.f Academic Year 2022-2023

5.1 A Model for Recommendation Systems, Content-Based Recommendations, Collaborative Filtering

5.2 Case Study : Product Recommendation

5.3 Social Networks as Graphs, Clustering of Social-Network Graphs, Direct Discovery of Communities in a social graph

5.1 A MODEL FOR RECOMMENDATION SYSTEMS, CONTENT-BASED RECOMMENDATIONS, COLLABORATIVE FILTERING

UQ. How recommendation is done based on properties of product? Explain with suitable example. (MU - May 19, 10 Marks)

UQ. What are Recommendation Systems? (MU - Dec. 17, 5 Marks)

5.1.1 A Model for Recommendation Systems

A recommendation system is also called as Recommender system. A recommender system is really an automated system to filter some entities can be any products, ads, people, movies or songs. And we see this from all over on a daily basis from Amzon, Netfix, Youtube, FilpCart etc.

A recommender system captures the pattern of people's behaviour and use it to predict what else they might want or like.

For example, we watch a movie and then later on we get a recommendation for a different movie based on the power of previous viewing history. It could also be a product that we bought and then we get a recommendation for another product based on the previous product viewing or purchase history. And recommender doesn't work only in what products we are being shown, but also in what order the products are being ranked.

For example, if you've recently purchased a book on Machine Learning in Python and you've enjoyed reading it, it's very likely that you'll also enjoy reading a book on Data Visualization. People also tend to have similar tastes to those of the people they're close to in their lives. Recommender systems try to capture these patterns and similar behaviours, to help predict what else you might like.

Applications

- What to buy?
  - E-commerce, books, movies, shoes, etc.
- Where to eat?
- Which job to apply to?
- Who you should be friends with?
  - LinkedIn, Facebook
- Personalize your experience on the web

- News platforms, news personalization

Recommender systems function with two kinds of information:

- **Characteristic information**. This is information about items (keywords, categories, etc.) and users (preferences, profiles, etc.).
- **User-item interactions.** This is information such as ratings, number of purchases, likes, etc.

Why the Recommendation system?

- Benefits users in finding items of their interest.
- Help item providers in delivering their items to the right user.
- Identity products that are most relevant to users.
- Personalized content.
- Help websites to improve user engagement.

Advantages of Recommender system

1. Broder exposure
2. Possibility of continual usage or purchase of products
3. Provides better experience

5.1.2 Applications for Recommendation System

UQ. Clearly explain two applications for Recommendation system. **(MU - Dec. 17, 5 Marks)**

(1) **Entertainment** : recommendations for movies, music, and IPTV.

(2) **Content** : personalized newspapers, recommendation for documents, recommendations of Web pages, e-learning applications, and e-mail filters.

(3) **E-commerce** : recommendations for consumers of products to buy such as books, cameras, PCs etc.

(4) **Services** : recommendations of travel services, recommendation of experts for consultation, recommendation of houses to rent, or matchmaking services

5.1.3 Types of Recommender Systems

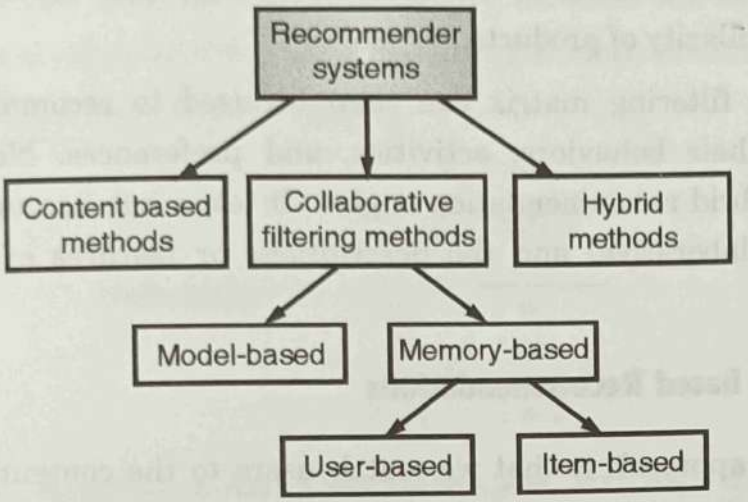


Fig. 5.1.1

Collaborative filtering

- Collaborative filtering focuses on collecting and analyzing data on user behavior, activities, and preferences, to predict what a person will like, based on their similarity to other users.
- To plot and calculate these similarities, collaborative filtering uses a matrix style formula. An advantage of collaborative filtering is that it doesn't need to analyze or understand the content (products, films, books). It simply picks items to recommend based on what they know about the user.

Content-based filtering

- Content-based filtering works on the principle that if you like a particular item, you will also like this other item.
- To make recommendations, algorithms use a profile of the customer's preferences and a description of an item (genre, product type, color, word length) to work out the similarity of items using cosine and Euclidean distances.
- The downside of content-based filtering is that the system is limited to recommending products or content similar to what the person is already buying or using. It can't go beyond this to recommend other types of products or content. For example, it couldn't recommend products beyond homeware if the customer had only brought homeware.

Hybrid model

- A hybrid recommendation engine looks at both the meta (collaborative) data and the transactional (content-based) data. Because of this, it outperforms both.

- To recommend items that are most similar to the items the user has bought, we compute cosine similarity between the articles the user has read and other articles. The ones that are most similar are recommended. Thus this is item-item similarity.

$$\text{cosine}(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}$$

- Cosine similarity is best suited when you have high dimensional features, especially in information retrieval and text mining.

2. **Jaccard similarity:**

- Also known as intersection over union, the formula is as follows:

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

- This is used for item-item similarity. We compare item vectors with each other and return the items that are most similar.
- Jaccard similarity is useful only when the vectors contain binary values. If they have rankings or ratings that can take on multiple values, Jaccard similarity is not applicable.
- This method is useful when we have a whole lot of 'external' features, like weather conditions, market factors, etc. which are not a property of the user or the product and can be highly variable. For example, the previous day's opening and closing price play an important role in determining the profitability of investing in a particular stock.
- This comes under the class of supervised problems where the label is whether the user liked/clicked on a product or not (0/1) or the rating the user gave that product or the number of units the user bought.
- For example, if a user likes movies such as 'Mission Impossible' then we can recommend him the movies of 'Tom Cruise' or movies with the genre 'Action'.

- In this filtering, two types of data are used. First, the likes of the user, the user's interest, user's personal information such as age or, sometimes the user's history too. This data is represented by the user vector. Second, information related to the product's known as an item vector. The item vector contains the features of all items based on which similarity between them can be calculated.

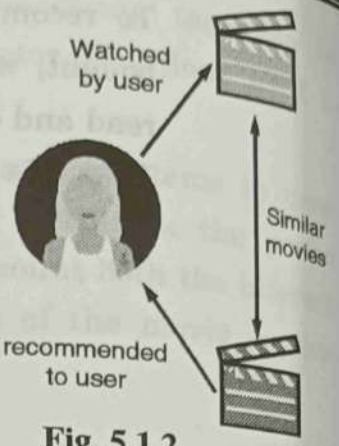


Fig. 5.1.2

- The recommendations are calculated using cosine similarity. If 'A' is the user vector and 'B' is an item vector then cosine similarity Values calculated in the cosine similarity matrix are sorted in descending order and the items at the top for that user are recommended.

5.1.5 Content-Based Recommendations Advantages and Disadvantages

Sr. No.	Advantages	Disadvantages
1	Does not depend on data of other users.	When we have a new user, without much information about his transactions, we cannot make accurate recommendations
2	There is no cold start problem for new items. This is because, using the item features we can easily find items it is similar to.	Clear-cut groups of similar products may result in not recommending different products. E may end up recommending a small subset over and over again
3	Recommendation results are interpretable.	If there is limited information about the content, it is difficult to clearly discriminate between items and group recommendations

5.1.6 Collaborative Filtering

- The recommendations are done based on the user's behavior. History of the user plays an important role. For example, if the user 'A' likes 'Coldplay', 'The Linkin Park' and 'Britney Spears' while the user 'B' likes 'Coldplay',

- 'The Linkin Park' and 'Taylor Swift' then they have similar interests. So, there is a huge probability that the user 'A' would like 'Taylor Swift' and the user 'B' would like 'Britney Spears'. This is the way collaborative filtering is done.

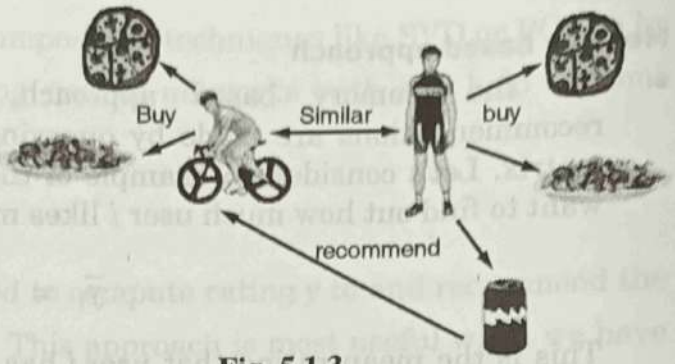


Fig. 5.1.3

- The underlying assumption of the collaborative filtering approach is that if A and B buy similar products, A is more likely to buy a product that B has bought than a product which a random person has bought.
- Unlike content based, there are no features corresponding to users or items here. All we have is the Utility Matrix. This is what it looks like:

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- A, B, C, D are the users, and the columns represent movies. The values represent ratings (1–5) a user has given a movie. In other cases, these values could be 0/1 depending on whether the user watched the movie or not. There are a 2 broad categories that collaborative filtering can be split into:

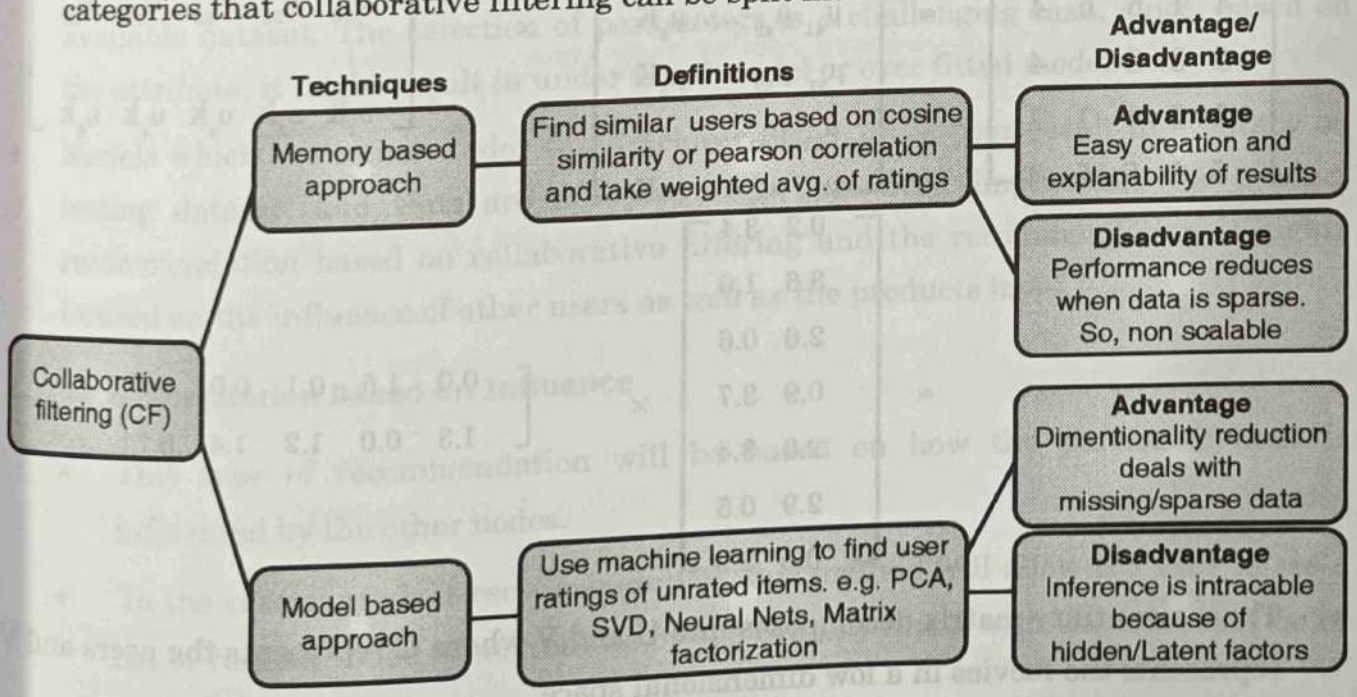


Fig. 5.1.4

Memory based approach

- For the memory based approach, the utility matrix is memorized and recommendations are made by querying the given user with the rest of the utility matrix. Let's consider an example of the same: If we have m movies and u users, we want to find out how much user i likes movie k.

$$\bar{y}_i = \frac{1}{|I_i|} \sum_{j \in I_i} y_{ij}$$

This is the mean rating that user i has given all the movies she/he has rated. Using this, we estimate his rating of movie k as follows:

Similarity between users a and i

$$\hat{y}_{ik} = \bar{y}_i + \frac{1}{\Sigma a \in U_k = |w_{ia}|} \sum_{a \in U_k} w_{ia}(y_{ak} - \bar{y}_a)$$

$w_{ia}(y_{ak} - \bar{y}_a)$ ← a's rating of k – a's average ratings

$a \in U_k$ ← All users that have rated k

Similarity between users a and i can be computed using any methods like cosine similarity / Jaccard similarity / Pearson's correlation coefficient, etc. These results are very easy to create and interpret, but once the data becomes too sparse, performance becomes poor.

Model based approach

- One of the more prevalent implementations of model based approach is Matrix Factorization. In this, we create representations of the users and items from the utility matrix. This is what it looks like :

$$\begin{bmatrix} 5 & 1 & 4 & 5 & 1 \\ & 5 & 2 & 1 & 4 \\ 1 & 4 & 1 & 1 & 2 \\ 4 & 1 & 5 & 5 & 4 \\ 5 & 3 & 3 & & 4 \\ 1 & 5 & 1 & 1 & 1 \\ 5 & 1 & 5 & 5 & 4 \end{bmatrix} \approx \begin{bmatrix} \mu_{11} & \mu_{12} & \dots & \mu_1 K \\ \mu_{21} & \mu_{22} & \dots & \mu_2 K \\ \mu_{31} & \mu_{32} & \dots & \mu_3 K \\ \mu_{41} & \mu_{42} & \dots & \mu_4 K \\ \mu_{51} & \mu_{52} & \dots & \mu_5 K \\ \mu_{61} & \mu_{62} & \dots & \mu_6 K \\ \mu_{71} & \mu_{72} & \dots & \mu_7 K \end{bmatrix} \times \begin{bmatrix} \upsilon_{11} & \upsilon_{21} & \upsilon_{31} & \upsilon_{41} & \upsilon_{61} \\ \upsilon_{12} & \upsilon_{22} & \upsilon_{32} & \upsilon_{42} & \upsilon_{62} \\ \vdots & & & & \\ \upsilon_1 k & \upsilon_2 k & \upsilon_3 k & \upsilon_4 k & \upsilon_6 k \end{bmatrix}$$

$$\approx \begin{bmatrix} 0.2 & 3.4 \\ 3.6 & 1.0 \\ 2.6 & 0.6 \\ 0.9 & 3.7 \\ 2.0 & 3.4 \\ 2.9 & 0.5 \\ 0.8 & 3.9 \end{bmatrix} \times \begin{bmatrix} 0.0 & 1.5 & 0.1 & 0.0 & 0.7 \\ 1.3 & 0.0 & 1.2 & 1.4 & 0.7 \end{bmatrix}$$

- Thus, our utility matrix decomposes into U and V where U represents the users and V represents the movies in a low dimensional space.

- This can be achieved by using matrix decomposition techniques like SVD or PCA or by learning the 2 embedding matrices using neural networks with the help of some optimizer like Adam, SGD etc.

$$\hat{y}_{ij} = u_i \cdot v_j$$

- For a user i and every movie j we just need to compute rating y to and recommend the movies with the highest predicted rating. This approach is most useful when we have a ton of data and it has high sparsity.
- Matrix factorization helps by reducing the dimensionality, hence making computation faster. One disadvantage of this method is that we tend to lose interpretability as we do not know what exactly elements of the user/item vectors mean.

5.2 CASE STUDY PRODUCT RECOMMENDATION

- Systems that automatically recommend products and content may play a more significant role than you realize. With the number of options offered to the consumer continuing to explode, product recommendation systems embody a key way in which machine learning serves as an antidote to "information overload" and "analysis paralysis." Movies, music, books... there are just too many options!
- Netflix recommends movies by predicting which ones you'd rate highly. Spotify and Pandora recommend music. Amazon was a pioneer recommending books
- Recommending users the products based on various parameters inferred from the available dataset. The selection of parameters is a challenging task, since, based on the attribute, it might result in under fitted model or over fitted model.
- Models which are either under fitted or over fitted, gives poor result in accuracy on testing dataset, and thus are not preferred. This work implements the product recommendation based on collaborative filtering and the recommendation is highly focused on the influence of other users as well as the products itself.

1. Recommendation Based on Influence

- This type of recommendation will be based on how the particular node is influenced by the other nodes.
- In the case of product recommendation, this method will allow the user to see a list of recommended items that he or she can buy along with the item that he or she has already chosen.

- Suppose item X is bought together with item Y. If the item X is bought with several other items as well, then we say that X and Y don't influence each other whereas, if the item X is mostly bought along with item Y then we say that item X and Y influence each other. This is called _influence scoring'.

2. Recommendation by Commonly Bought Products

- This method is like the transitive property. If X is bought together with Y and most of the people buying Y have also bought Z along with it, then item Z will also be recommended when any user is buying item X. Also, if the item Z is bought commonly then its chances of being recommended increases.
- This system performs collaborative filtering by considering the recommendations based on both, influence and products that are widely bought together.

3. Movie Recommendation

- It uses both, collaborative filtering technique and content-based filtering technique for implementation of movie recommendation system. Collaborative-filtering techniques deal with data of the particular user along with the data of other users. On the contrary, content-based filtering techniques deal with the data of the user alone.
- The collaborative filtering technique is very simple where the requirement is just to find the neighbors that share similar interests with that of the user and then recommend the user the movies that the neighbors, having similar taste, like. The basic idea is to recommend similar items (movies in this case) to the similar end-users of the recommendation system based on the history of ratings by the user for the items .

This collaborative filtering technique is divided into the following steps :

(1) **Collaborative filtering learning algorithm :** The collaborative filtering algorithm is dependent on a few parameters based on the movie and the rating by users. With these rating on some movies, by some user, the system will begin by finding out those parameters that affect this prediction and find out the best fit movie. For the ease of calculating, a cost function is set up to unroll these parameters into single vector params;

(2) **Calculating collaborative filtering :** The collaborative filtering calculations are carried out using the cost functions and gradients of the values that are obtained .

(3) **Regularized cost function :** After the calculation of the collaborative filtering parameters, the calculated measures of cost function and the gradients are regularized to further work with it.

The content-based filtering technique is divided into the following steps – identifying the actors, genres, directors and plots of the movies that the user has watched; apply content-based filtering algorithm: to suggest movies based on calculations.

(4) **Amazon product recommendation :** In order to provide customers with accurate product recommendations, **Amazon's algorithm must analyze huge amounts of data**. In this way, it better understands the behavior of all users and the interests of each viewer.

To do this, the recommendation engine collects two types of information

- general data about products and users;
- data on relations and dependencies between them.
- Getting to know the existing relationships in the online store will provide the recommendation engine with an insight into the real mechanisms governing customers' purchasing decisions.
- Amazon's recommendation algorithm analyzes 3 main types of dependencies and relationships for its operation:

User-product

- This type of relationship occurs when some users with certain characteristics prefer products of a given type and buy them more often.
- A good example would be gamers buying expensive computer components or fans of various series and movies buying related gadgets and t-shirts.

Product-product

- Product-product relationships occur when the products offered in the store are similar in terms of both appearance and specification.
- Some examples include books, movies, series or music of the same genre, or dishes from the same cuisine.
- User-user it occurs when individual customers with certain characteristics have similar tastes or preferences for certain products.

- Examples of such relationships include teenagers massively buying merchandise from their favorite You Tuber or cooking fans who prefer a specific line of kitchen products.
- In addition to collecting information about relationships and connections Amazon's recommendation algorithm also uses different types of product and user data:

User behaviour data

- This type of data is useful information about the preferences of individual customers, their interaction with the products in question.
- Amazon uses cookies to collect data about your browsing history, likes, or session length.

User Demographics data

- User demographics data is linked to personal information about individual customers, such as age, education, income and location.
- In order to collect such data, the user's consent is required.

Product Attribute Data

- Product Attribute data is information related to the product itself, such as the specification of the computer, blouse size information, collection description.

A method of filtering data by Amazon's recommendation algorithm

Many of these systems can be classified as content-based filtering or collaborative filtering.

Content-based filtering on Amazonie

- Content-based filtering is one of the simplest systems and is constantly used by modern recommendation systems.
- The main idea behind content-based filtering is that if a customer likes a certain product, chances are they will like another product with a similar specification.

Collaborative filtering

- Unlike content-based filtering, group filtering uses the experience of other users to generate recommendations.
- Interestingly, Amazon pioneered this approach and published the article Recommendations: Item-to-Item Collaborative Filtering in 2003, which later won an award from the Institute of Electrical and Electronics Engineers (IEEE).]

- The undoubted advantage of this method is that it allows the recommendation engine to generate proposals for relatively complex products, such as movies or music, without the need to have specialist knowledge about them.
- Compared to content-based filtering, team filtering produces better results in several key areas:
  - **diversity** : group filtering generates a more diverse list of recommended products, offering customers a wider choice,
  - **randomness :** recommendations generated using the collaborative filtering method are much more likely to positively surprise the client and show them a product of interest, which they might not otherwise discover,
  - **randomness :** the collaborative filtering method is able to more effectively present to customers novelties in the store's offer that users would most likely be interested in.
  - It is worth noting, however, that both collaborative and content-based methods offer the best and most effective results if they work together in one comprehensive recommendation engine.
  - This is how the Amazon algorithm works - by combining the best elements of both strategies, it offers its clients the highest quality recommendations.
  - Amazon's recommendation algorithm is probably the most complex and effective of the ecommerce market. The company has been steadily developing its recommendation system for almost 20 years and is now responsible for a large proportion of sales.
  - Amazon 's personalized recommendation engine improves customer experience and presents products in real time based on their browsing history

5.3 SOCIAL NETWORKS AS GRAPHS, CLUSTERING OF SOCIAL-NETWORK GRAPHS, DIRECT DISCOVERY OF COMMUNITIES IN A SOCIAL GRAPH

5.3.1 Social Networks as Graphs

- There is collection of entities that participate in network. Typically, people, but could be something else too. There is at least one relationship between entities of the network. For example: friends
- Sometimes Boolean : two people are either friends or they are not. They may have a Discrete degree. Eg. friends, family, acquaintances, or none. It could be real number : the fraction of the average day that two people. An example spends talking to each other.

- There is an assumption of non randomness or locality. This condition is the hardest to formalize, but the intuition is that relationships tend to cluster. That is, if entity A is related to both B and C, then there is a higher probability than average that B and C are related.
- Social networks are naturally modeled as graphs, which we sometimes refer to as a social graph. The entities are the nodes, and an edge connects two nodes if the nodes are related by the relationship that characterizes the network.
- If there is a degree associated with the relationship, this degree is represented by labeling the edges. Often, social graphs are undirected, as for the Face book friend's graph. But they can be directed graphs, as for example the graphs of followers on Twitter or Google+.

Ex. 5.3.1 : Fig. Ex. 5.3.1 is an example of a tiny social network. The entities are the nodes A through G. The relationship, which we might think of as "friends," is represented by the edges. For instance, B is friends with A, C, and D.Is this graph really typical of a social network, in the sense that it exhibits locality of relationships?

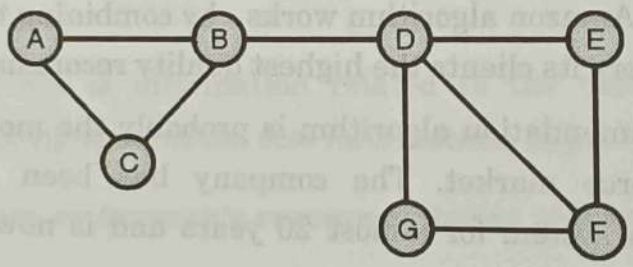


Fig. Ex. 5.3.1 : Example of a small social network

Soln. :

- Check for the non-randomness criterion. In a random graph (V, E) of 7 nodes and 9 edges, if XY is an edge, YZ is an edge, what is the probability that XZ is an edge?
- For a large random graph, it would be close to $\frac{|E|}{|V|C2} = \frac{9}{21} \sim 0.43$
- Small graph: XY and YZ are already edges, so compute within the rest. So the probability is $\frac{|E|-2}{(|V|C2-2)} = \frac{7}{19} = 0.37.$
- Now let's compute what is the probability for this graph in particular. For each X, check possible YZ and check if YZ is an edge or not Example : if X = A, YZ = {BC}, it is an edge.

X =	YZ =	Yes / Total
A	BC	$\frac{1}{1}$
B	AC, AD, CD	$\frac{1}{3}$
C	AB	$\frac{1}{1}$
D	BE, BG, BF, EF, EG, FG	$\frac{2}{6}$

X =	YZ =	Yes/ Total
E	DF	$\frac{1}{1}$
F	DE, DG, EG	$\frac{2}{3}$
G	DF	$\frac{1}{1}$
Total		$\frac{9}{16} \sim 0.56$

Varieties of Social Networks

Telephone networks

- Nodes are phone numbers.
- AB is an edge if A and B talked over phone within the last one week, or month, or ever.
- Edges could be weighted by the number of times phone calls were made, or total time of conversation.

Email networks: nodes are email addresses

- AB is an edge if A and B sent mails to each other within the last one week, or month, or ever.
  - One directional edges would allow spammers to have edges.
- Edges could be weighted.
- Other networks: collaboration network – authors of papers, jointly written papers or not.
- Also networks exhibiting locality property

Collaboration networks

- Nodes represent individuals who have published research papers. There is an edge between two individuals who published one or more papers jointly.
- Option-ally, we can label edges by the number of joint publications. The communities in this network are authors working on a particular topic.
- An alternative view of the same data is as a graph in which the nodes are papers. Two papers are connected by an edge if they have at least one author in common. Now, we form communities that are collections of papers on the same topic.

Other Examples of Social Graphs

- Many other phenomena give rise to graphs that look something like social graphs, especially exhibiting locality.
- Examples include: information networks(documents, web graphs, patents), infrastructure networks (roads, planes, water pipes, power grids), biological networks (genes, proteins, food-webs of animal seating each other), as well as other types, like product co-purchasing networks(e.g., Group on).

5.3.2 Clustering of Social-Network Graphs

GQ. Explain clustering of Social-Network Graphs with example. **(10 Marks)**

An important aspect of social networks is that they contain communities of entities that are connected by many edges. These typically correspond to groups of friends at school or groups of researchers interested in the same topic, for example.

Distance Measures for Social-Network Graphs

- When the edges of the graph have labels, these labels might be usable as a distance measure, depending on what they represented. But when the edges are unlabeled, as in a "friends" graph, there is not much we can do to define a suitable distance.
- Our first instinct is to assume that nodes are close if they have an edge between them and distant if not. Thus, we could say that the distance d (x, y) is 0 if there is an edge (x, y) and 1 if there is no such edge. We could use any other two values, such as 1 and ∞, as long as the distance is closer when there is an edge.
- Neither of these two-valued "distance measures" – 0 and 1 or 1 and ∞– is a true distance measure.

Applying Standard Clustering Methods

- In particular, suppose we use as the inter cluster distance the minimum distance between nodes of the two clusters. Hierarchical clustering of a social-network graph starts by combining some two nodes that are connected by an edge.
- Successively, edges that are not between two nodes of the same cluster would be chosen randomly to combine the clusters to which their two nodes belong. The choices would be random, because all distances represented by an edge are the same.

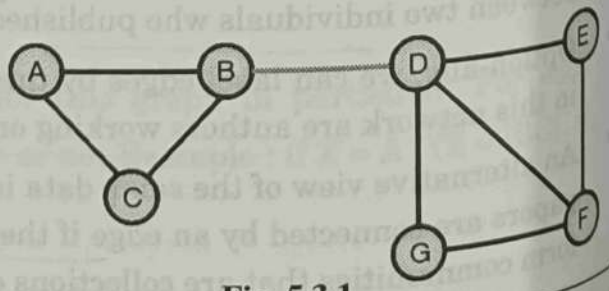


Fig. 5.3.1

Point-assignment approach

- in k-means randomly picked nodes might be in the same cluster
- randomly chosen and another as far away as possible doesn't do much better (i.e. E and G)
- even choosing B and F – problem with assignment of D

Betweenness

This method based on finding the edges that are **least likely to be inside a community.** Betweenness of and edge (a, b) is the number of pairs of nodes x and y such that the edge (a, b) lies on the shortest path between x and y. High value suggests that (a, b) runs between two different communities – a and b do not belong to the same community

The Girvan-Newman Algorithm

To exploit betweenness, we need to calculate the number of shortest paths going through each edge

Girvan-Newman Algorithm visits each node X once and computes the number of shortest paths from X to other nodes through each of the edges.

- **Step I :**

  BFS : Start at a node X, perform a BFS with X as root

  Observe : level of node Y = length of shortest path from X to Y

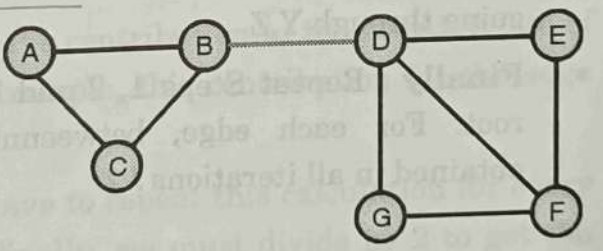


Fig. 5.3.2

Edges between level are called "DAG" edges

Each DAG edge is part of at least one shortest path from X

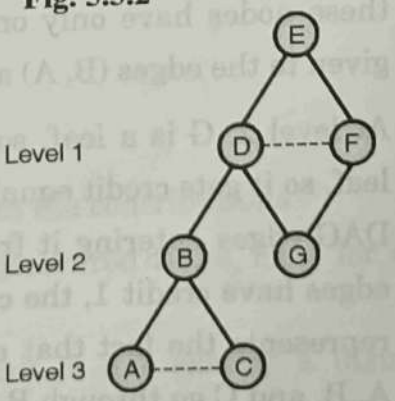


Fig. 5.3.3 : Newman Algorithm

Step II :

Label each node by the number of shortest paths that reach it from the root the sum of the labels of its parents

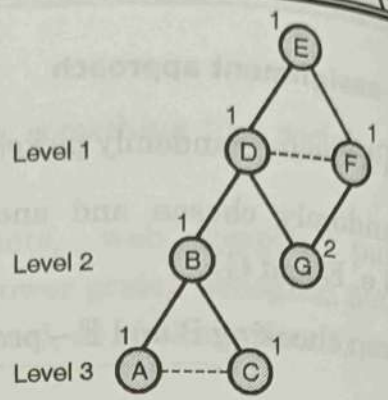


Fig. 5.3.4 : Newman Algorithm

Step III :

- do the calculation starting from the bottom according to rules:
- leaf gets a credit of 1 node that is not a leaf gets a credit equal to 1 plus the sum of the credits of the edges to the level below edge entering node from the level above is given a proportional share of that node

Credit of DAG edges : Let Y_i (I = 1, ...k) be parents of Z, p_i = label (Y_i)

$$\text{credit (Y, Z)} = \frac{\text{credit (Z)} \times p_i}{(p_i + \ldots p_i)}$$

- **Intuition :** a DAG edge Y, Z gets the share of credit of Z proportional to the ≠ of shortest paths from X to Z going through Y,Z
- **Finally :** Repeat Steps 1, 2 and 2 with each node as root. For each edge, betweenness = sum credits obtained in all iterations / 2

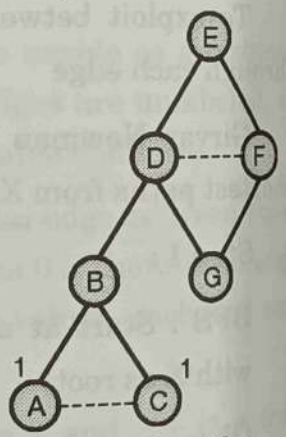


Fig. 5.3.5 : Newman Algorithm

- First, A and C, being leaves, get credit 1. Each of these nodes have only one parent, so their credit is given to the edges (B, A) and (B, C), respectively.
- At level 2, G is a leaf, so it gets credit 1. B is not a leaf, so it gets credit equal to 1 plus the credits on the DAG edges entering it from below. Since both these edges have credit 1, the credit of B is 3. Intuitively 3 represents the fact that all shortest paths from E to A, B, and C go through B. Fig. 5.3.6 shows the credits assigned so far.

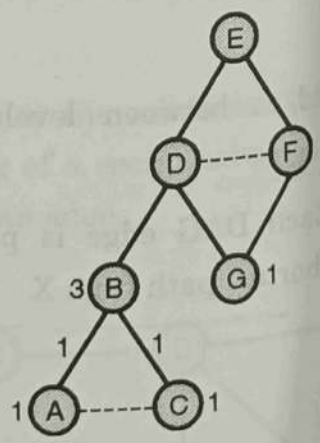


Fig. 5.3.6 : Newman Algorithm – levels 3 and 2

- Now, let us proceed to level 1. B has only one parent, D, so the edge (D,B) gets the entire credit of B, which is 3. However, G has two parents, D and F. We therefore need to divide the credit of 1 that G has between the edges (D, G) and (F, G). In what proportion do we divide? If you examine the labels of Fig. 5.3.4. you see that both D and F have label 1, representing the fact that there is one shortest path from E to each of these nodes.

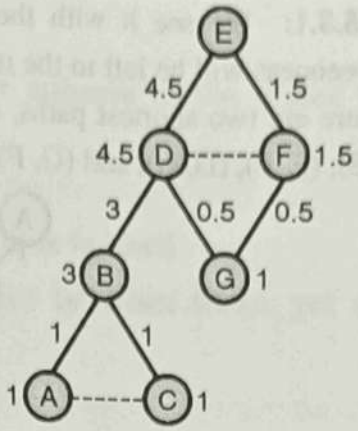


Fig. 5.3.7 : Newman Algorithm

- Thus, we give half the credit of G to each of these edges; i.e., their credit is each $1/(1+1) = 0.5$.
- Had the labels of D and F in Fig. 5.3.7 been 5 and 3, meaning there were five shortest paths to D and only three to F, then the credit of edge (D, G) would have been 5/8 and the credit of edge (F, G) would have been 3/8.
- Now, we can assign credits to the nodes at level 1. D gets 1 plus the credits of the edges entering it from below, which are 3 and 0.5. That is, the credit of D is 4.5. The credit of F is 1 plus the credit of the edge (F, G), or 1.5. Finally, the edges (E, D) and (E, F) receive the credit of D and F, respectively, since each of these nodes has only one parent. These credits are all shown in Fig. 5.3.7.
- The credit on each of the edges in Fig. 5.3.7 is the contribution to the betweenness of that edge due to shortest paths from E. For example, this contribution for the edge (E, D) is 4.5.
- To complete the betweenness calculation, we have to repeat this calculation for every node as the root and sum the contributions. Finally, we must divide by 2 to get the true betweenness, since every shortest path will be discovered twice, once for each of its endpoints.

Using Betweenness to Find Communities

- to complete betweenness calculation:
  - repeat these steps for every node as the root and sum the contributions
  - divide by 2, because every shortest path will be discovered twice, once for each endpoints
- betweenness may behave like a distance measure, but is not exactly a distance measure
- ordering edges by betweenness and removing / adding nodes from graph

Ex. 5.3.1: We see it with the betweenness for each edge in Fig. Ex.5.3.1. The calculation of the betweenness will be left to the reader. The only tricky part of the count is to observe that between E and G there are two shortest paths, one going through D and the other through F. Thus, each of the edges (D, E), (E, F), (D, G), and (G, F) are credited with half a shortest path.

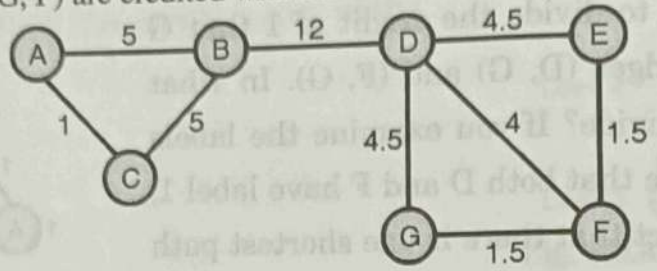


Fig. 5.3.1 : Betweenness scores for the graph

Soln. : Clearly, edge (B,D) has the highest betweenness, so it is removed first. That leaves us with exactly the communities we observed make the most sense, namely: {A, B, C} and {D, E, F, G}. However, we can continue to remove edges.

Next to leave are (A, B) and (B, C) with a score of 5, followed by (D, E) and (D, G) with a score of 4.5. Then, (D, F), whose score is 4, would leave the graph. We see in Fig. Ex. 5.3.1 (a) the graph that remains.

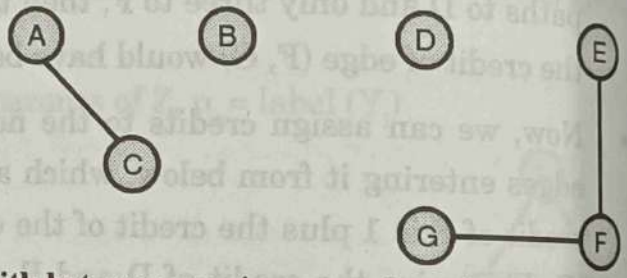


Fig. Ex. 5.3.1 (a) : All the edges with betweenness 4 or more have been removed

The "communities" of Fig. Ex. 5.3.1 (a) look strange. One implication is that A and C are more closely knit to each other than to B. That is, in some sense B is a "traitor" to the community {A, B, C} because he has a friend D outside that community. Likewise, D can be seen as a "traitor" to the group {D, E, F, G}, which is why in Fig. Fig. Ex. 5.3.1 (a) , only E, F, and G remain connected.

5.3.3 Direct Discovery of Communities in a social graph

UQ. Explain the Clique Percolation Method (CPM) used in direct discovery of communities in a social graph with example. **(MU - Dec. 18, 10 Marks)**

UQ. What is a "Community" in a social network Graph? **(MU - May 18, 5 Marks)**

UQ. What is a "Community" in a social Network Graph? Explain any one algorithm for finding communities in asocial graph. **(MU - Dec. 17, 10 Marks)**

Discovering communities directly by looking for subsets of the nodes that have a relatively large number of edges among them. Interestingly, the technique for doing this search on a large graph involves finding large frequent item sets

Finding Cliques

We want to discover communities directly by looking for subsets of the nodes that have a relatively large number of edges among them

- first thought-finding a large clique
- NP-complete problem, even approximating the maximal clique is hard

It is possible to have a set of nodes with almost all edges between them, yet with relatively small cliques

Complete Bipartite Graphs

- Graph that consists of 's' nodes on one side and 't' nodes on the other side with all 'st' possible edges between them
- It is possible to guarantee that a bipartite graph with many edges has a large complete bipartite sub graph (unlike cliques)
- It might be regarded as the nucleus of community

Finding Complete Bipartite Sub graphs

we are given a large bipartite graph G , and we want to find instances of Ks,t within it. It is similar to finding frequent item sets. Items" on the left side (Ks,t – t nodes there) assumption that $t \leq s$ The baskets" on the right side. member of the basket are the nodes from left side connected to that node. support threshold s frequent item set of size t and s of the baskets, in which all those items appear, form an instance of Ks,t

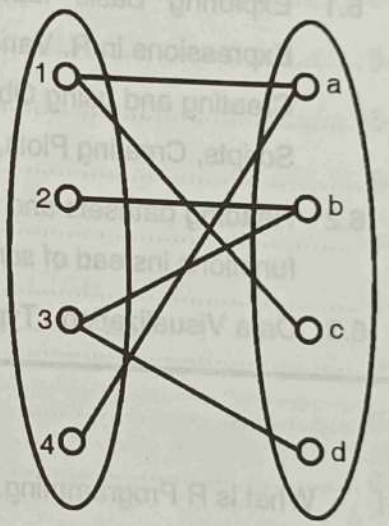


Fig. 5.3.7 : The bipartite graph

Why Complete Bipartite Graphs Must Exist

If there is a community with n nodes and average degree d, then this community is guaranteed to have a complete bipartite sub graph Ks,t when:

$$n\frac{\binom{d}{t}}{\binom{n}{t}} \geq s$$

which approximately is :

$$n\left(\frac{d}{n}\right) \geq s$$

Chapter Ends...

❑❑❑

MODULE VI

CHAPTER 6

Data Analytics with R

University Prescribed Syllabus w.e.f Academic Year 2022-2023

6.1 Exploring Basic features of R, Exploring RGUI, Exploring RStudio, Handling Basic Expressions in R, Variables in R, Working with Vectors, Storing and Calculating Values in R, Creating and using Objects, Interacting with users, Handling data in R workspace, Executing Scripts, Creating Plots, Accessing help and documentation in R

6.2 Reading datasets and Exporting data from R, Manipulating and Processing Data in R, Using functions instead of script, built-in functions in R

6.3 Data Visualization : Types, Applications

6.1 WHAT IS R PROGRAMMING

- R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was initially written by **Ross Ihaka** and **Robert Gentleman** at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.
- The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.
- It includes machine learning algorithms, linear regression, time series, and statistical inference.. Most of the R libraries are written in R, but for heavy computational tasks, C, C++ and Fortran codes are preferred.
- It is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on.
- Data analysis with R is done in a series of steps; programming, transforming, discovering, modeling and communicate the results

6.2 WHY R PROGRAMMING ?

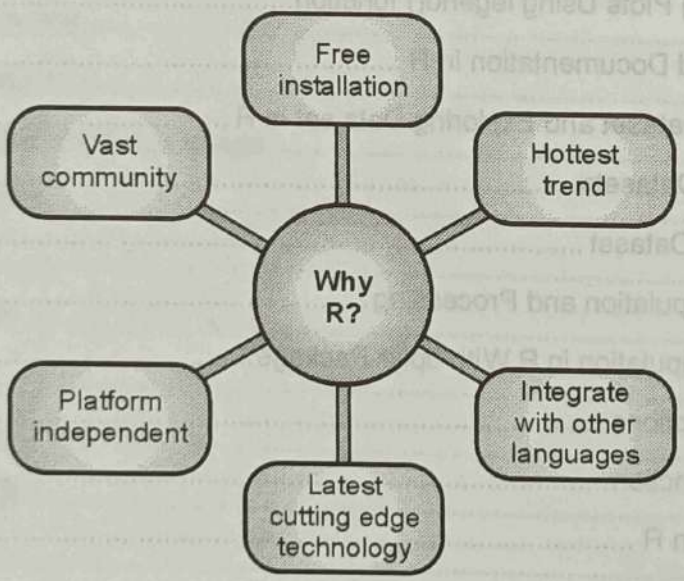


Fig. 6.2.1

GQ. What is R ? Describe Programming features of R language?

6.3 FEATURES OF R PROGRAMMING LANGUAGE

- R programming is used as a leading tool for machine learning, statistics, and data analysis. Objects, functions, and packages can easily be created by R.
- It's a platform-independent language. That means it can be applied to all operating system.
- It's an open-source free language. That means anyone can install it in any organization without purchasing a license.
- R programming language is not only a statistic package but also allows us to integrate with other languages (C, C++). Thus, you can easily interact with many data sources and statistical packages.
- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R is currently one of the most requested programming languages in the Data Science job market that makes it the hottest trend nowadays.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

6.4 PROGRAMMING USING R

R is much similar to other widely used languages syntactically, it is easier to code and learn in R. Programs can be written in R in any of the widely used IDE like **R Studio, Rattle, Tinn-R**, etc. After writing the program save the file with the extension **.r**.

6.5 ADVANTAGES OF R

- R is the most comprehensive statistical analysis package. As new technology and concepts often appear first in R.
- R programming language is an open source. Thus, R can run anywhere and at any time.
- R programming language is suitable for GNU/Linux and Windows operating system.

- R programming is cross-platform which runs on any operating system.
- In R, everyone is welcome to provide new packages, bug fixes, and code enhancements.

6.6 DISADVANTAGES OF R

1. In the R programming language, the standard of some packages is less than perfect.
2. R commands give little pressure to memory management. So R programming language may consume all available memory.
3. In R basically, nobody to complain if something doesn't work.
4. R programming language is much slower than other programming languages such as Python and MATLAB.

6.7 APPLICATIONS OF R

1. R is use for Data Science. It gives a broad variety of libraries related to statistics. It also provides the environment for statistical computing and design.
2. R is used by many quantitative analysts as its programming tool. Thus, it helps in data importing and cleaning.
3. R is the most prevalent language. So many data analysts and research programmers use it. Hence, it is used as a fundamental tool for finance.
4. Tech giants like Google, Facebook, bing, Twitter, Accenture, Wipro and many more using R nowadays.

6.8 INSTALLATION OF R BASE AND R STUDIO

- **Step 1 :** Download R from http://cran.us.r-project.org/.
- **Step 2 :** Click on Download R for Windows. Click on base. Click on Download R 4.2.1 for Windows (or a newer version that appears).
- **Step 3 :** Install R. Leave all default settings in the installation options.
- **Step 4 :** Download RStudio Desktop for windows from http://rstudio.org/download/desktop (it should be called something like Download RStudio for Windows 2022.02.3 + 492 — Windows Vista/7/8/10). Choose default installation options.
- **Step 5 :** Open RStudio.

6.8.1 Basics of R

6.8.1.1 RGUI

RGUI, the standard R user interface, is a simple interface to the R language, with some menus and toolbars, as well as a number of windows; when you start R, the Console window is displayed.

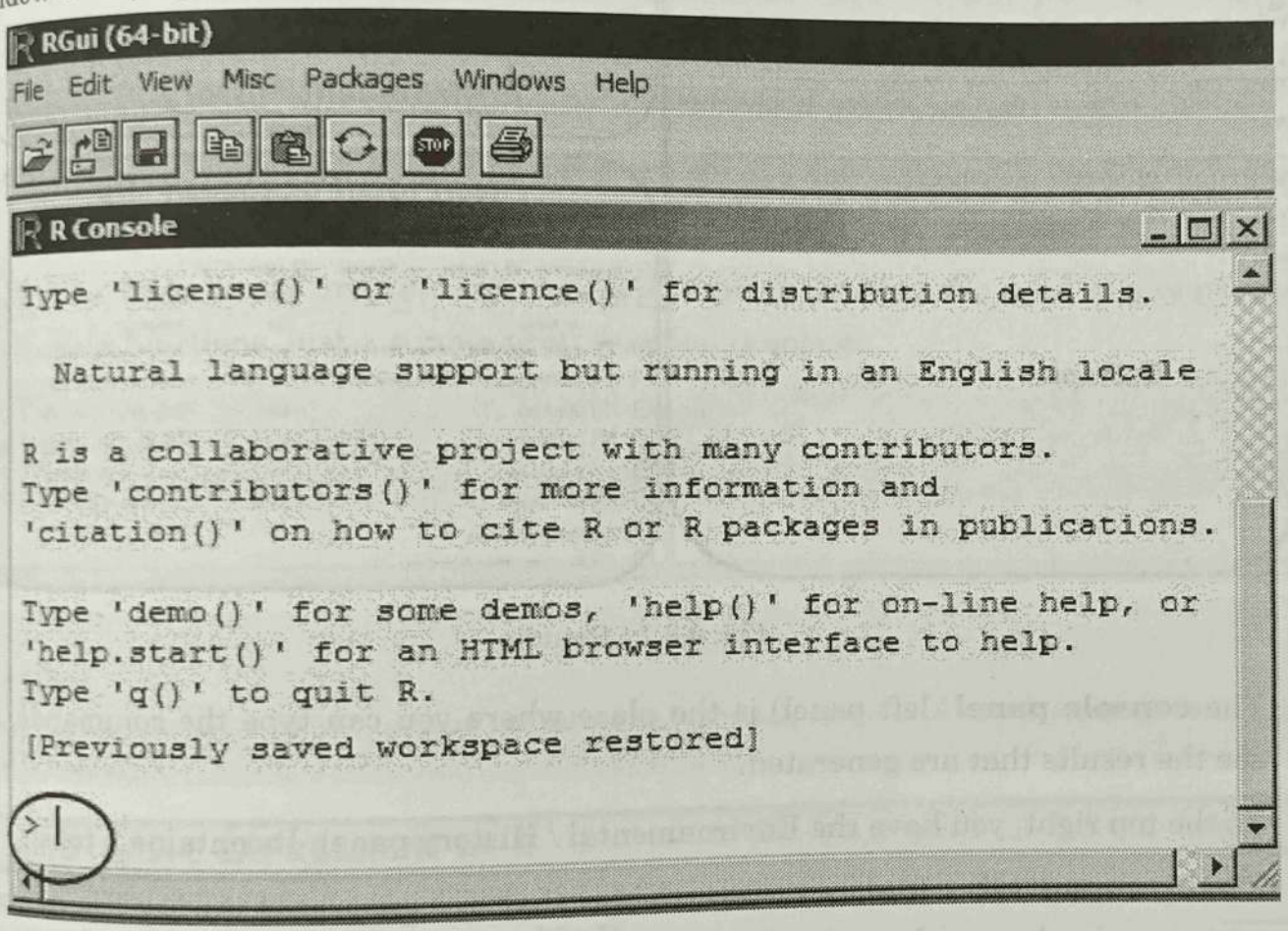


Fig. 6.8.1 : RGUI

6.8.1.2 R Studio

RStudio is an integrated development environment which allows us to interact with R more readily. RStudio is similar to the standard RGui, but it is considered more user-friendly.

- R Studio is available as both Open source and Commercial software.
- R Studio is also available as both Desktop and Server versions.
- R Studio is also available for various platforms such as Windows, Linux, and macOS.

After the installation process is over, the R Studio interface looks like :

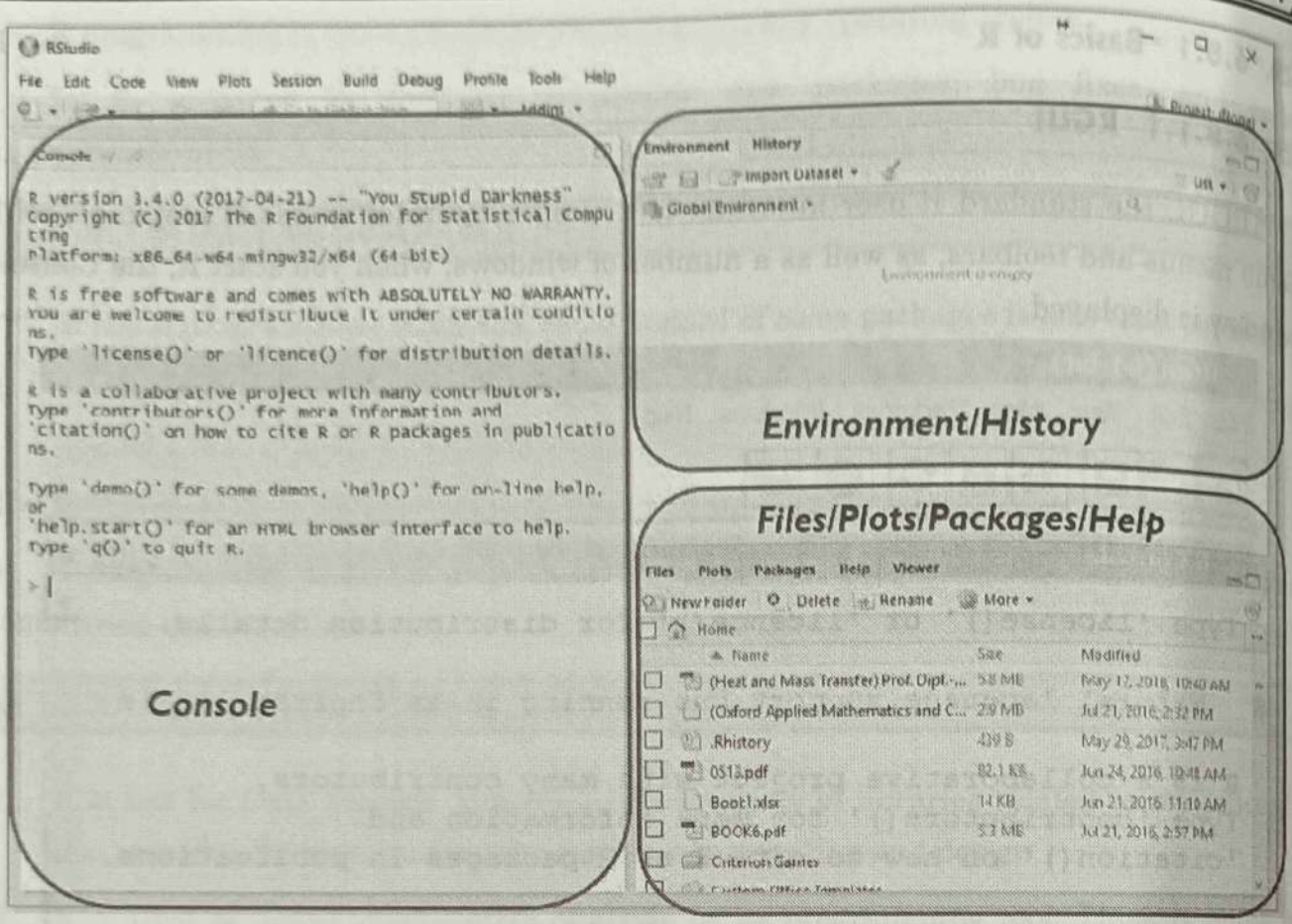


Fig. 6.8.2 : Rstudio

- The **console panel** (left panel) is the place where you can type the commands, and see the results that are generated.
- To the top right, you have the Environmental / History panel. It contains 2 tabs:
- **Environment tab :** It shows the variables that are generated during the course of programming in a workspace that is temporary.
- **History tab :** In this tab, you'll see all the commands that are used till now from the start of usage of R Studio.
- To the right bottom, you have another panel, which contains multiple tabs, such as files, plots, packages, help, and viewer.
- The Files tab shows the files and directories that are available within the default workspace of R.
- The Plots tab shows the plots that are generated during the course of programming.
- The Packages tab helps you to look at what are the packages that are already installed in the R Studio and it also gives a user interface to install new packages.

- The Help tab is the most important one where you can get help from the R Documentation on the functions that are in built-in R.
- The final and last tab is that the Viewer tab which can be used to see the local web content that's generated using R.

6.9 SYNTAX OF R PROGRAMMING

- R Programming is a very popular programming language which is broadly used in data analysis. The way in which we define its code is quite simple.
- The "Hello World!" is the basic program for all the languages, and now we will understand the syntax of R programming with "Hello world" program.
- For the HelloWorld program, we just need a print function. No need of any packages or main functions, just a simple print function is enough.

```
print("Hello World")
```

Result

```
Hello World
```

print() is a function which is used to print the values on to the output screen. It also has arguments, we can use it if needed.

6.10 BASIC EXPRESSION IN R

Simple expression can be executed in R.

Example of adding two numbers

```
num <- 1
num2 <- 4
sum <- num + num2
sum
```

Output

```
[1] 5
```

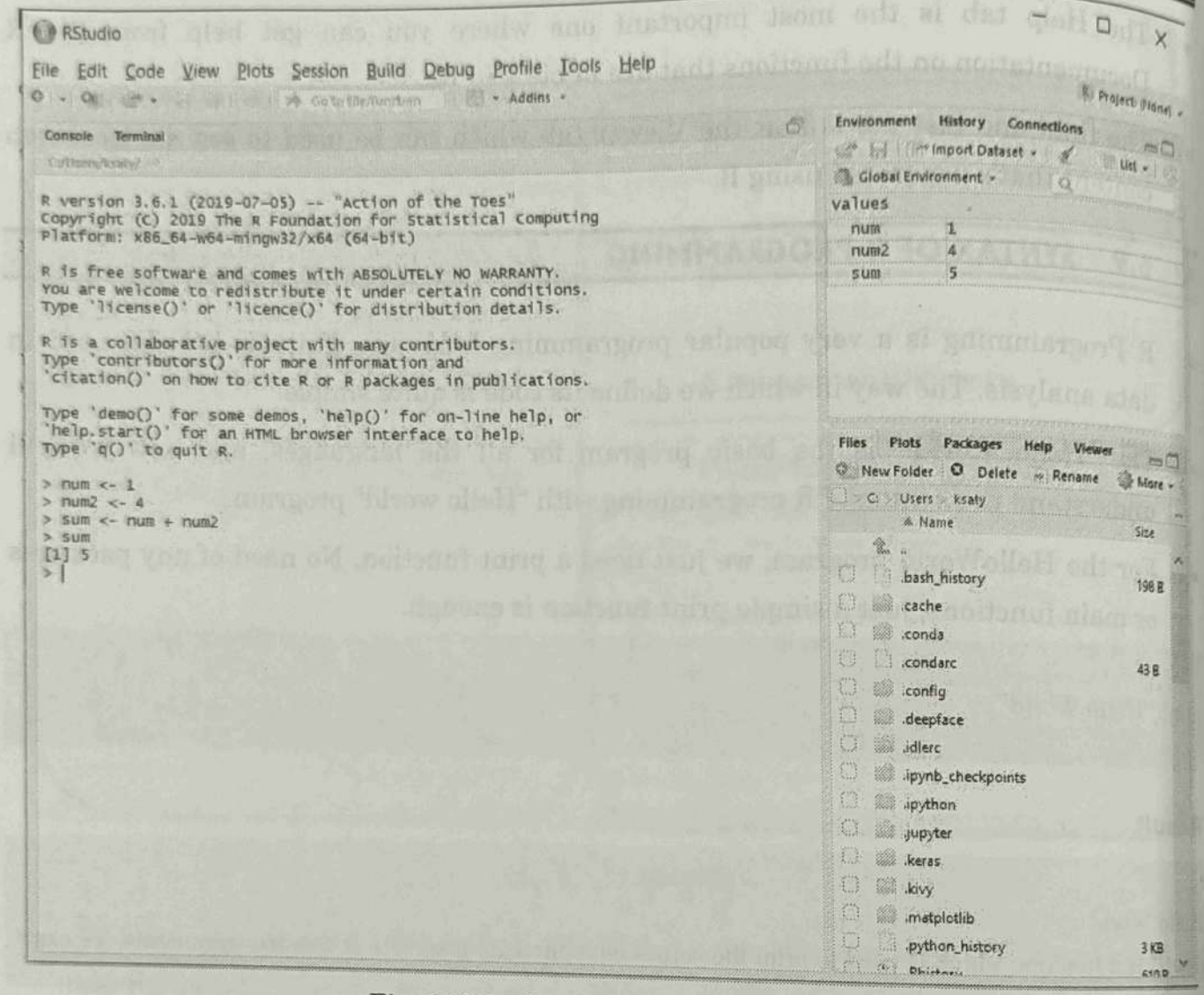


Fig. 6.10.1 : Simple expression execution

Subtraction

```
2-2#Substraction
## [1] 0
```

Multiplication & Division

```
2*2# Multiplication
## [1] 4
2/2# Division
## [1] 1
```

Squaring a number

```
2**2
## [1] 4
```

Square Root

```
2^(1/2)
## [1] 1.414214
```

Alternatively you can use sqrt function

```
sqrt(2)
## [1] 1.414214
```

Integer Division- Returns Integer after division

```
5%/%2
## [1] 2
```

Modulus- Returns Remainder after division

```
5%%2
## [1] 1
```

6.10.1 R Data Types

- In R, there are several data types such as integer, string, etc.
- The operating system allocates memory based on the data type of the variable and decides what can be stored in the reserved memory. Below table give detail about different data types

Data Type	Example	Description
Logical	True, False	It is a special data type for data with only two possible values which can be construed as true/false.
Numeric	11,46,124,5935	Decimal value is called numeric in R, and it is the default computational data type.
Integer	8L, 66L, 2386L	Here, L tells R to store the value as an integer
Complex	Z = 1 + 3i, t = 7 + 6i	A complex value in R is defined as the pure imaginary value i.
Character	'a', '"good"', "TRUE", '55.4'	In R programming, a character is used to represent string values. We convert objects into character values with the help of as.character() function.
Raw		A raw data type is used to holds raw bytes.

GQ. List and Explain Data types of R ?

6.11 VARIABLES IN R

- A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many Robjects.
- A valid variable name consists of letters, numbers and the dot or underline characters.
- The variable name starts with a letter or the dot not followed by a number. Below table gives some example of valid and invalid variable name

Variable Name	Validity	Reason
var_name2.	Valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	Invalid	Starts with a number
.var_name, var.name	Valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	Invalid	The starting dot is followed by a number making it invalid.
_var_name	Invalid	Starts with _ which is not valid

6.11.1 Variable Assignment

- The variables can be assigned values using leftward, rightward and equal to operator.
- The values of the variables can be printed using **print()** or **cat()** function.
- The **cat()** function combines multiple items into a continuous print output.

GQ. Explain variable assignment in R along with example ?

Example of R Programming Variable Assignment

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)
```

```
# Assignment using leftward operator.
var.2 <- c("learn","R")
# Assignment using rightward operator.
c(TRUE,1) -> var.3
print(var.1)
cat ("var.1 is ", var.1 ,"\n")
cat ("var.2 is ", var.2 ,"\n")
cat ("var.3 is ", var.3 ,"\n")
```

Result

```
[1] 0 1 2 3
var.1 is 0 1 2 3
var.2 is learn R
var.3 is 1 1
```

6.11.2 Data Type of a Variable

In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. So R is called a dynamically typed language, which means that we can change a variable's data type of the same variable again and again when using it in a program.

Example of R Programming Data Type of a Variable

```
var_x<- "Hello"
cat("The class of var_x is ",class(var_x),"\n")
var_x<- 34.5
cat(" Now the class of var_x is ",class(var_x),"\n")
var_x<- 27L
cat(" Next the class of var_x becomes ",class(var_x),"\n")
```

Result

```
The class of var_xis  character
 Now the class of var_xis  numeric
 Next the class of var_xbecomes  integer
```

GQ. Explain Data types of R ?

6.12 CREATING AND USING OBJECTS (R DATA STRUCTURES)

- Data structures are very important to understand. Data structure are the objects which we will manipulate in our day-to-day basis in R.
- We can say that everything in R is an object.
- R has many data structures, which include :

1. Vector	2. List	3. Array
4. Matrices	5. Data Frame	6. Factors

GQ. What is vector? Explain vector creation?

1. Vectors

- A vector is the basic data structure in R, or we can say vectors are the most basic R data objects.
- "A vector is a collection of elements which is most commonly of mode character, integer, logical or numeric" When you want to create vector with more than one element, you should use c() function which means to combine the elements into a vector. There are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

1.1 Vector Creation

Single Element Vector

In R when we write just one value, it becomes a vector of length 1 and belongs to one of the above vector types.

Example of vector creation

```
# Atomic vector of type character.
print("xyz");
# Atomic vector of type double.
print(19.5)
# Atomic vector of type integer.
print(69L)
# Atomic vector of type logical.
print(TRUE)
# Atomic vector of type complex.
print(2+4i)
# Atomic vector of type raw.
print(charToRaw('hello'))
```

Result

```
[1] "xyz"
[1] 19.5
[1] 69
[1] TRUE
[1] 2+4i
[1] 68 65 6c 6c 6f
```

1.2 Using the c() function

The non-character values are coerced to character type if one of the elements is a character.

Example of R Programming Using the c() function

```
# The logical and numeric values are converted to characters.
s <- c('apple','red',5,TRUE)
print(s)
```

result

```
[1] "apple" "red"   "5"     "TRUE"
```

1.4 Accessing Vector Elements

- Elements of a Vector are accessed using indexing. The **[ ] brackets** are used for indexing.
- Indexing starts with position 1. Giving a negative value in the index drops that element from result. **TRUE**, **FALSE** or **0** and **1** can also be used for indexing.

Example of R Programming Accessing Vector Elements

```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)
# Accessing vector elements using logical indexing.
v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
print(v)
# Accessing vector elements using negative indexing.
x <- t[c(-2,-5)]
print(x)
# Accessing vector elements using 0/1 indexing.
y <- t[c(0,0,0,0,0,0,1)]
print(y)
```

Result

```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
```

2. Lists

- A list in R is a generic object consisting of an ordered collection of objects. Lists are one-dimensional, heterogeneous data structures. The list can be a list of vectors, a list of matrices, a list of characters and a list of functions, and so on.
- A list is a vector but with heterogeneous data elements. A list in R is created with the use of list() function. R allows accessing elements of a list with the use of the index value. In R, the indexing of a list starts with 1 instead of 0 like other programming languages.
- To create a List in R you need to use the function called "list()". In other words, a list is a generic vector containing other objects.
- We want to build a list of employees with the details. So for this, we want attributes such as ID, employee name, and the number of employees.

```
e.gempList = list(empId, empName, numberOfEmp)
```

3. Arrays

- While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension.
- In R, an array is created with the help of **array()** function. This function takes a vector as an input and uses the value in the dim parameter to create an array.

```
e.g. a <- array(c('green','yellow'),dim = c(3,3,2))
```

4. Matrices

- A matrix is an R object in which the elements are arranged in a two-dimensional rectangular layout.
- In the matrix, elements of the same atomic types are contained. For mathematical calculation, this can use a matrix containing the numeric element. A matrix is created with the help of the matrix() function in R.

Syntax

The basic syntax of creating a matrix is as follows :

```
1. matrix(data, nrow, ncol, by_row, dim_name)
```

data: input vector which becomes data element of matrix
nrow and ncol: number of rows and coumns
by_row: logical value. If true then input vector elements arranged by rows else by columns.
Dim_name: name assigned to rows and columns.

```
e.g. M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
```

5. Factors

- These are also data objects that are used to categorize the data and store it as levels. Factors can store both strings and integers.
- Columns have a limited number of unique values so that factors are very useful in columns. It is very useful in data analysis for statistical modeling.
- Factors are created with the help of **factor()** function by taking a vector as an input parameter.

E.g.

```
# Create a vector.
apple_colors <- c('green','green','yellow','red','red','red','green')
# Create a factor object.
factor_apple <- factor(apple_colors)
```

6. Data Frames

- Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data.
- The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.
- Data Frames are created using the **data.frame()** function.

E.g.

```
# Create the data frame.
BMI <- data.frame(
gender = c("Male", "Male","Female"),
height = c(152, 171.5, 165),
weight = c(81,93, 78),
Age = c(42,38,26)
)
```

6.13 STORING AND CALCULATING VALUES

- We can extend the calculator function to create variables to store the information. We can do the following :

```
x <-4
y <-9
z <-sqrt(x + y)
```

- With the above R code, you would *assign* (using the <- operator) x to have the value of 4; y to have the value of 9; and z to have the result of the square root of the sum of x + y
- If you type x in the console at the *ready* prompt, you see it is assigned the value of 4.
- Similarly if you type z in the console at the *ready* prompt, you see the result of that calculation.

6.14 INTERACTING WITH USERS

GQ. Explain methods for accepting input from user ?

- Developers often have a need to interact with users, either to get data or to provide some sort of result.
- Most programs today use a dialog box as a way of asking the user to provide some type of input.
- Like other programming languages in R it's also possible to take input from the user. For doing so, there are two methods in R.

1. Using readline() method
2. Using scan() method

1. Readline Method

- In R language **readline()** method takes input in string format. If one inputs an integer then it is inputted as a string, lets say, one wants to input **255**, then it will input as **"255"**, like a string. So one needs to convert that inputted value to the format that he needs.
- In this case, string **"255"** is converted to integer 255. To convert the inputted value to the desired data type, there are some functions in R,

```
as.integer(n); —> convert to integer
as.numeric(n); —> convert to numeric type (float, double etc)
```

```
as.complex(n); —> convert to complex number (i.e 3+2i)
as.Date(n) —> convert to date ..., etc
```

Syntax

```
var = readline();
var = as.integer(var);
Note that one can use "<-" instead of "="
```

Example

```
# R program to illustrate
# taking input from the user
 # taking input using readline()
# this command will prompt you
# to input a desired value
var = readline();
 # convert the inputted value to integer
var = as.integer(var);
 # print the value
print(var)
```

Result

```
255
[1] 255
```

Taking multiple inputs in R

- Taking multiple inputs in R language is same as taking single input, just need to define multiple **readline()** for inputs.
- One can use **braces** for define multiple **readline()** inside it.

Syntax

```
var1 = readline("Enter 1st number : ");
var2 = readline("Enter 2nd number : ");
var3 = readline("Enter 3rd number : ");
var4 = readline("Enter 4th number : ");
or,
{
    var1 = readline("Enter 1st number : ");
    var2 = readline("Enter 2nd number : ");
    var3 = readline("Enter 3rd number : ");
    var4 = readline("Enter 4th number : ");
}
```

```
# R program to illustrate
# taking input from the user

# taking multiple inputs
# using braces
{
    var1 = readline("Enter 1st number : ");
    var2 = readline("Enter 2nd number : ");
    var3 = readline("Enter 3rd number : ");
    var4 = readline("Enter 4th number : ");
}

# converting each value
var1 = as.integer(var1);
var2 = as.integer(var2);
var3 = as.integer(var3);
var4 = as.integer(var4);

# print the sum of the 4 number
print(var1 + var2 + var3 + var4)
```

Result

```
Enter 1st number : 12
Enter 2nd number : 13
Enter 3rd number : 14
Enter 4th number : 15
[1] 54
```

2. Using scan() method

- Another way to take user input in R language is using a method, called **scan()** method. This method takes input from the console.
- This method is a very handy method while inputs are needed to taken quickly for any mathematical calculation or for any dataset. This method reads data in the form of a vector or list. This method also uses to reads input from a file also.

Example

This is simple method to take input using **scan()** method, where some integer number is taking as input and print those values in the next line on the console.

```
# R program to illustrate
# taking input from the user
```

```
# taking input using scan()
x = scan()
# print the inputted values
print(x)

x = scan()
```

Result

```
1: 1 2 3 4 5
6: 3 6 7
9:
Read 8 items
[1] 1 2 3 4 5 3 6 7
```

scan() method is taking input continuously, to terminate the input process, need to press Enter key 2 times on the console.

6.15 HANDLING DATA IN R WORKSPACE

- The workspace is your current R working environment and includes any user-defined objects (vectors, matrices, functions, data frames, or lists). At the end of an R session, you can save an image of the current workspace that's automatically reloaded the next time R starts.
- Commands are entered interactively at the R user prompt. You can use the up and down arrow keys to scroll through your command history.
- Doing so allows you to select a previous command, edit it if desired, and resubmit it using the Enter key.
- The current working directory is the directory R will read files from and save results to by default. You can find out what the current working directory is by using the getwd() function.
- You can set the current working directory by using the setwd() function. If you need to input a file that isn't in the current working directory, use the full pathname in the call.
- Always enclose the names of files and directories from the operating system in quote marks.

- Some standard commands for managing your workspace are listed below

Functions	Description
getwd()	List the current working directory
setwd("mydirectory")	Change the current working directory to mydirectory .
ls()	List the objects in the current workspace.
rm(a, b, ..)	Removes the objects a, b... from your workspace
rm(list = ls())	Removes all objects in your workspace
list.files()	Returns the names of all files in the working directory
help(options)	Learn about available options.
options()	View or set current options.
savehistory("myfile")	Save the commands history to *myfile* (default =.Rhistory).
loadhistory("myfile")	Reload a command's history (default = .Rhistory).
save.image("myfile")	Save the workspace to myfile (default = .RData).
save(objectlist,file="myfile")	Save specific objects to a file.
load("myfile")	Load a workspace into the current session (default =.RData).
q()	Quit R. You'll get a prompt to save the workspace.

6.15.1 R scripts

GQ. What is the use of R scripts ? Explain process of R scripting ?

6.15.2 Why R scripts

- While entering and running your code at the R command line is effective and simple. This technique has its limitations.
- Each time you want to execute a set of commands, you have to re-enter them at the command line.

Complex commands are potentially subject to typographical errors, necessitating that they be re-entered correctly. Repeating a set of operations requires re-entering the code stream.

Fortunately, R and RStudio provide a method to mitigate these issues. R scripts are that solution.

6.15.3 About R scripts

A script is simply a text file containing a set of commands and comments. The script can be saved and used later to re-execute the saved commands.

The script can also be edited so you can execute a modified version of the commands.

6.15.4 Creating an R script

It is easy to create a new script in RStudio. You can open a new empty script by clicking the New File icon in the upper left of the main RStudio toolbar. This icon looks like a white square with a white plus sign in a green circle.

Clicking the icon opens the New File Menu. Click the R Script menu option and the script editor will open with an empty script.

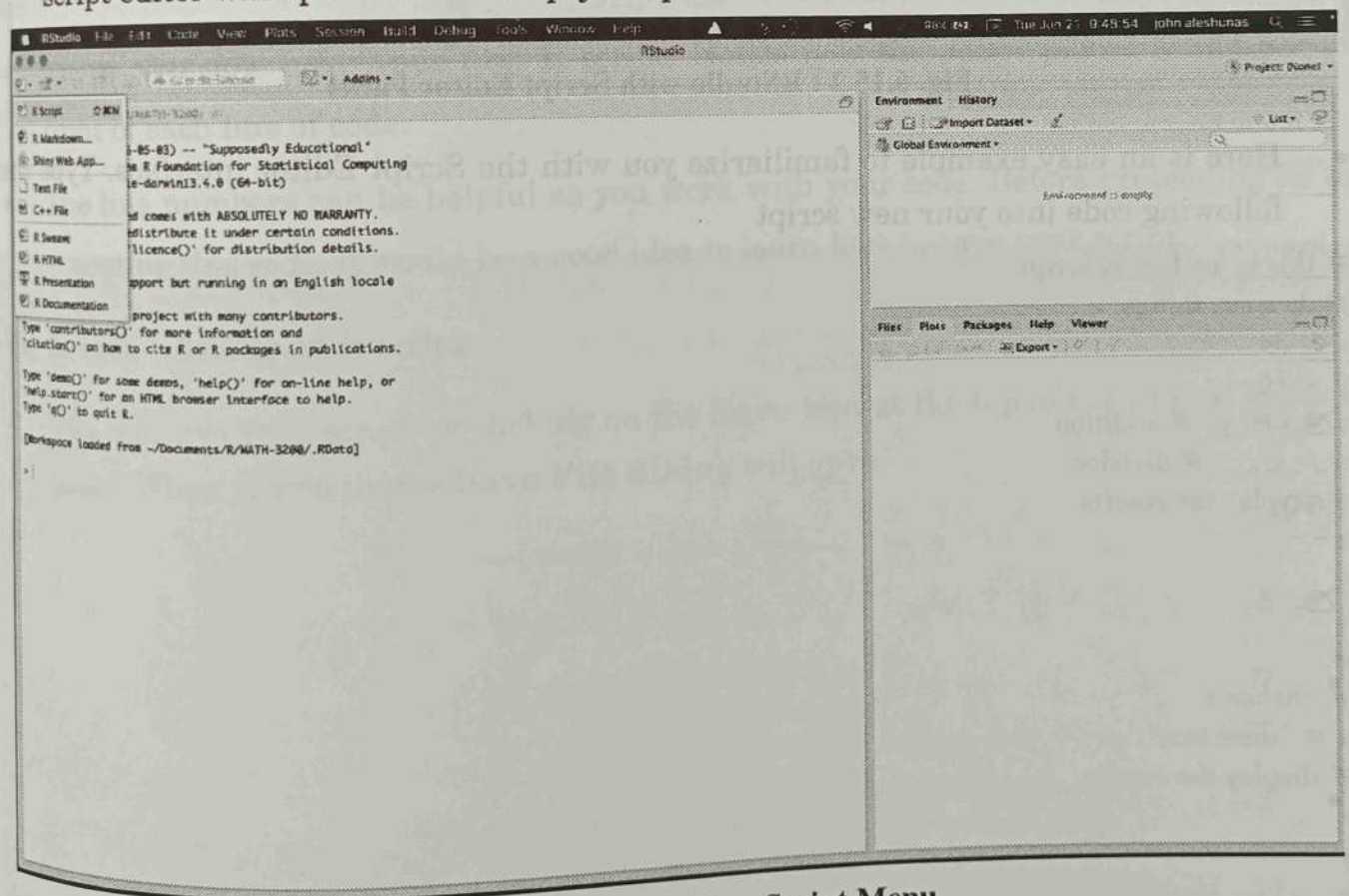


Fig. 6.15.1 : RStudio New Script Menu

- Once the new script opens in the Script Editor panel, the script is ready for text entry, and your RStudio session will look like this.

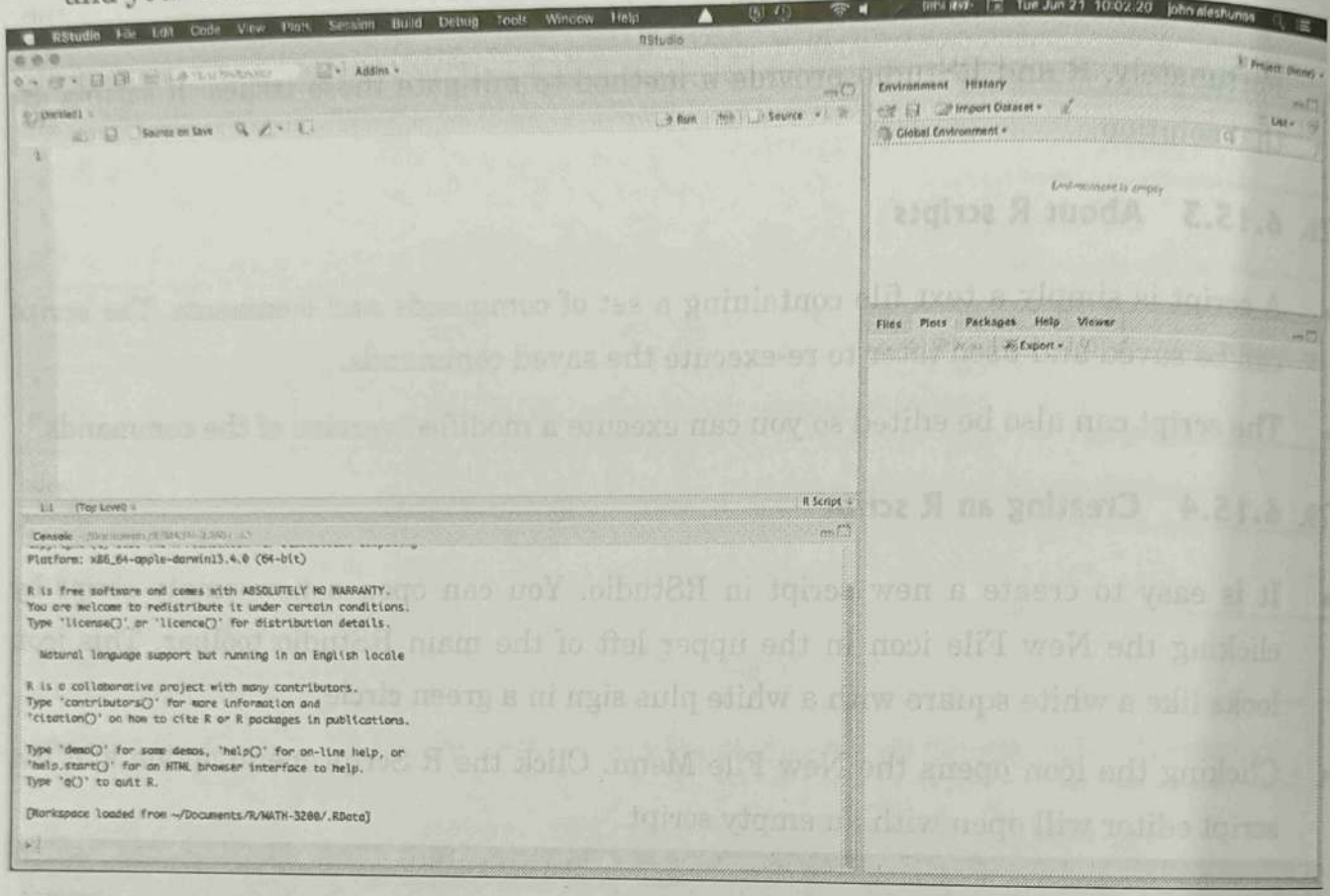


Fig. 6.15.2 : RStudio with Script Editor Panel

- Here is an easy example to familiarize you with the Script Editor interface. Type the following code into your new script

```
# this is my first R script
# do some things
x = 34
y = 16
z = x + y   # addition
w = y/x     # division
# display the results
x
y
z
w
# change x
x = "some text"
# display the results
x
y
z
w
```

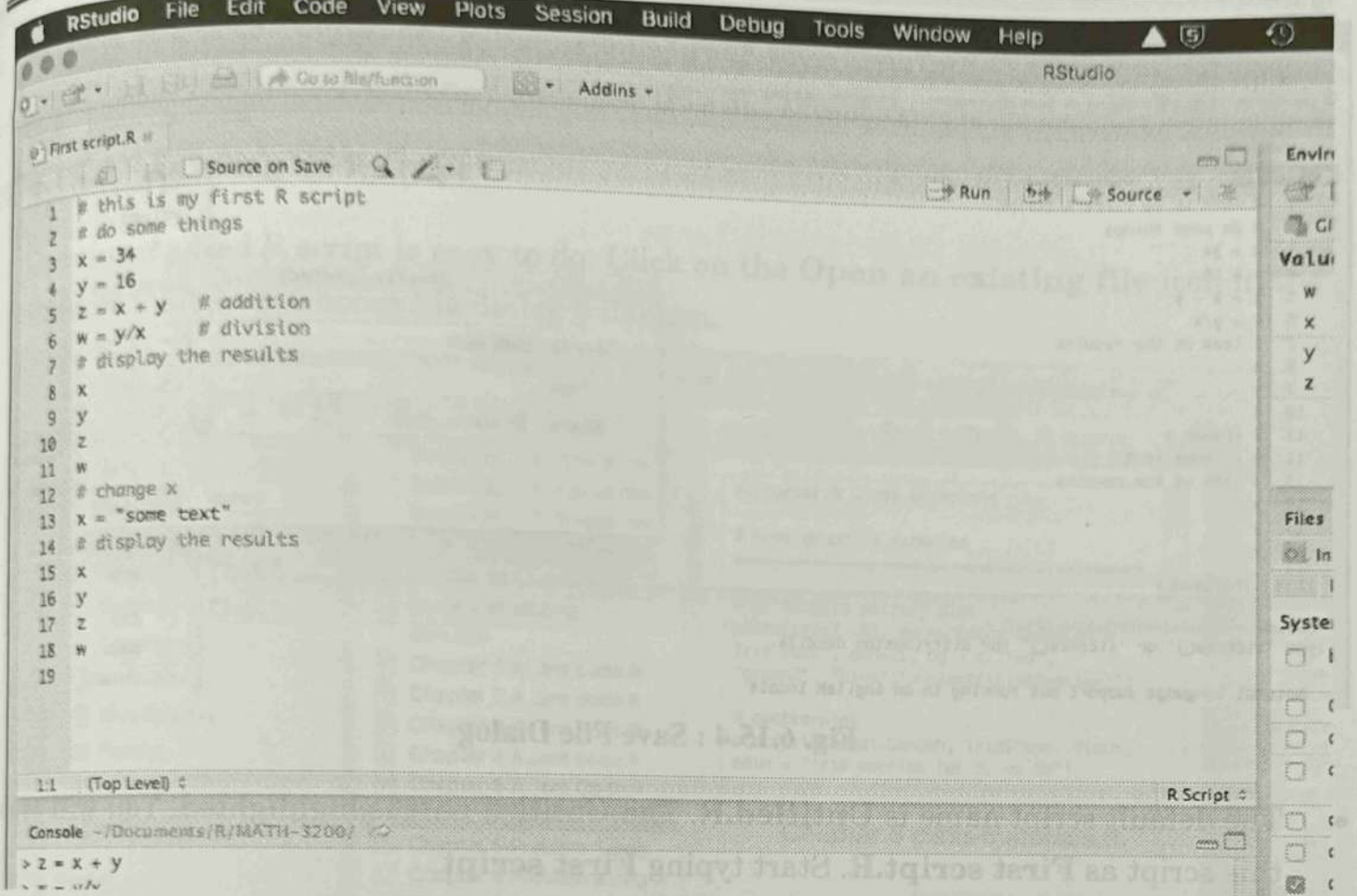


Fig. 6.15.3 : R Script Example

- There, you now have your first R script. Notice how the editor places a number in front of each line of code.
- The line numbers can be helpful as you work with your code. Before proceeding on to executing this code, it would be a good idea to learn how to save your script.

6.15.5 Saving an R script

- You can save your script by clicking on the **Save** icon at the top of the **Script Editor** panel. When you do that, a **Save File dialog** will open.

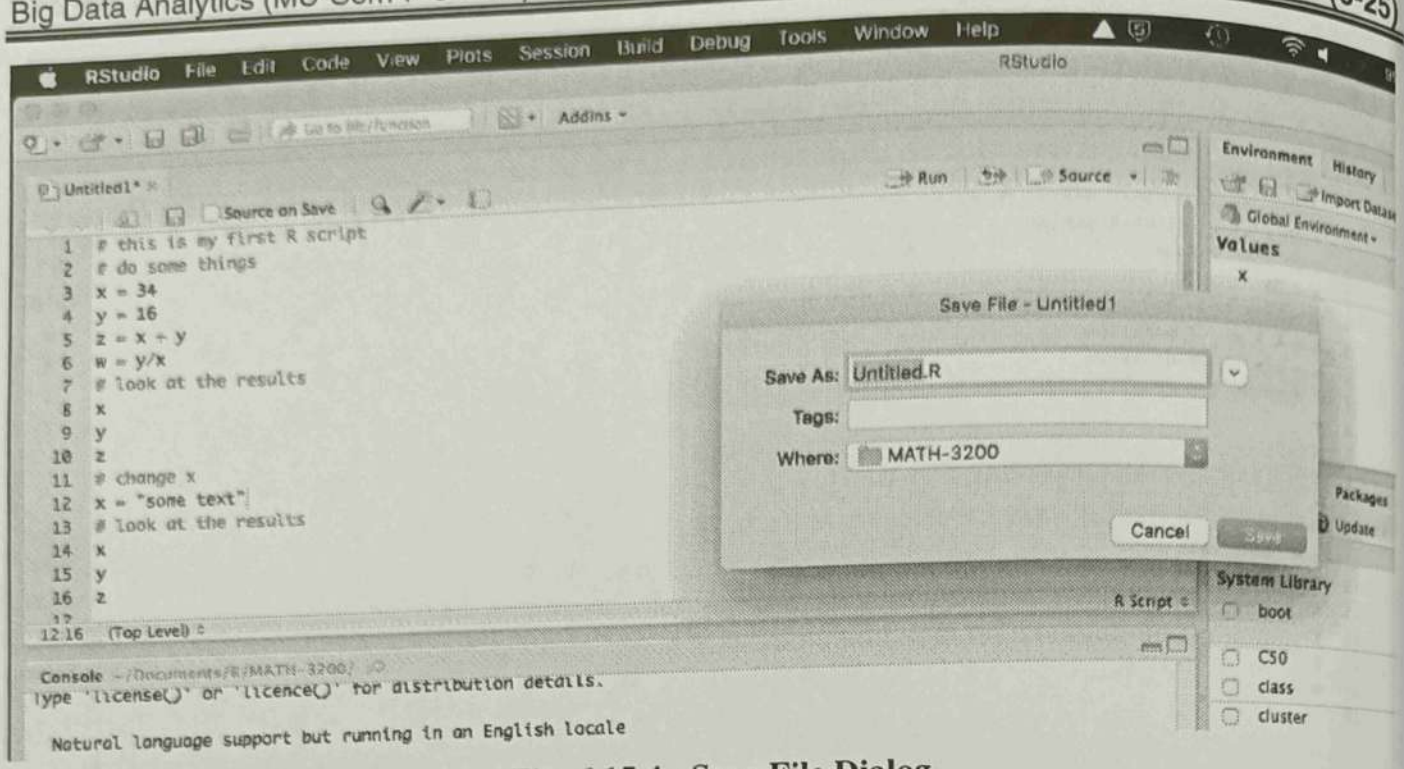


Fig. 6.15.4 : Save File Dialog

- The default script name is **Untitled.R**. The Untitled part is highlighted. You will save this script as **First script.R**. Start typing **First script**.
- RStudio overwrites the highlighted default name with your new name, leaving the .R file extension. The Save File dialog should now look like this.

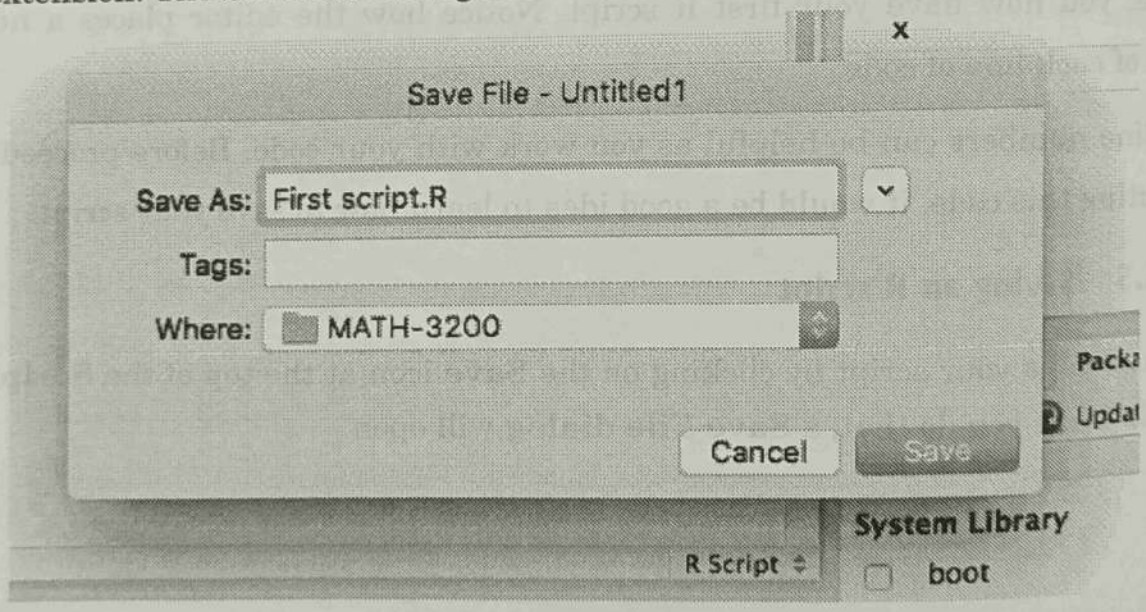


Fig. 6.15.5 : Save First script.R

- Notice that RStudio will save your script to your current working folder. Press the **Save** button and your script is saved to your working folder.
- Notice that the name in the file tab at the top of the Script Editor panel now shows your saved script file name.

It is not necessary to use an **.R** file extension for your R scripts, it does make it easier for RStudio to work with them if your use this file extension.

6.15.6 Opening an R script

Opening a saved R script is easy to do. Click on the **Open an existing file** icon in the RStudio toolbar. A Choose file dialog will open.

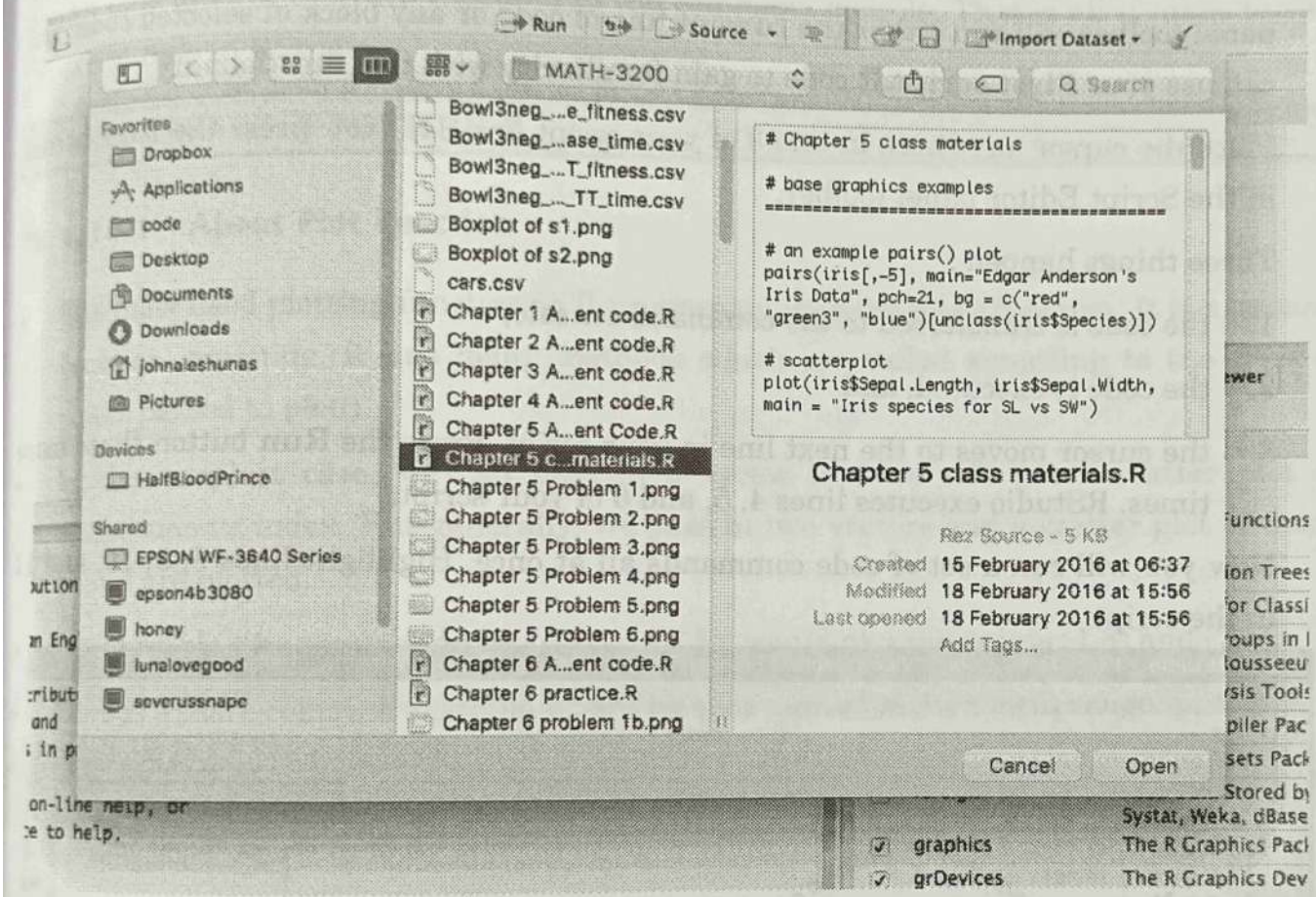


Fig. 6.15.6 : RStudio Open Script Dialog

Select the R script you want to open [this is one place where the .R file extension comes in handy] and click the **Open** button. Your script will open in the Script Editor panel with the script name in an editor tab.

We will use the script you created above [**First script.R**] for this exercise. First, you will need to close the script.

You can close this script by clicking the **X** in the right side of the editor tab where the script name appears.

Since you only had one script open, when you close **First script.R**, the Script Editor panel disappears.

- Now, click on the **Open an existing file** icon in the RStudio toolbar. The Choose file dialog will open. Select **First script.R** and then press the **Open** button in the dialog. Your script is now open in the Script Editor panel and ready to use.

6.15.7 Executing Code in an R Script

- You can run the code in your R script easily. The **Run** button in the Script Editor panel toolbar will run either the current line of code or any block of selected code. You can use your **First script.R** code to gain familiarity with this functionality.
- Place the cursor anywhere in line 3 of your script [x = 34]. Now press the **Run** button in the Script Editor panel toolbar.
- Three things happen :
  1. the code is transferred to the command console,
  2. the code is executed, and
  3. the cursor moves to the next line in your script. Press the **Run** button three more times. RStudio executes lines 4, 5, and 6 of your script.
- Now you will run a set of code commands all at once. Highlight lines 8, 9, 10, and 11 in the script.

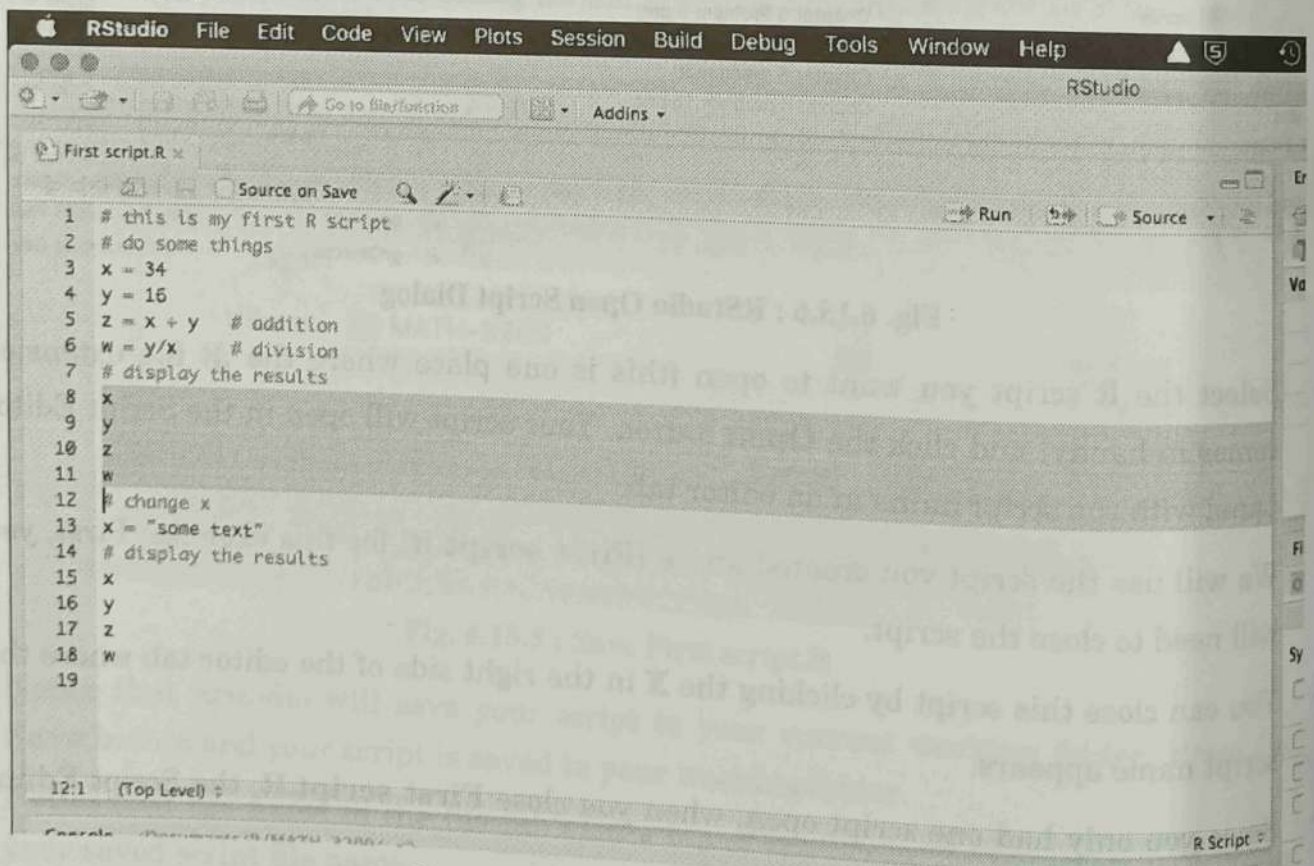


Fig. 6.15.7 : Highlighted Script Code

- Highlighting is accomplished similar to what you may be familiar with in word processor applications.
- You click your left mouse button and the beginning of the text you want to highlight, you hold the mouse button and drag the cursor to the end of the text and release the button. With those four lines of code highlighted, click the editor **Run** button.
- All four lines of code are executed in the command console. That is all it takes to run script code in RStudio.

6.16 CREATING PLOTS IN R

6.16.1 About Plot Function

- The most used plotting function in R programming is the **plot()** function. It is a generic function, meaning, it has many methods which are called according to the type of object passed to **plot()**.
- In the simplest case, we can pass in a vector and we will get a scatter plot of magnitude vs index. But generally, we pass in two vectors and a scatter plot of these points are plotted.
- For example, the command plot (c(1,2), c(3,5)) would plot the points (1,3) and (2,5).
- Here is a more concrete example where we plot a sine function form range -pi to pi.

```
x <- seq(-pi,pi,0.1)
plot(x, sin(x))
```

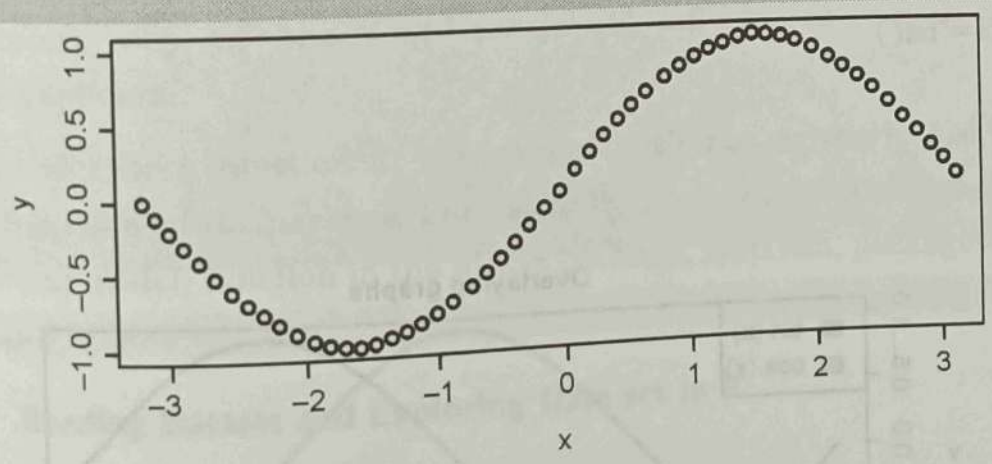


Fig. 6.16.1

6.16.2 Adding Titles and Labeling Axes

We can add a title to our plot with the parameter main. Similarly, xlab and ylab can be used to label the x-axis and y-axis respectively.

```
plot(x, sin(x),
main="The Sine Function",
ylab="sin(x)")
```

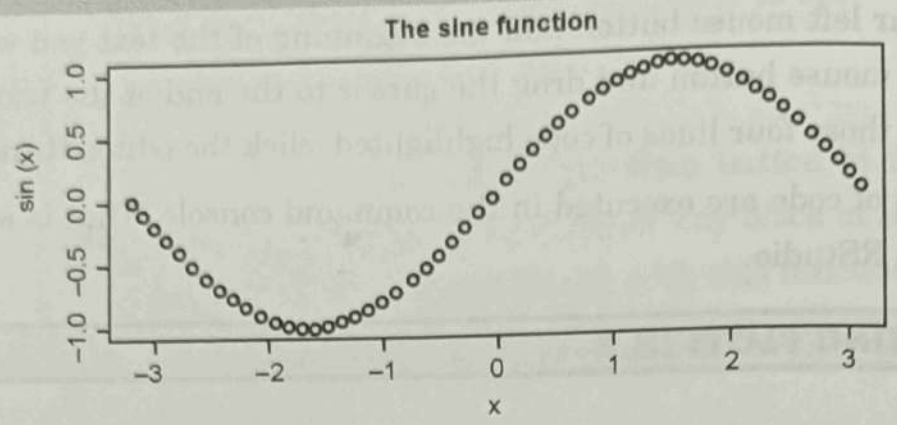


Fig. 6.16.2

6.16.3 Overlaying Plots Using legend() function

- Calling plot() multiple times will have the effect of plotting the current graph on the same window replacing the previous one.
- However, sometimes we wish to overlay the plots in order to compare the results.
- This is made possible with the functions lines() and points() to add lines and points respectively, to the existing plot.

```
plot(x, sin(x),
main="Overlaying Graphs",
ylab="",
type="l",
col="blue")
lines(x,cos(x), col="red")
legend("topleft",
c("sin(x)","cos(x)"),
fill=c("blue","red")
)
```

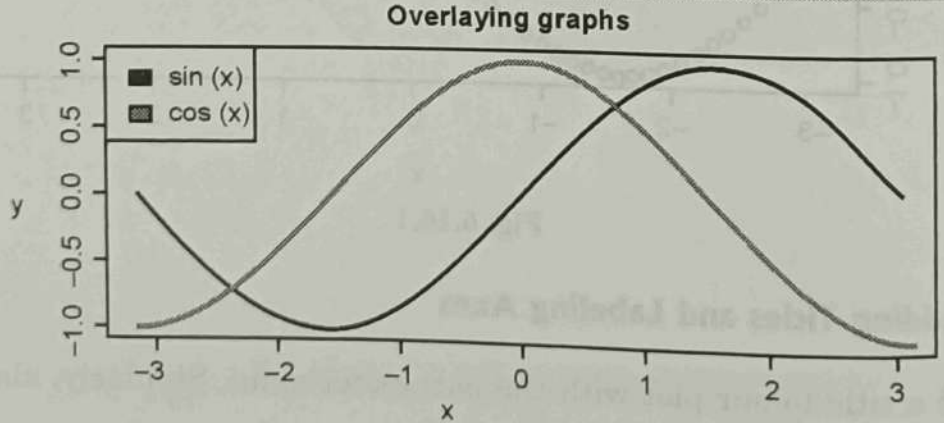


Fig. 6.16.3

- We have used the function legend() to appropriately display the legend.

6.17 ACCESSING HELP AND DOCUMENTATION IN R

- The help() function and ? help operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.
- The help() function and ? help operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.
- To get general help just type the below command

```
help.start()
```

- To access documentation for the standard lm (linear model) function, for example, enter the command.

```
help(lm)
```

OR

```
help("lm")
```

OR

```
?lm
```

OR

```
?"lm"
```

- You can use any of the four commands all serves same purpose only. (quotes are optional.)
- To access help for a function in a package that's *not* currently loaded, specify in addition the name of the package: For example, to obtain documentation for the rlm() (robust linear model) function in the MASS package, help(rlm, package="MASS").

```
help(rlm, package="MASS")
```

6.17.1 Reading Dataset and Exploring Data set in R

6.17.1.1 Reading Datasets

read.csv(): read.csv() is used for reading "comma separated value" files (".csv"). In this also the data will be imported as a data frame.

Example

```
myData = read.csv("basic.csv")
print(myData)
```

Output

	Name	Age	Qualification	Address
1	Amiya	18	MCA	BBS
2	Niru	23	Msc	BLS
3	Debi	23	BCA	SBP
4	Biku	56	ISC	JJP

read.csv2(): read.csv() is used for variant used in countries that use a comma "," as decimal point and a semicolon ";" as field separators.

Example

```
myData = read.csv2("basic.csv")
print(myData)
```

Output

	Name.	Age.	Qualification.	Address
1	Amiya,	18,	MCA,	BBS
2	Niru,	23,	MSc,	BLS
3	Debi,	23,	BCA,	SBP
4	Biku,	56,	ISC,	JJP

Reading a file from the internet

It's possible to use the functions **read.delim()**, **read.csv()** and **read.table()** to import files from the web.

Example

```
myData = read.delim("http://www.sthda.com/upload/boxplot_format.txt")
print(head(myData))
```

Output

	Nom	variable	Group
1	IND1	10	A
2	IND2	7	A
3	IND3	20	A
4	IND4	14	A
5	IND5	14	A
6	IND6	12	A

6.17.2 Exploring Dataset

For Data Analysis sometimes creating CSV data file is required and do some operations on it as per our requirement. To write to csv file write.csv() function is used.

Syntax

write.csv(data, path)

Parameter

- data: data to be added to csv
- path: pathname of the file

Approach

- Create a DataFrame
- Pass required values to the function
- Write to file
- Let us first create a data frame.

For example

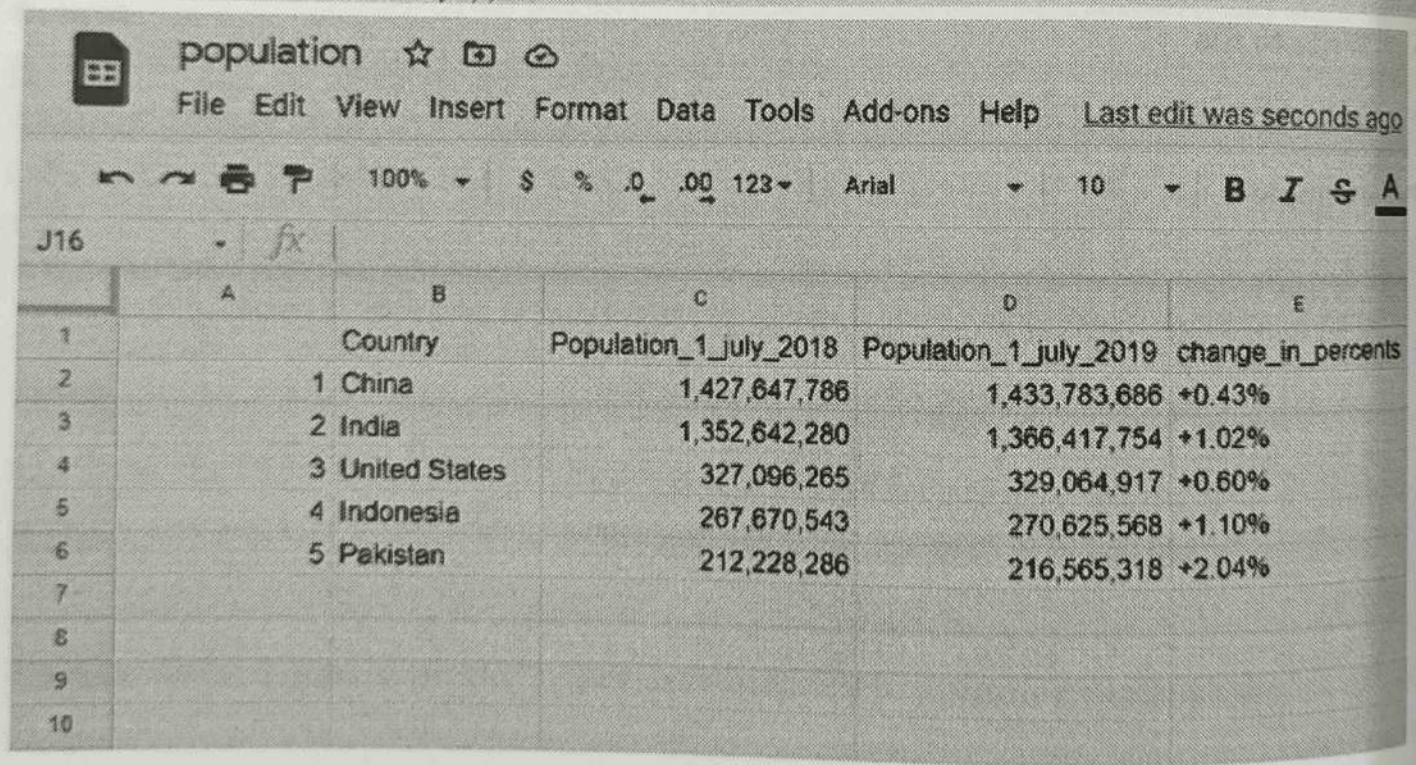
```
Country <- c("China", "India", "United States", "Indonesia", "Pakistan")

Population_1_july_2018 <- c("1,427,647,786", "1,352,642,280",
                "327,096,265", "267,670,543", "212,228,286")

Population_1_july_2019 <- c("1,433,783,686", "1,366,417,754",
                "329,064,917", "270,625,568", "216,565,318")

change_in_percents<- c("+0.43%", "+1.02%", "+0.60%", "+1.10%", "+2.04%")
data <- data.frame(Country, Population_1_july_2018, Population_1_july_2019, change_in_percents)
print(data)
```

Output

```
> print(data)
        Country Population_1_july_2018 Population_1_july_2019 change_in_percents
1         China           1,427,647,786           1,433,783,686             +0.43%
2         India           1,352,642,280           1,366,417,754             +1.02%
3 United States             327,096,265             329,064,917             +0.60%
4     Indonesia             267,670,543             270,625,568             +1.10%
5      Pakistan             212,228,286             216,565,318             +2.04%
>
```

Example of writing data to a csv file and save it to a required location.

```
Country <- c("China", "India", "United States", "Indonesia", "Pakistan")

Population_1_july_2018 <- c("1,427,647,786", "1,352,642,280",
                            "327,096,265", "267,670,543", "212,228,286")

Population_1_july_2019 <- c("1,433,783,686", "1,366,417,754",
                            "329,064,917", "270,625,568", "216,565,318")

change_in_percents<- c("+0.43%", "+1.02%", "+0.60%", "+1.10%", "+2.04%")


data <- data.frame(Country, Population_1_july_2018, Population_1_july_2019, change_in_percents)
print(data)

write.csv(data,"C:\\Users\\...YOUR PATH...\\population.csv")
nt ('CSV file written Successfully :)')
```



	A	B	C	D	E
1		Country	Population_1_july_2018	Population_1_july_2019	change_in_percents
2	1	China	1,427,647,786	1,433,783,686	+0.43%
3	2	India	1,352,642,280	1,366,417,754	+1.02%
4	3	United States	327,096,265	329,064,917	+0.60%
5	4	Indonesia	267,670,543	270,625,568	+1.10%
6	5	Pakistan	212,228,286	216,565,318	+2.04%
7					
8					
9					
10					

Fig. 6.17.1

6.17.3 Data Manipulation and Processing

GQ. Explain Data Manipulation techniques in R programming ?

- Data manipulation involves modifying data to make it easier to read and to be more organized.

- We manipulate data for analysis and visualization. It is also used with the term 'data exploration' which involves organizing data using available sets of variables.
- At times, the data collection process done by machines involves a lot of errors and inaccuracies in reading.
- Data manipulation is also used to remove these inaccuracies and make data more accurate and precise.

For example

use the default **iris** table in R, as follows:

```
#To load datasets package
library("datasets")
#To load iris dataset
data(iris)
summary(iris)
```

Output

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa: 50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	versicolor:0.300	versicolor:50
Median: 5.800	Median: 3.000	Median: 4.350	Median: 1.300	Virginica: 50
Mean: 5.843	Mean: 3.057	Mean: 3.758	Mean: 1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	

Sample()

- It is used to generate a sample of a specific size from a vector or a dataset, either with or without replacement.
- The basic syntax of sample() function is as follows :

```
sample(data, size, replace = FALSE, prob = NULL)
```

- For example :

```
#To return 5 random rows
index<-sample(1:nrow(iris), 5)
index
iris[index,]
```

Output

Sl. No.	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
137	6.3	3.4	5.6	2.4	Virginica
85	5.4	3.0	4.5	1.5	Versicolor
14	4.3	3.0	1.1	0.1	Setosa
54	5.5	2.3	4.0	1.3	Versicolor
4	4.6	3.1	1.5	0.2	Setosa

Table()

- It is used to create a frequency table to calculate the occurrences of unique values of a variable.
- The table() function generates an object of the table class.
- For example:

```
#To find the frequency distribution of Species in iris table
data(iris)
freq.table<- table(iris$Species)
head(freq.table)
```

Output

Setosa	Versicolor	Virginica
50	50	50

6.17.3.1 Data Manipulation in R With dplyr Package

- There are different ways to perform data manipulation in R, such as using Base R functions like subset(), with(), within(), etc., Packages like data.table, ggplot2, reshape2, readr, etc., and different Machine Learning algorithms.
- However, in this tutorial, we are going to use the dplyr package to perform data manipulation in R.
- The dplyr package consists of many functions specifically used for data manipulation. These functions process data faster than Base R functions and are known the best for data exploration and transformation, as well.

- Following are some of the important functions included in the dplyr package

```
select() :- To select columns (variables)
filter() :-To filter (subset) rows.
mutate() :-To create new variables
summarise() :- To summarize (or aggregate) data
group_by() :- To group data
arrange() :- To sort data
join() :- To join data frames.
```

- To install the dplyr package, run the following command :

```
install.packages("dplyr")
```

- We are going to use the iris dataset from the datasets package in R programming that can be loaded as follows :

```
#To load dplyr package
library("dplyr")
#To load datasets package
library("datasets")
#To load iris dataset
data(iris)
summary(iris)
```

Output

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa: 50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	versicolor:0.300	versicolor:50
Median: 5.800	Median: 3.000	Median: 4.350	Median: 1.300	virginica: 50
Mean: 5.843	Mean: 3.057	Mean: 3.758	Mean: 1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	

- It contains 150 samples of three plant species (setosa, virginica, and versicolor) and four features measured for each sample.

Select()

- It is used to select data by its column name. We can select any number of columns in a number of ways.

- For example :

```
#To select the following columns
selected <- select(iris, Sepal.Length, Sepal.Width, Petal.Length)
head(selected)
#To select all columns from Sepal.Length to Petal.Length
selected1 <- select(iris, Sepal.Length:Petal.Length)
#To print first four rows
head(selected1, 4)
#To select columns with numeric indexes
selected1 <- select(iris,c(3:5))
head(selected1)
```

Output

Sl.No.	Sepal.Length	Sepal.Width	Petal.Length
1	5.1	3.5	1.4
2	4.9	3.0	1.4
3	4.7	3.2	1.3
4	4.6	3.1	1.5
5	5.0	3.6	1.4
6	5.4	3.9	1.7

Output

Sl.No.	Sepal.Length	Sepal.Width	Petal.Length
1	5.1	3.5	1.4
2	4.9	3.0	1.4
3	4.7	3.2	1.3
4	4.6	3.1	1.5

Output

Sl.No.	Petal.Length	Petal.Width	Species
1	1.4	0.2	Setosa
2	1.4	0.2	Setosa
3	1.3	0.2	Setosa
4	1.5	0.2	Setosa
5	1.4	0.2	Setosa
6	1.7	0.4	Setosa

```
#We use(-)to hide a particular column
selected <- select(iris, -Sepal.Length, -Sepal.Width)
head(selected)
```

Output

Sl.No.	Petal.Length	Petal.Width	Species
1	1.4	0.2	Setosa
2	1.4	0.2	Setosa
3	1.3	0.2	Setosa
4	1.5	0.2	Setosa
5	1.4	0.2	Setosa
6	1.7	0.4	Setosa

Filter()

- It is used to find rows with matching criteria. It also works like the select() function, i.e., we pass a data frame along with a condition separated by a comma.
- For example :

```
#To select the first 3 rows with Species as setosa
filtered <- filter(iris, Species == "setosa" )
head(filtered,3)
```

Output

Sl. No.	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
3	4.7	3.2	1.3	0.2	Setosa

```
#To select the last 5 rows with Species as versicolor and Sepal width more than 3
filtered1 <- filter(iris, Species == "versicolor", Sepal.Width > 3)
tail(filtered1)
```

Output

Sl. No.	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
4	6.3	3.3	4.7	1.6	Versicolor
5	6.7	3.1	4.4	1.4	Versicolor
6	5.9	3.2	4.8	1.8	Versicolor
7	6.0	3.4	4.5	1.6	Versicolor
8	6.7	3.1	4.7	1.5	Versicolor

Mutate()

- It creates new columns and preserves the existing columns in a dataset.
- For example:

```
#To create a column "Greater.Half" which stores TRUE if given condition
is TRUE
col1 <- mutate(iris, Greater.Half = Sepal.Width > 0.5 * Sepal.Length)
tail(col1)
```

Output

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Greater.Half
145	6.7	3.3	5.7	2.5	Virginica	FALSE
146	6.7	3.0	5.2	2.3	Virginica	FALSE
147	6.3	2.5	5.0	1.9	Virginica	FALSE

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Greater.Half
148	6.5	3.0	5.2	2.0	Virginica	FALSE
149	6.2	3.4	5.4	2.3	Virginica	TRUE
150	5.9	3.0	5.1	1.8	Virginica	TRUE

```
#To check how many flowers satisfy this condition
table(col1$Greater.Half)
```

Output

```
FALSE=84 TRUE=66
```

Arrange()

It is used to sort rows by variables in both an ascending and descending order. For example:

```
#To arrange Sepal Width in ascending order
arranged <- arrange(col1, Sepal.Width)
head(arranged)
#To arrange Sepal Width in descending order
arranged <- arrange(col1, desc(Sepal.Width))
head(arranged)
```

Output

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Greater.Half
1	5.0	2.0	3.5	1.0	Versicolor	FALSE
2	6.0	2.2	4.0	1.0	Versicolor	FALSE
3	6.2	2.2	4.5	1.5	Versicolor	FALSE
4	6.0	2.2	5.0	1.5	Virginica	FALSE
5	4.5	2.3	1.3	0.3	Setosa	TRUE
6	5.5	2.3	4.0	1.3	Versicolor	FALSE

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Greater.Half
1	5.7	4.4	1.5	0.4	Setosa	TRUE
2	5.5	4.2	1.4	0.2	Setosa	TRUE
3	5.2	4.1	1.5	0.1	Setosa	TRUE
4	5.8	4.0	1.2	0.2	Setosa	TRUE
5	5.4	3.9	1.7	0.4	Setosa	TRUE
6	5.4	3.9	1.3	0.4	Setosa	TRUE

Summarise()

- It is used to find insights(mean, median, mode, etc.) from a dataset. It reduces multiple values down to a single value.
- For example :

```
summarised<- summarise(arranged, Mean.Width = mean(Sepal.Width))
head(summarised)
```

Output

```
Mean.Width
1  3.057333
```

Grouping 10

- It is done to group observations within a dataset by one or more variables. Most data operations are performed on groups defined by variables.
- For example:

```
#To find mean sepal width by Species, we use grouping as follows
gp<- group_by(iris,Species)
mn<- summarise(gp,Mean.Sepal = mean(Sepal.Width))
head(mn)
```

Output

Sl. No.	Species <*fct*>	Mean.Sepal <*dbl*>
1	Setosa	3.43
2	Versicolor	2.77
3	Virginica	2.97

6.17.4 Using Functions

GQ. List and explain built in functions in R programming?
GQ. How to declare function in R programming?

- A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.
- In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.
- The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Function Definition

An R function is created by using the keyword **function**. The basic syntax of an R function definition is as follows –

```
function_name<- function(arg_1, arg_2, ...) {
 Function body
}
```

Function Components

The different parts of a function are as follows :

1. **Function Name :** This is the actual name of the function. It is stored in R environment as an object with this name.
2. **Arguments :** An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
3. **Function Body :** The function body contains a collection of statements that defines what the function does.
4. **Return Value :** The return value of a function is the last expression in the function body to be evaluated.

R has many **in-built** functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as **user defined** functions.

6.17.5 Built-in Function

Simple examples of in-built functions are **seq()**, **mean()**, **max()**, **sum(x)** and **paste(...)** etc. They are directly called by user written programs. You can refer most widely used R functions.

```
# Create a sequence of numbers from 32 to 44.
print(seq(32,44))

# Find mean of numbers from 25 to 82.
print(mean(25:82))
# Find sum of numbers frm 41 to 68.
print(sum(41:68))
```

When we execute the above code, it produces the following result −

```
[1] 32 33 34 35 36 37 38 39 40 41 42 43 44
[1] 53.5
[1] 1526
```

User-defined Function

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
# Create a function to print squares of numbers in sequence.
new.function<-function(a){
for(iin1:a){
    b <- i^2
print(b)
}
}
```

Calling a Function

```
# Create a function to print squares of numbers in sequence.
new.function<-function(a){
for(iin1:a){
    b <- i^2
print(b)
}
}

# Call the function new.function supplying 6 as an argument.
new.function(6)
```

When we execute the above code, it produces the following result –

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
```

Calling a Function without an Argument

```
# Create a function without an argument.
new.function<-function(){
for(in1:5){
print(i^2)
}
}

# Call the function without supplying an argument.
new.function()
```

When we execute the above code, it produces the following result –

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

Calling a Function with Argument Values (by position and by name)

The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

```
# Create a function with arguments.
new.function<-function(a,b,c){
result<- a * b + c
print(result)
}

# Call the function by position of arguments.
new.function(5,3,11)

# Call the function by names of the arguments.
new.function(a =11, b =5, c =3)
```

When we execute the above code, it produces the following result –

```
[1] 26
[1] 58
```

☞ Calling a Function with Default Argument

We can define the value of the arguments in the function definition and call the function without supplying any argument to get the default result. But we can also call such functions by supplying new values of the argument and get non default result.

```
# Create a function with arguments.
new.function<-function(a =3, b =6){
result<- a * b
print(result)
}

# Call the function without giving any argument.
new.function()

# Call the function with giving new values of the argument.
new.function(9,5)
```

When we execute the above code, it produces the following result –

```
[1] 18
[1] 45
```

☞ Lazy Evaluation of Function

Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body.

```
# Create a function with arguments.
new.function<-function(a, b){
print(a^2)
print(a)
print(b)
}

# Evaluate the function without supplying one of the arguments.
new.function(6)
```

When we execute the above code, it produces the following result –

```
[1] 36
[1] 6
Error in print(b) : argument "b" is missing, with no default
```

6.18 DATA VISUALIZATION IN R

GQ. List and explain different types of Data visualization techniques ?
GQ. What are the different application areas of data visualization ?

Data visualization is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

☞ Data Visualization in R Programming Language

- The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana.
- The various data visualization platforms have different capabilities, functionality, and use cases. They also require a different skill set.R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.
- Consider the following *airquality* data set for visualization in R:

Ozone	Solar R.	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

6.18.1 Types of Data Visualizations

Some of the types of visualizations offered by R are as follow :

1. Bar Plot

- There are two types of bar plots- horizontal and vertical which represent data points as horizontal or vertical bars of certain lengths proportional to the value of the data item.
- They are generally used for continuous and categorical variable plotting. By setting the **horiz** parameter to true and false, we can get horizontal and vertical bar plots respectively.

```
barplot(airquality$Ozone,
    main = 'Ozone Concenteration in air',
    xlab = 'ozone levels', horiz = TRUE)
```

Result

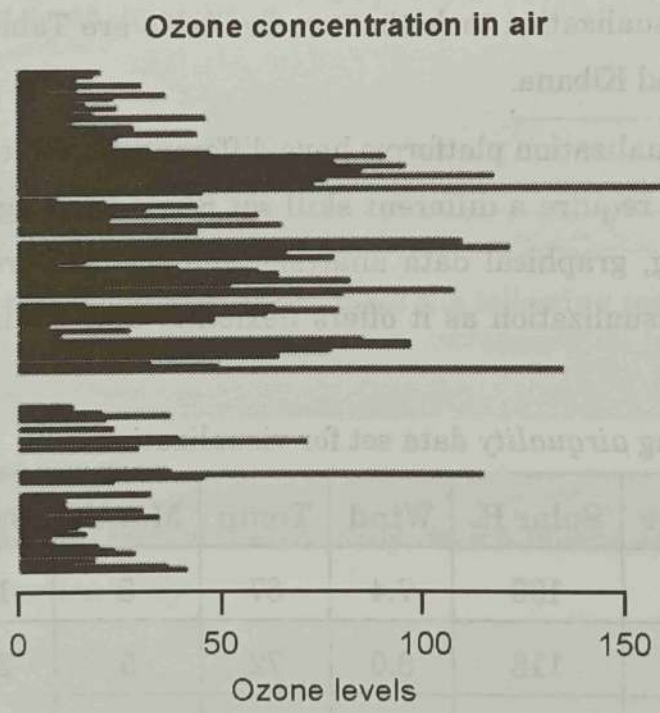


Fig. 6.18.1

Bar plots are used for the following scenarios :

- To perform a comparative study between the various data categories in the data set.
- To analyze the change of a variable over time in months or years.

8. Histogram

- A histogram is like a bar chart as it uses bars of varying height to represent data distribution. However, in a histogram values are grouped into consecutive intervals called bins.
- In a Histogram, continuous values are grouped and displayed in these bins whose size can be varied.

Example :

```
data(airquality)
hist(airquality$Temp, main ="La Guardia Airport's\
Maximum Temperature(Daily)",
 xlab ="Temperature(Fahrenheit)",
 xlim = c(50, 125), col ="yellow",
 freq = TRUE)
```

Output

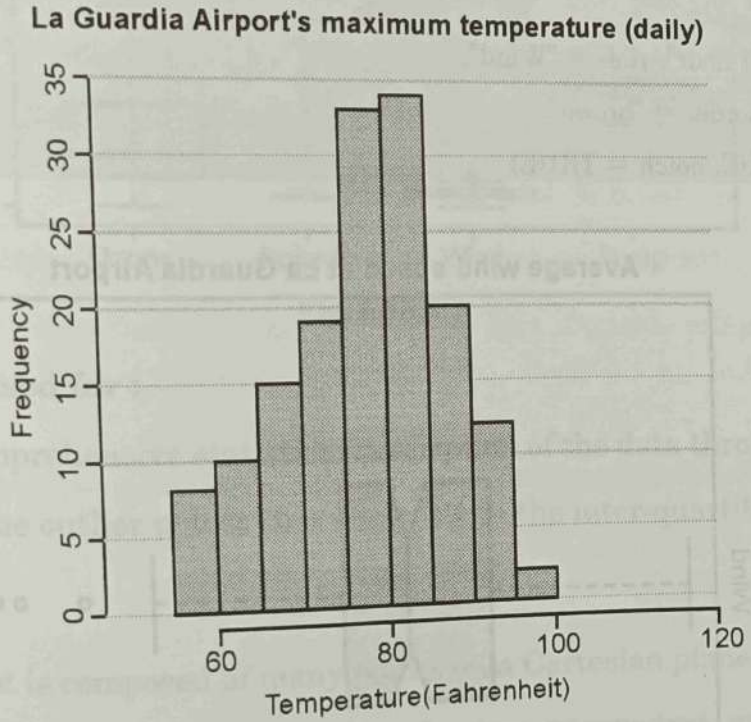


Fig. 6.18.2

- For a histogram, the parameter **xlim** can be used to specify the interval within which all values are to be displayed.
- Another parameter **freq** when set to TRUE denotes the frequency of the various values in the histogram and when set to FALSE, the probability densities are represented on the y-axis such that they are of the histogram adds up to one.

Histograms are used in the following scenarios

- To verify an equal and symmetric distribution of the data.
- To identify deviations from expected values.

3. Box Plot

- The statistical summary of the given data is presented graphically using a boxplot.
- A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.

Example :

```
data(airquality)
boxplot(airquality$Wind, main = "Average wind speed\
at La Guardia Airport",
    xlab = "Miles per hour", ylab = "Wind",
    col = "orange", border = "brown",
    horizontal = TRUE, notch = TRUE)
```

Output

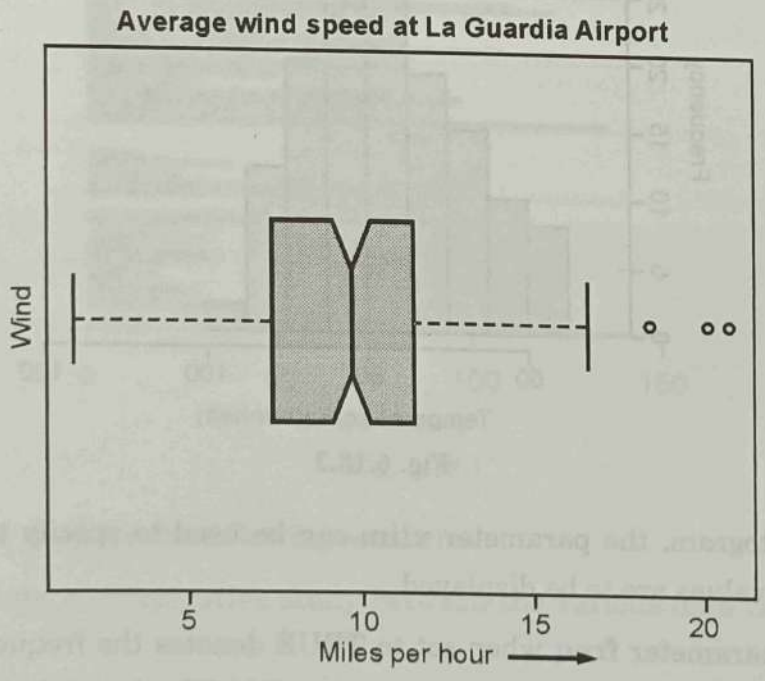


Fig. 6.18.3

Multiple box plots can also be generated at once through the following code

Example :

```
# Multiple Box plots, each representing
# an Air Quality Parameter
boxplot(airquality[, 0:4],
   main ='Box Plots for Air Quality Parameters')
```

Output

Output:

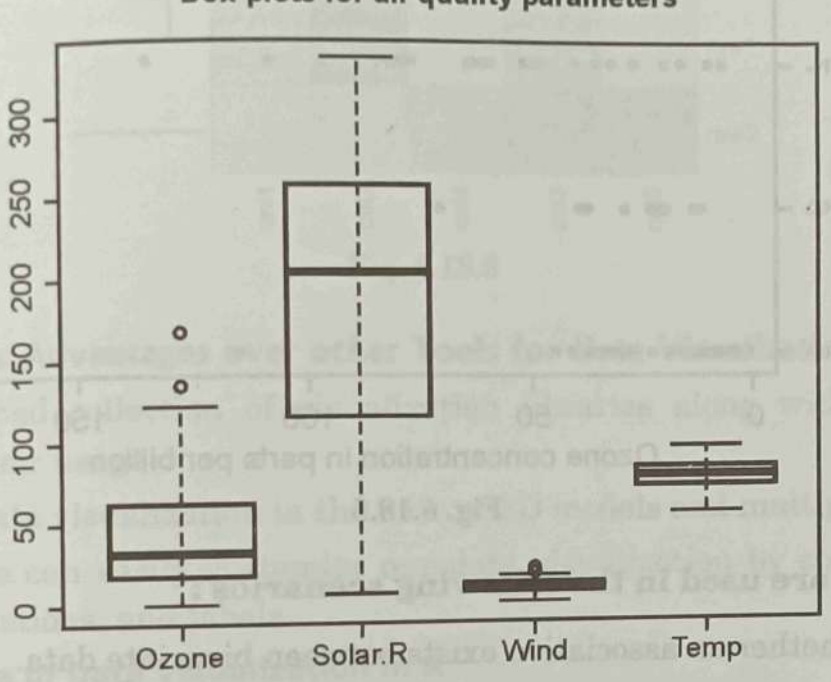


Fig. 6.18.4

Box Plots are used for :

- To give a comprehensive statistical description of the data through a visual cue.
- To identify the outlier points that do not lie in the inter-quartile range of data.

4. Scatter Plot

- A scatter plot is composed of many points on a Cartesian plane.
- Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.

```
data(airquality)
plot(airquality$Ozone, airquality$Month,
   main ="Scatterplot Example",
   xlab ="Ozone Concentration in parts per billion",
   ylab =" Month of observation ", pch = 19)
```

Output

Output:

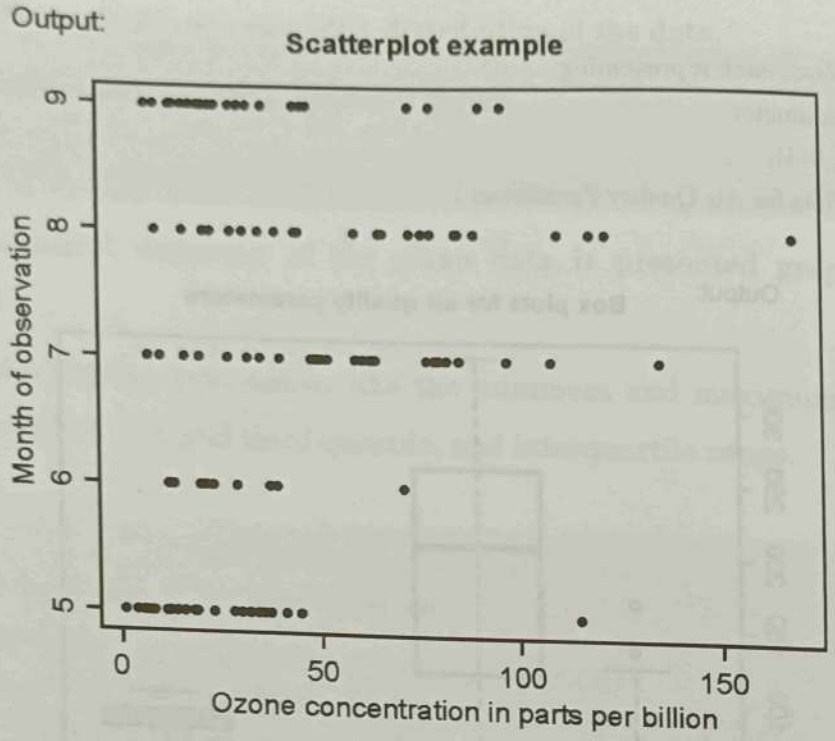


Fig. 6.18.5

Scatter Plots are used in the following scenarios :

- To show whether an association exists between bivariate data.
- To measure the strength and direction of such a relationship.

5. Heat Map

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. heatmap() function is used to plot heatmap.

Syntax : heatmap(data)

```
data <- matrix(rnorm(50, 0, 5), nrow = 5, ncol = 5)
 # Column names
colnames(data) <- paste0("col", 1:5)
rownames(data) <- paste0("row", 1:5)
 # Draw a heatmap
heatmap(data)
```

Output

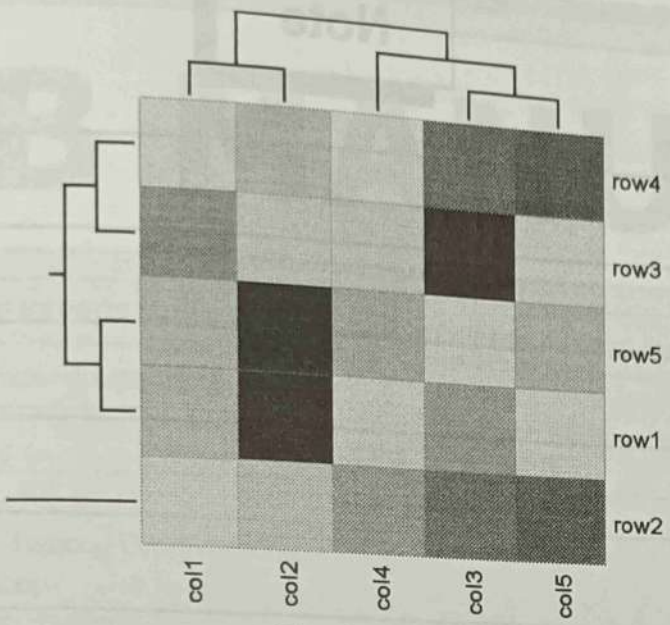


Fig. 6.18.6

6.18.2 R has Advantages over other Tools for Data Visualization

(1) R offers a broad collection of visualization libraries along with extensive online guidance on their usage.

(2) R also offers data visualization in the form of 3D models and multipanel charts.

(3) Through R, we can easily customize our data visualization by changing axes, fonts, legends, annotations, and labels.

Disadvantages of Data Visualization in R

(1) R also has the following disadvantages :

(2) R is only preferred for data visualization when done on an individual standalone server.

(3) Data visualization using R is slow for large amounts of data as compared to other counterparts.

Application Areas

(1) Presenting analytical conclusions of the data to the non-analysts departments of your company.

(2) Health monitoring devices use data visualization to track any anomaly in blood pressure, cholesterol and others.

(3) To discover repeating patterns and trends in consumer and marketing data.

(4) Meteorologists use data visualization for assessing prevalent weather changes throughout the world.

(5) Real-time maps and geo-positioning systems use visualization for traffic monitoring and estimating travel time.

Chapter Ends...

❑❑❑

LAB MANUAL

SUGGESTED LIST OF EXPERIMENTS (Select a case study and perform the experiments 1 to 8.).

Star (*) marked experiments are compulsory.

Sr. No.	Name of the Experiment
1*	Hadoop HDFS Practical: - HDFS Basics, Hadoop Ecosystem Tools Overview. - Installing Hadoop. - Copying File to Hadoop. - Copy from Hadoop File system and deleting file. - Moving and displaying files in HDFS. - Programming exercises on Hadoop **(Refer Experiment 1)**
2	Use of Sqoop tool to transfer data between Hadoop and relational database servers. a. Sqoop - Installation. b. To execute basic commands of Hadoop eco system componentSqoop. **(Refer Experiment 2)**
3*	To install and configure MongoDB/ Cassandra/ HBase/ Hypertable to execute NoSQL commands **(Refer Experiment 3)**
4	Experiment on Hadoop Map-Reduce: - Write a program to implement a word count program using MapReduce. **(Refer Experiment 4)**
5	Experiment on Hadoop Map-Reduce: - Implementing simple algorithms in Map-Reduce: Matrix multiplication, Aggregates, Joins, Sorting, Searching, etc **(Refer Experiment 5)**
6	Create HIVE Database and Descriptive analytics-basic statistics. **(Refer Experiment 6)**
7*	Data Stream Algorithms (any one): – Implementing DGIM algorithm using any Programming Language – Implement Bloom Filter using any programming language Implement Flajolet Martin algorithm using any programming language **(Refer Experiment 7)**
8	Social Network Analysis using R (for example: Community Detection Algorithm)**(Refer Experiment 8)**
9	Data Visualization using Hive/PIG/R/Tableau/. **(Refer Experiment 9)**
10	Exploratory Data Analysis using Spark/ Pyspark. **(Refer Experiment 10)**
11*	Mini Project: One real life large data application to be implemented (Use standard Datasets available on the web). – Streaming data analysis – use flume for data capture, HIVE/PYSpark for analysis of twitter data, chat data, weblog analysis etc. – Recommendation System (for example: Health Care System, Stock Market Prediction, Movie Recommendation, etc.) SpatioTemporal DataAnalytics **(Refer Experiment 11)**

LAB ASSIGNMENT

Experiment 1

HDFS stands for Hadoop Distributed File System. HDFS is one of the core components of the Hadoop framework and is responsible for the storage aspect. Unlike the usual storage available on our computers, HDFS is a Distributed File System and parts of a single large file can be stored on different nodes across the cluster. HDFS is a distributed, reliable, and scalable file system.

Features of HDFS

- HDFS is implemented in Java and any computer which can run Java can host a NameNode/DataNode on it.
- Designed to be portable across all the major hardware and software platforms.
- Basic file operations include reading and writing of files, creation, deletion, and replication of files, etc.
- Provides necessary interfaces which enable the movement of computation to the data unlike in traditional data processing systems where the data moves to computation.
- HDFS provides a command line interface called "FS Shell" used to interact with HDFS.

HDFS Architecture

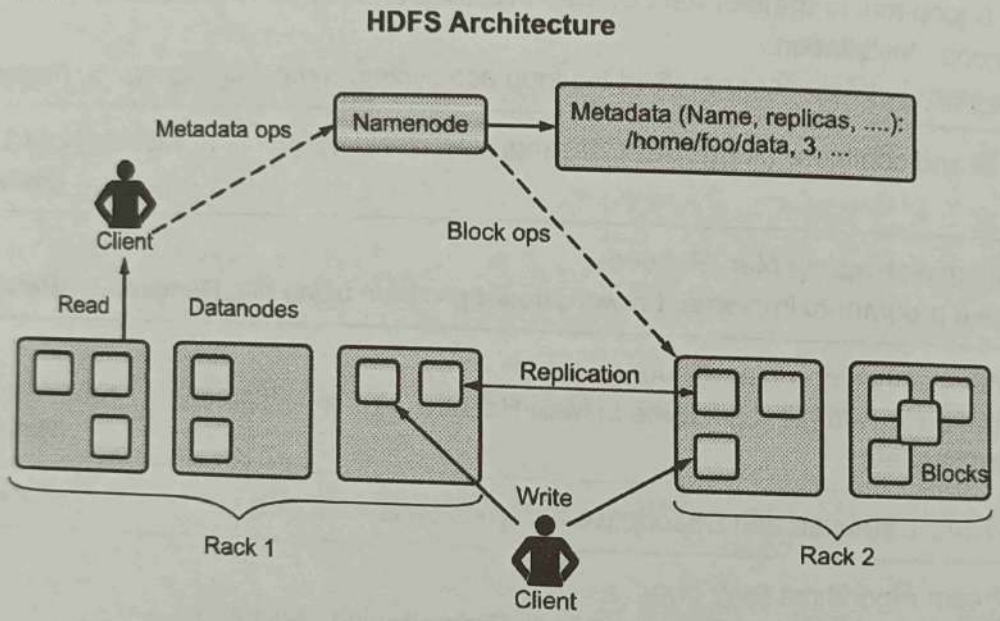


Fig. L1

HDFS works in a master-slave/master-worker fashion.

- NameNode servers as the master and each DataNode servers as a worker/slave.
- NameNode and each DataNode have built-in web servers.
- NameNode is the heart of HDFS and is responsible for various tasks including - it holds the file system namespace, controls access to file system by the clients, keeps track of the DataNodes, keeps track of replication factor and ensures that it is always maintained.
- User data is stored on the local file system of DataNodes. DataNode is not aware of the files to which the blocks stored on it belong to. As a result of this, if the NameNode goes down then the data in HDFS is non-usable as only the NameNode knows which blocks belong to which file, where each block located etc.
- NameNode can talk to all the live DataNodes and the live DataNodes can talk to each other.

There is also a Secondary NameNode which comes in handy when the Primary NameNode goes down. Secondary NameNode can be brought up to bring the cluster online. This process of switching of nodes needs to be done manually and there is no automatic failover mechanism in place.

- NameNode receives heartbeat signals and a Block Report periodically from each of the DataNodes.
- Heartbeat signals from a DataNode indicates that the corresponding DataNode is alive and is working fine. If a heartbeat signal is not received from a DataNode then that DataNode is marked as dead and no further I/O requests are sent to that DataNode.
- Block Report from each DataNode contains a list of all the blocks that are stored on that DataNode.
- Heartbeat signals and Block Reports received from each DataNode help the NameNode to identify any potential loss of Blocks/Data and to replicate the Blocks to other active DataNodes in the event when one or more DataNodes holding the data blocks go down.
- Data is replicated across different DataNodes to ensure a high degree of fault-tolerance.

Advantages of HDFS

1. HDFS can be used for storing the Big Data sets for further processing.
2. HDFS can be used for storing archive data since it is cheaper as HDFS allows storing the data on low cost commodity hardware while ensuring a high degree of fault-tolerance.

Drawbacks

- HDFS is not suitable for storing data related to applications requiring low latency data access.
- HDFS is not suitable for storing a large number of small files as the metadata for each file needs to be stored on the NameNode and is held in memory.
- HDFS is not suitable for scenarios requiring multiple/simultaneous writes to the same file.

Hadoop Ecosystem Tools Overview

There are *four major elements of Hadoop* i.e. **HDFS, MapReduce, YARN, and Hadoop Common.**

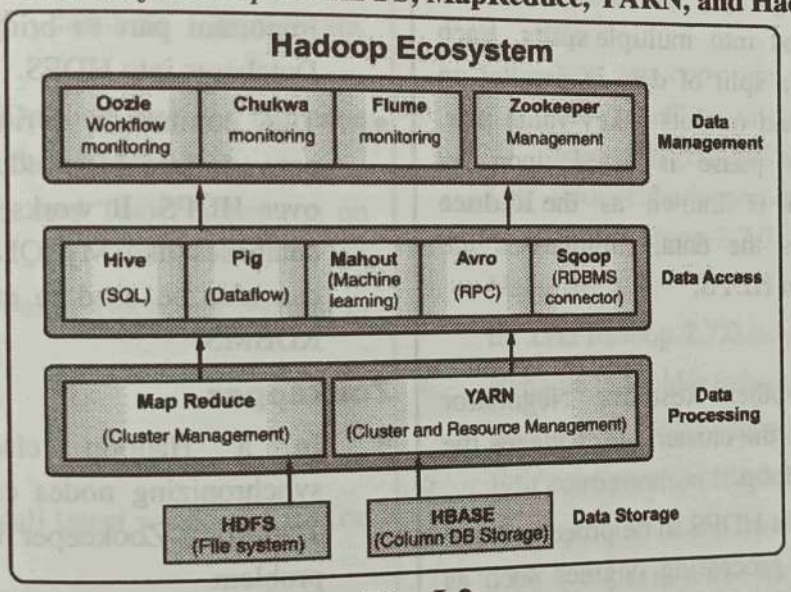


Fig. L2

- **HDFS:** Hadoop Distributed File System
- **YARN:** Yet Another Resource Negotiator
- **MapReduce:** Programming based Data Processing
- **Spark:** In-Memory data processing
- **PIG, HIVE:** Query based processing of data services
- **HBase:** NoSQL Database
- **Mahout, Spark MLLib:** Machine Learning algorithm libraries
- **Solar, Lucene:** Searching and Indexing
- **Zookeeper:** Managing cluster
- **Oozie:** Job Scheduling

HDFS

It is the storage component of Hadoop that stores data in the form of files. Each file is divided into blocks of 128MB (configurable) and stores them on different machines in the cluster.

It has a master-slave architecture with two main components: Name Node and Data Node.

- **Name node** is the master node and there is only one per cluster. Its task is to know where each block belonging to a file is lying in the cluster
- **Data node** is the slave node that stores the blocks of data and there are more than one per cluster. Its task is to retrieve the data as and when required. It keeps in constant touch with the Name node through heartbeats

MapReduce

- To handle Big Data, Hadoop relies on the MapReduce algorithm introduced by Google and makes it easy to distribute a job and run it in parallel in a cluster. It essentially divides a single task into multiple tasks and processes them on different machines.
- In layman terms, it works in a divide-and-conquer manner and runs the processes on the machines to reduce traffic on the network.

  It has two important phases: Map and Reduce.
- **Map phase** filters, groups, and sorts the data. Input data is divided into multiple **splits**. Each map task works on a split of data in parallel on different machines and outputs a key-value pair. The output of this phase is acted upon by the **reduce task** and is known as the **Reduce phase**. It aggregates the data, summarises the result, and stores it on HDFS.

YARN

- YARN or Yet Another Resource Negotiator manages resources in the cluster and manages the applications over Hadoop.
- It allows data stored in HDFS to be processed and run by various data processing engines such as batch processing, stream processing, interactive processing, graph processing, and many more. This increases efficiency with the use of YARN.

HBase

HBase is a Column-based NoSQL database. It runs on top of HDFS and can handle any type of data. It allows for real-time processing and random read/write operations to be performed in the data.

Pig

- Pig was developed for analyzing large datasets and overcomes the difficulty to write map and reduce functions. It consists of two components: Pig Latin and Pig Engine.
- Pig Latin is the Scripting Language that is similar to SQL. Pig Engine is the execution engine on which Pig Latin runs. Internally, the code written in Pig is converted to MapReduce functions and makes it very easy for programmers who aren't proficient in Java

Hive

- Hive is a distributed data warehouse system developed by Facebook. It allows for easy reading, writing, and managing files on HDFS. It has its own querying language for the purpose known as Hive Querying Language (HQL) which is very similar to SQL.
- This makes it very easy for programmers to write MapReduce functions using simple HQL queries.

Sqoop

- A lot of applications still store data in relational databases, thus making them a very important source of data. Therefore, Sqoop plays an important part in bringing data from Relational Databases into HDFS.
- The commands written in Sqoop internally converts into MapReduce tasks that are executed over HDFS. It works with almost all relational databases like MySQL, Postgres, SQLite, etc. It can also be used to export data from HDFS to RDBMS.

Zookepper

- In a Hadoop cluster, coordinating and synchronizing nodes can be a challenging task. Therefore, Zookeeper is the perfect tool for the problem.
- It is an open-source, distributed, and centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services across the cluster.

Spark

- It is an alternative framework to Hadoop built on Scala but supports varied applications written in Java, Python, etc. Compared to MapReduce it provides in-memory processing which accounts for faster processing.
- In addition to batch processing offered by Hadoop, it can also handle real-time processing.

Installing Hadoop

Hadoop software can be installed in three modes of operation:

- **Stand Alone Mode**: Hadoop is a distributed software and is designed to run on a commodity of machines. However, we can install it on a single node in stand-alone mode. In this mode, Hadoop software runs as a single monolithic java process. This mode is extremely useful for debugging purpose. You can first test run your Map-Reduce application in this mode on small data, before actually executing it on cluster with big data.
- **Pseudo Distributed Mode**: In this mode also, Hadoop software is installed on a Single Node.
- Various daemons of Hadoop will run on thesame machine as separate java processes.
- Hence all the daemons namely NameNode ,DataNode, Secondary NameNode, JobTracker, TaskTracker run on single machine.
- Fully **Distributed Mode**: In Fully DistributedMode, the daemons NameNode, JobTracker,
- SecondaryNameNode (Optional and can be runon a separate node) run on the Master Node.
- The daemons DataNode and TaskTracker run on the Slave Node.
- Hadoop Installation: Ubuntu Operating System in stand-alonemode

Steps for Installation

1. sudo apt-get update
2. In this step, we will install latest version of JDKon the machine.

   The Oracle JDK is the official JDK; however, it is no longer provided by Oracle as a default installation for Ubuntu. You can still install it using apt-get. To install any version, first execute the following commands:

   a. sudo apt-get install python-software properties

   b. sudo add-apt-repository ppa:webupd8team/java

   c. sudo apt-get update

   Then, depending on the version you want to install, execute one of the following commands:

   - **Oracle JDK 7:** sudo apt-get install oraclejava7-installer

   Oracle JDK 8: sudo apt-get install oracle java 8 - installer
3. Now, let us setup a new user account for Hadoop installation. This step is optional, but recommended because it gives you flexibility to have a separate account for Hadoop installation by separating this installation from other software installation

   a. sudoadduserhadoop_dev(Upon executing this command, you will prompted to enter the new password for this user. Please enter the password and enter other details. Don't forget to save the details at the end)

   b. su - hadoop_dev(Switches the user from current user to the new user created i.e Hadoop_dev)
4. Download the latest Hadoop distribution.

   a. Visit this URL and choose one of the mirror sites.

   You can copy the download link and also use "wget" to download it from command prompt:Wgethttp://apache.mirrors.lucidnetworks.net/hadoop/common/hadoop-2.7.0/hadoop-2.7.0.tar.gz
5. Untar the file :

   tar xvzf hadoop-2.7.0.tar.gz
6. Rename the folder to hadoop2

   mv hadoop-2.7.0 hadoop2
7. Edit configuration file /home/hadoop_dev/ hadoop2/etc/hadoop/hadoop-env.sh and set JAVA_HOME in that file.

   a. vim /home/hadoop_dev/hadoop2/etc/hadoop/ hadoop-env.sh

   b. uncomment JAVA_HOME and update it followingline:

   export JAVA_HOME=/usr/lib/jvm/java-8-oracle

   (Please check for your relevant javainstallation and set this value accordingly. Latest versions of Hadoop require > JDK1.7)

8. Let us verify if the installation is successful ornot (change to home directory cd /home/ hadoop_dev/hadoop2/):
   a. bin/hadoop(running this command should prompt you with various options)
9. This finishes the Hadoop setup in stand-alone mode.
10. Let us run a sample hadoop programs that is provided to you in the download package:
    - $ mkdir input (create the input directory)
    - $ cp etc/hadoop /*.xml
    - input (copy over all the xml files to input folder)
    - $ bin/hadoop jar share/hadoop/mapreduce/
    - hadoop-mapreduce-examples-2.7.0.jar grep
    - input output 'dfs[a-z.]+' (grep/find all the files matching the pattern 'dfs[a-z.]+' and copy
    - those files to output directory)
    - $ cat output/* (look for the output in the output directory that Hadoop creates for you).
    - Hadoop Installation: Psuedo Distributed Mode(Locally)

Steps for Installation

1. Edit the file /home/Hadoop_dev/hadoop2/etc/hadoop/core-site.xml as below:

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Note: This change sets the namenodeip andport.

2. Edit the file /home/Hadoop_dev/hadoop2/etc/hadoop/hdfs-site.xml as below:

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

Note: This change sets the default replication count for blocks used by HDFS.

3. We need to setup password less login so that the master will be able to do a password-less ssh to start the daemons on all the slaves.

   Check if ssh server is running on your host or not:
   a. ssh local host (enter your password and if you are able to login then ssh server is running)
   b. In step a. if you are unable to login, then installssh as follows:

      sudo apt-get install ssh
   c. Setup password less login as below:
      i. ssh-keygen -t dsa -P " -f ~/.ssh/id_dsa
      ii. cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
4. We can run Hadoop jobs locally or on YARN inthis mode. In this Post, we will focus on running the jobs **locally.**
5. Format the file system. When we for matnamenode it formats the meta-data related to datanodes.

   By doing that, all the information on the data nodes are lost and they can be reused for new data
   a. bin/hdfsnamenode –format
   6. Start the daemons
   a. sbin/start-dfs.sh (Starts NameNode and DataNode)

      You can check If NameNode has started successfully or not by using the following web interface: http://0.0.0.0:50070 . If you are unable to see this, try to check the logs in the /home/ hadoop_dev/hadoop2/logs folder.
7. You can check whether the daemons are running or not by issuing Jps command.
8. This finishes the installation of Hadoop in pseudo distributed mode.

4. Let us run the same example we can in the previous blog post:
   i) Create a new directory on the hdfs
   bin/hdfsdfs -mkdir –p /user/hadoop_dev
   ii) Copy the input files for the program to hdfs:
   bin/hdfsdfs -put etc/hadoop input
   iii) Run the program:
   bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.jar grep input output 'dfs[a-z.]+'
   iv) View the output on hdfs:
   bin/hdfsdfs -cat output/*
10. Stop the daemons when you are done executing the jobs, with the below command:
   sbin/stop-dfs.sh

Hadoop Installation – PsuedoDistributed Mode(YARN)

Steps for Installation

1. Edit the file /home/hadoop_dev/hadoop2/etc/hadoop/mapred-site.xml as below:

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

2. Edit the fie /home/hadoop_dev/hadoop2/etc/hadoop/yarn-site.xml as below:

```
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
</configuration>
```

> **Note:** This particular configuration tellsMapReduce how to do its shuffle. In this case it uses the mapreduce_shuffle.

3. Format the NameNode:
   bin/hdfsnamenode –format
4. Start the daemons using the command:
   sbin/start-yarn.sh
   - This starts the daemons Resource Manager and Node Manager.
   - Once this command is run, you can check if Resource Manager is running or not by visiting the following URL on browser : http://0.0.0.0:8088 . If you are unable to see this, check for the logs in the directory: /home/hadoop_dev/hadoop2/logs
5. To check whether the services are running, issuea jps command. The following shows all the services necessary to run YARN on a single server:
   $ jps
   15933 Jps
   15567 ResourceManager
   15785 NodeManager
6. Let us run the same example as we ran before:
   i) Create a new directory on the hdfs

```
bin/hdfsdfs -mkdir –p /user/hadoop_dev
```

   ii) Copy the input files for the program to hdfs:

```
bin/hdfsdfs -put etc/hadoop input
```

   iii) Run the program:

```
bin/yarn jar share/hadoop/mapreduce/
hadoop-mapreduce-examples-2.6.0.jar grep
input output 'dfs[a-z.]+'
```

   iv) View the output on hdfs:

```
bin/hdfsdfs -cat output/*
```

7. Stop the daemons when you are done executing the jobs, with the below command:

```
sbin/stop-yarn.sh
```

This completes the installation part of Hadoop.

Basic Operations

Create a directory in HDFS at given path(s).

Usage

```
hadoop fs -mkdir<paths>
```

Example

```
hadoop fs -mkdir /user/saurzcode/dir1
/user/saurzcode/dir2
```

List the contents of a directory

Usage

```
hadoop fs -ls <args>
```

Example

```
hadoop fs -ls /user/saurzcode
Upload and download a file in HDFS.
Upload
hadoop fs -put:
```

Copy single src file, or multiple src files from local file system to the Hadoop data file system

Usage

```
hadoop fs -put <localsrc> ... <HDFS_dest_Path>
```

Example

```
hadoop fs -put /home/saurzcode/Samplefile.txt
/user/
saurzcode/dir3/
```

Download

hadoop fs -get

Copies/Downloads files to the local file system

Usage

```
hadoop fs -get <hdfs_src><localdst>
```

Example

```
hadoop fs -get /user/saurzcode/dir3/Samplefile.txt
/home/
```

See contents of a file

Same as unix cat command

Usage

```
hadoop fs -cat <path[filename]>
```

Example

```
hadoop fs -cat /user/saurzcode/dir1/abc.txt
-Copy from Hadoop File system and deleting file.
```

Copy a file from source to destination

This command allows multiple sources as well in which case the destination must be a directory.

Usage

```
hadoop fs -cp <source><dest>
```

Example

```
hadoop fs -cp /user/saurzcode/dir1/abc.txt
/user/saurzcode/dir2
```

Remove files specified as argument. Deletes directory only when it is empty

Usage

```
hadoop fs -rm <arg>
```

Example

```
hadoop fs -rm /user/saurzcode/dir1/abc.txt
Recursive version of delete.
```

Usage

```
hadoop fs -rmr<arg>
```

Example

```
hadoop fs -rmr /user/saurzcode/
```

-Copying File to Hadoop

Copy a file from/To Local file system to HDFS copy From Local

Usage

```
hadoop fs -copyFromLocal<localsrc> URI
```

Example

```
hadoop fs -copyFromLocal /home/saurzcode/abc.txt
/user/
saurzcode/abc.txt
```

Similar to put command, except that the source is restricted toa local file reference.

Copy To Local

Usage

```
hadoop fs -copyToLocal [-ignorecrc] [-crc] URI
<localdst>
```

Similar to get command, except that the destination is restricted to a local file reference.

-Moving and displaying files in HDFS.

Move file from source todestination.

> **Note :** Moving files across file system is not permitted.

Usage

```
hadoop fs -mv <src><dest>
```

Example

```
hadoop fs -mv /user/saurzcode/dir1/abc.txt
/user/saurzcode/
dir2
```

Remove a file or directory inHDFS:

Remove files specified as argument. Deletes directory only when it is empty

Usage :

```
hadoop fs -rm <arg>
```

Example:

```
hadoop fs -rm /user/saurzcode/dir1/abc.txt
```

Display last few lines of a file:

Similar to tail command in Unix.

Usage :

```
hadoop fs -tail <path[filename]>
```

Example:

```
hadoop fs -tail /user/saurzcode/dir1/abc.txt
```

Display the aggregate length of a file:

Usage :

```
hadoop fs -du <path>
```

Example:

```
hadoop fs -du /user/saurzcode/dir1/abc.txt
```

Experiment No. 2

Use of Sqoop tool to transfer data between Hadoop and relational database servers.

a. Sqoop - Installation.

b. To execute basic commands of Hadoop eco system componentSqoop.

Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and external datastores such as relational databases, enterprise data warehouses. Sqoop is used to import data from external datastores into Hadoop Distributed File System or related Hadoop eco-systems like Hive and HBase. Similarly, Sqoop can also be used to extract data from Hadoop or its eco-systems and export it to external data stores such as relational databases, enterprise data warehouses. Sqoop works with relational databases such as Teradata, Netezza, Oracle, MySQL, Postgres etc. Sqoop provides many salient features like:

1. Full Load
2. Incremental Load
3. Parallel import/export
4. Import results of SQL query
5. Compression
6. Connectors for all major RDBMS Databases
7. Kerberos Security Integration
8. Load data directly into Hive/Hbase
9. Support for Accumulo

- Download Sqoop from this apache
- visit the official site (https://sqoop.apache.org/)
- After download, we need to extract the files. Open the command prompt and go the folder where Sqoop is downloaded. Then run the following command to extract files:
- $ cd /path/to/sqoop/downloaded/file
- $ tar -xvzf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
- Next step is to add environment variables.
- In the start menu, type "environment variables" and click on the Environment Variable tab.
- Refer the below screenshots to add the path.

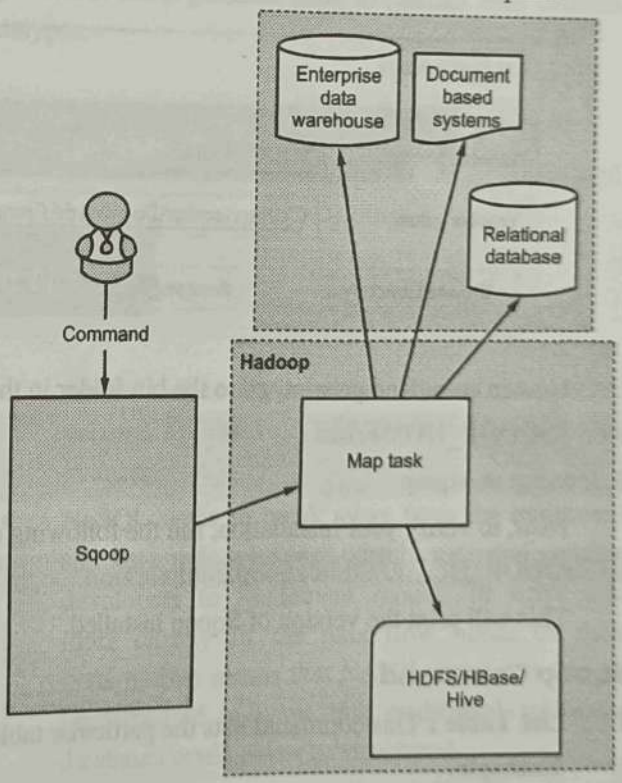
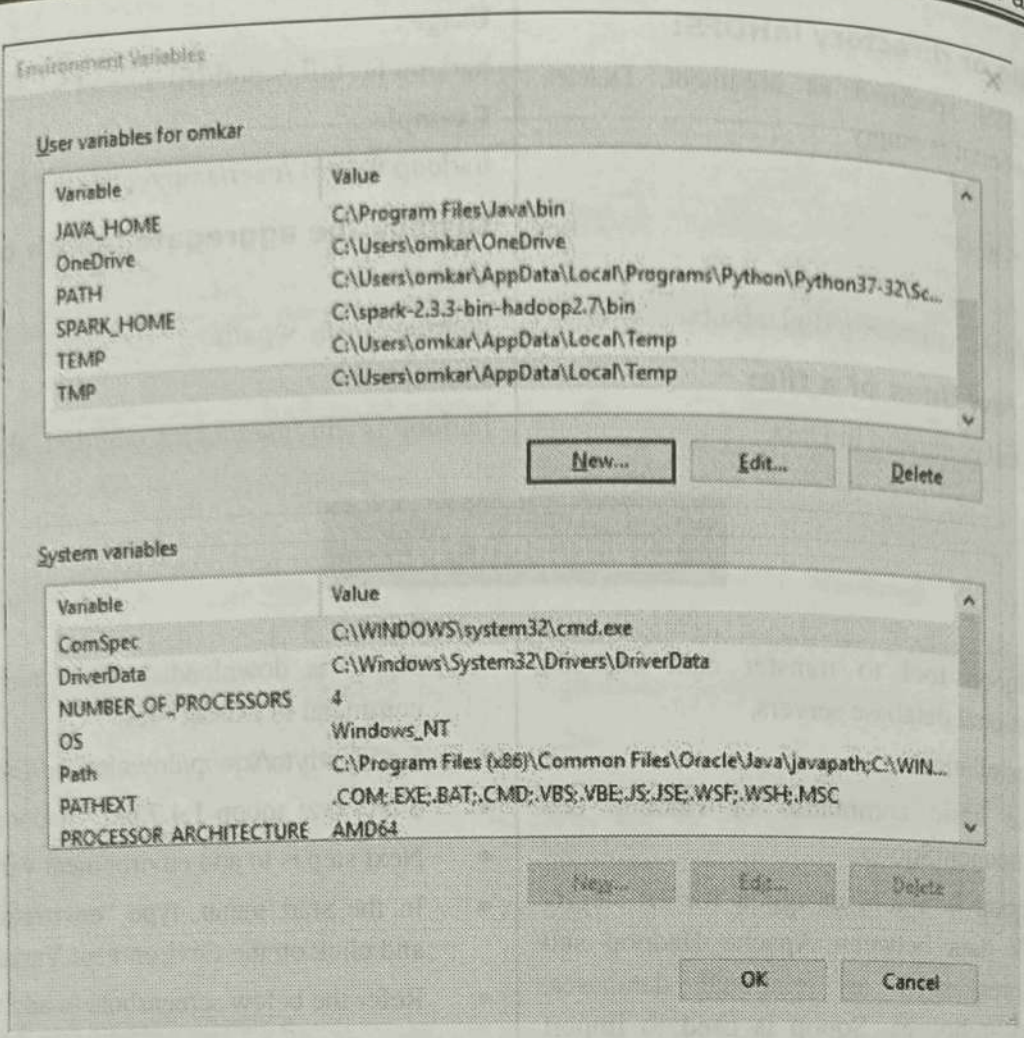
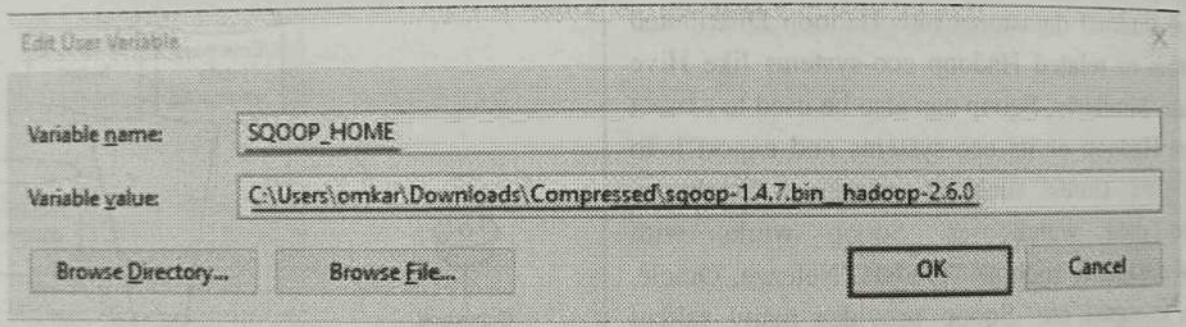


Fig. L1 : Sqoop Architecture





Now in command prompt, go to the bin folder in the Sqoop folder and run the following command:

```
$ cd $SQOOP_HOME/bin
$ ./configure-sqoop
```

Now, to verify your installation, run the following command:

```
$ %SQOOP_HOME%\bin\sqoop.cmd version
```

This will print the version of Sqoop installed.

Sqoop Command

1. **List Table :** This command lists the particular table of the database in MYSQL server.

   Example:

```
sqoop list - tables --connect jdbc:mysql://localhost/payment --username gatne
```

2. **Target directory :** This command import table in a specific directory in HDFS. -m denotes mapper argument. They have an integer value.

Example:

```
$sqoop import --connect jdbc:mysql://db.foo.com:3306/bar\    --table Employees\
```

3. **sqoop-eval :** This command runs quickly SQL queries of the respective database.

Example:

```
$ sqoop eval --connect --query "SQLQuery"
```

4. **sqoop – version :** This command displays version of the sqoop.

Example:

```
$ sqoop version   sqoop {revnumber}
```

5. **sqoop-job :** This command allows us to create a job, the parameters that are created can be invoked at any time. They take options like (–create,–delete,–show,–exit).

Example:

```
$ sqoop job --create --import --connect  --table
```

6. **List Database :** This Sqoop command lists have all the available database in the RDBMS server.

Example:

```
$ sqoop list - database -- connect
```

7. **code gen :** This Sqoop command creates java class files which encapsulate the imported records. All the java files are recreated, and new versions of a class are generated. They generate code to interact with database records. Retrieves a list of all the columns and their datatypes.

Example:

```
$ sqoop codegen --connect -table
```

Experiment No. 3

To install and configure MongoDB/ Cassandra/ HBase/ Hypertable to execute NoSQL commands

- *NoSQL* stands for "Not only SQL," to emphasize the fact that NoSQL databases are an alternative to SQL and can, in fact, apply SQL-like query concepts.
- NoSQL covers any database that is not a traditional relational database management system (RDBMS). The motivation behind NoSQL is mainly simplified design, horizontal scaling, and finer control over the availability of data. NoSQL databases are more specialized for types of data, which makes them more efficient and better performing than RDBMS servers in most instances.
- NoSQL seeks to break away from the traditional structure of relational databases, and enable developers to implement models in ways that more closely fit the data flow needs of their system. This means that NoSQL databases can be implemented in ways that traditional relational databases could never be structured.

Document Store Databases

- Document store databases apply a document-oriented approach to storing data. The idea is that all the data for a single entity can be stored as a document, and documents can be stored together in collections.
- A document can contain all the necessary information to describe an entity. This includes the capability to have subdocuments, which in RDBMS are typically stored as an encoded string or in a separate table. Documents in the collection are accessed via a unique key.

Key-Value Databases

- The simplest type of NoSQL database is the key-value stores. These databases store data in a completely schema-less way, meaning that no defined structure governs what is being stored. A key can point to any type of data, from an object, to a string value, to a programming language function.
- The advantage of key-value stores is that they are easy to implement and add data to. That makes them great to implement as simple storage for storing and retrieving data based on a key. The downside is that you cannot find elements based on the stored values.

Column Store Databases

- Column store databases store data in columns within a key space. The key space is based on a unique name, value, and timestamp. This is similar to the key-value databases; however, column store databases are geared toward data that uses a timestamp to differentiate valid content from stale content. This provides the advantage of applying aging to the data stored in the database.

Graph Store Databases

Graph store databases are designed for data that can be easily represented as a graph. This means that elements are interconnected with an undetermined number of relations between them, as in examples such as family and social relations, airline route topology, or a standard road map.

Choosing RDBMS, NoSQL, or Both

- When investigating NoSQL databases, keep an open mind regarding which database to use and how to apply it. This is especially true with high-performance systems.
- You might need to implement a strategy based on only RDBMS or NoSQL—or you might find that a combination of the two offers the best solution in the end.

The following are some additional reasons MongoDB has become the most popular NoSQL database:

- **Document oriented:** Because MongoDB is document oriented, the data is stored in the database in a format that is very close to what you will be dealing with in both server-side and client-side scripts. This eliminates the need to transfer data from rows to objects and back.
- **High performance:** MongoDB is one of the highest-performing databases available. Especially in today's world, where many people interact with websites, having a back end that can support heavy traffic is important.
- **High availability:** MongoDB's replication model makes it easy to maintain scalability while keeping high performance and scalability.
- **High scalability:** MongoDB's structure makes it easy to scale horizontally by sharding the data across multiple servers.
- **No SQL injection:** MongoDB is not susceptible to SQL injection (putting SQL statements in web forms or other input from the browser that compromises the DB security) because objects are stored as objects, not by using SQL strings.

Understanding Collections

- MongoDB groups data through collections. A collection is simply a grouping of documents that have the same or a similar purpose. A collection acts similarly to a table in a traditional SQL database. However, it has a major difference: In MongoDB, a collection is not enforced by a strict schema.
- Instead, documents in a collection can have a slightly different structure from one another, as needed. This reduces the need to break items in a document into several different tables, as is often done in SQL implementations.

Understanding Documents

- A document is a representation of a single entity of data in the MongoDB database. A collection consists of one or more related objects. A major difference exists between MongoDB and SQL, in that documents are different from rows.
- Row data is flat, with one column for each value in the row. However, in MongoDB, documents can contain embedded subdocuments, providing a much closer inherent data model to your applications.
- For example, a document in MongoDB might be structured similar to the following, with name, version, languages, admin, andpaths fields:

```
{
  name: "New Project",
  version: 1,
  languages: ["JavaScript", "HTML", "CSS"],
  admin: {name: "Brad", password: "****"},
  paths: {temp: "/tmp", project:"/opt/project",
html: "/opt/project/html"}
}
```

Notice that the document structure contains fields/properties that are strings, integers, arrays, and objects, just as in a JavaScript object. Table 1 lists the different data types for field values in the BSON document.

Table 1 : MongoDB Data Types and Corresponding ID Number

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object ID	7
Boolean	8
Date	9
Null	10
Regular expression	11
javaScript	13
Symbol	14
javaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

The field names cannot contain null characters, dots (.), or dollar signs ($). In addition, the _id field name is reserved for the Object ID. The _id field is a unique ID for the system that consists of the following parts:

- A 4-byte value representing the seconds since the last epoch
- A 3-byte machine identifier
- A 2-byte process ID
- A 3-byte counter, starting with a random value

The maximum size of a document in MongoDB is 16MB, to prevent queries that result in an excessive amount of RAM or intensive hits to the file system. You might never come close to this, but you still need to keep the maximum document size in mind when designing some complex data types that contain file data into your system.

MongoDB Data Types

- MongoDB assigns each data type of an integer ID number from 1 to 255 when querying by type. Table 1 lists the data types MongoDB supports, along with the number MongoDB uses to identify them.
- When comparing values of different BSON types, MongoDB uses the following comparison order, from lowest to highest:
  1. Min key (internal type)
  2. Null
  3. Numbers (32-bit integer, 64-bit integer, double)
  4. Symbol, String
  5. Object
  6. Array
  7. Binary data
  8. Object ID
  9. Boolean
  10. Date, timestamp
  11. Regular expression
  12. Max key (internal type)

Installing and running the MongoDB server

MongoDB can be downloaded from the following webpage: https://www.mongodb.org/downloads Put the downloaded file wherever you prefer, open a terminal, go to the directory where you put the downloaded file, then type the following commands

```
tar -xzf mongodb-linux-x86_64-2.6.5.tgz
cd mongodb-linux-x86_64-2.6.5
sudomkdir –p /data/dbsudo ./bin/mongod
```

Interacting with the MongoDB server

In order to interact with the MongoDB server, a client is necessary. The client will connect to the server on port 27017 and it will send the commands through the connection. In this lab, we will use the Java client; these are the steps to follow for using such a client:

- Download the jar of the Jedis client: http://central.maven.org/maven2/org/mongodb/mongo-javadriver/2.12.4/mongo-java-driver-2.12.4.jar o alternatively, check this page http://docs.mongodb.org/ecosystem/drivers/java/
- Put the jar in the same directory of your Java program;
- In your Java program, import the Jedis class: import com.mongodb.*;
- In your Java program, use the Jedis client, e.g.: MongoClient mongo = new MongoClient("localhost", 27017);
- Compile your Java program: javac -cp mongo-java-driver-2.12.4.jar myProgram.java
- Run your Java program (that includes the Redis client): java -cp '.: mongo-java-driver-2.12.4.jar' myProgram

1. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.


```
{
"address": {
"building": "1007",
"coord": [ -73.856077, 40.848447 ],
"street": "Morris Park Ave",
"zipcode": "10462"
},
"borough": "Bronx",
"cuisine": "Bakery",
"grades": [
{ "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
{ "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
{ "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
{ "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
{ "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
],
"name": "Morris Park Bake Shop",
"restaurant_id": "30075445"
}
```


2.

```
Product Catalog{
type: "Audio Album",
title: "A Love Supreme",
description: "by John Coltrane",
shipping: {
```

```
weight: 6, dimensions:
{ width: 10,
height: 10,
depth: 1 }, },
pricing: {
list: 1200,
retail: 1100,
savings: 100,
pct_savings: 8 },
details:{
title: "A Love Supreme [Original Recording Reissued]",
artist: "John Coltrane",
genre: [ "Jazz", "General" ],
tracks: [ "A Love Supreme Part I:
Acknowledgement", "A Love Supreme Part II - Resolution", "A Love Supreme, Part III: Pursuance", "A Love
Supreme, Part IV-Psalm" ], }, }
```

Industrial Application

- Managing content, posts, images and videos on social media platforms.
- Analyzing customer data in real-time for improving business performance

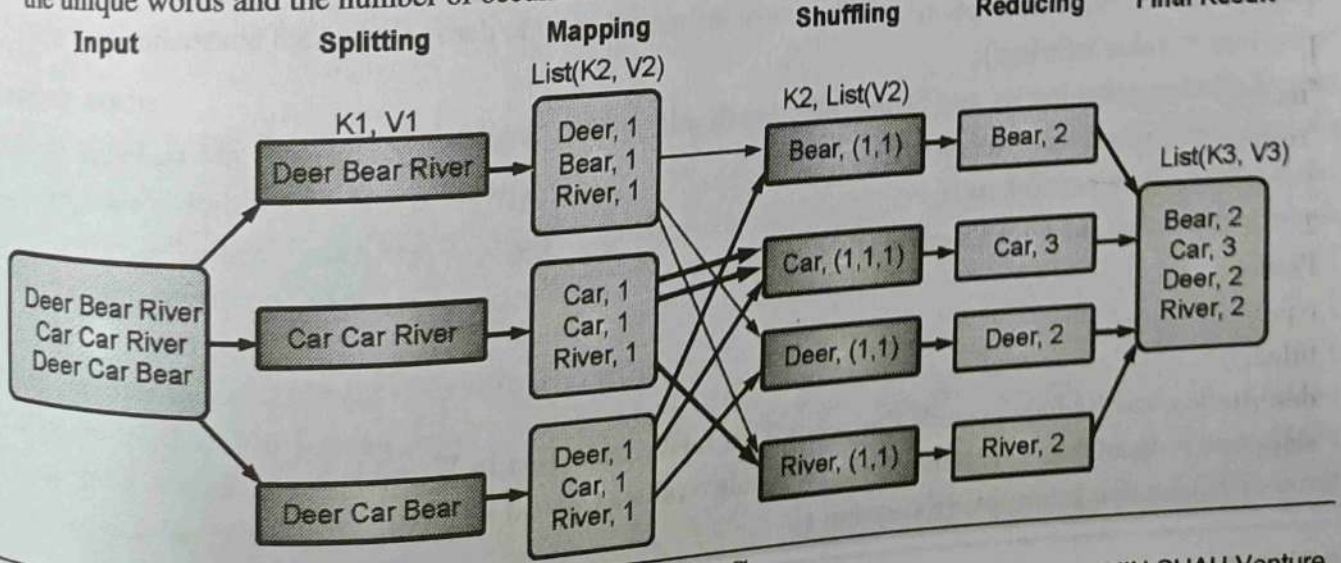
Experiment No. 4

Experiment on Hadoop Map-Reduce

Write a program to implement a word count program using MapReduce.

A Word Count Example of MapReduce

- Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:
- Dear, Bear, River, Car, Car, River, Deer, Car and Bear
- Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- First, we divide the input in three splits as shown in the figure. This will distribute the work among all the map nodes. Then, we tokenize the words in each of the mapper and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once. Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After mapper phase, a partition process takes place where sorting and shuffling happens so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Source code

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;
public class WordCount
{
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> {
public void map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
}
}

public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable> {
public void reduce(Text key, Iterable<IntWritable>values,Context context)
throws IOException,InterruptedException {
```

```
int sum=0;
for(IntWritable x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}
public static void main(String[] args) throws Exception {
Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//deleting the output path automatically from hdfs so that we don't have to
delete it explicitly
outputPath.getFileSystem(conf).delete(outputPath);
//exiting the job only if the flag value becomes false
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

We will understand the code for each of these three parts sequentially.

Mapper code

```
public static class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
public void map(LongWritable key, Text value, Context context) throws
IOException,InterruptedException {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
```

- We have created a class Map that extends the classMapper which is already defined in the MapReduceFramework.
- We define the data types of input and output key/value pair after the class declaration using angle brackets.
- Both the input and output of the Mapper is a key/valuepair.

Input

- The key is nothing but the offset of each line in the text file:LongWritable
- The value is each individual line :Text

Output

- The key is the tokenized words: Text
- We have the hardcoded value in our case which is
- 1: IntWritable
- **Example** – Dear 1, Bear 1, etc.
- We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to 1.

Reducer Code

```
public static class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> {
public void reduce(Text key, Iterable<IntWritable>values,Context
context)
throws IOException,InterruptedException {
int sum=0;
for(IntWritable x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}}
```

- We have created a class Reduce which extends classReducer like that of Mapper.
- We define the data types of input and output key/valuepair after the class declaration using angle brackets as done for Mapper.
- Both the input and the output of the Reducer is a key/value pair.

Input:

- The key nothing but those unique words which have been generated after the sorting and shuffling phase: Text
- The value is a list of integers corresponding to each key: IntWritable
- **Example** – Bear, [1, 1], etc.

Output

- The key is all the unique words present in the inputtext file: Text
- The value is the number of occurrences of each ofthe unique words: IntWritable
- **Example** – Bear, 2; Car, 3, etc.
- We have aggregated the values present in each of the list corresponding to each key and produced the final answer.
- In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

Driver Code

```
Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

- In the driver class, we set the configuration of ourMapReduce job to run in Hadoop.
- We specify the name of the job , the data type of input/output of the mapper and reducer.
- We also specify the names of the mapper and reducerclasses.
- The path of the input and output folder is also specified.
- The method setInputFormatClass () is used for specifyingthat how a Mapper will read the input data or what willbe the unit of work. Here, we have chosen
- TextInputFormat so that single line is read by the mapperat a time from the input text file.
- The main () method is the entry point for the driver. Inthis method, we instantiate a new Configuration objectfor the job.

Run the MapReduce code

The command for running a MapReduce code is:

hadoop jar hadoop-mapreduce-example.jar WordCount /sample/input /sample/output

Experiment No. 5

Experiment on Hadoop Map-Reduce

Implementing simple algorithms in Map-Reduce: Matrix multiplication, Aggregates,

Joins, Sorting, Searching, etc

Aggregation Example

- There are several methods of performing aggregations in MongoDB. These examples cover the new aggregation framework, using map reduce and using the group method.

Setup

To start, we'll insert some example data which we can perform aggregations on:

Aggregation Framework

- This example shows how to use the aggregate() method to use the aggregation framework. We'll perform a simple aggregation to count the number of occurrences for each tag in the tags array, across the entire collection. To achieve this we need to pass in three operations to the pipeline. First, we need to unwind the tags array, then group by the tags and sum them up, finally we sort by count.As python dictionaries don't maintain order you should use SON or collections.OrderedDict
- To run an explain plan for this aggregation use the command() method:

- As well as simple aggregations the aggregation framework provides projection capabilities to reshape the returned data. Using projections and aggregation, you can add computed fields, create new virtual sub-objects, and extract sub-fields into the top-level of results.

Map/Reduce

- Another option for aggregation is to use the map reduce framework. Here we will define map and reduce functions to also count the number of occurrences for each tag in the tags array, across the entire collection.
- Our map function just emits a single (key, 1) pair for each tag in the array:
- The reduce function sums over all of the emitted values for a given key:

Note

- We can't just return values.length as the reduce function might be called iteratively on the results of other reduce steps.
- Finally, we call map_reduce() and iterate over the result collection:

Advanced Map/Reduce

- PyMongo's API supports all of the features of MongoDB's map/reduce engine. One interesting feature is the ability to get more detailed results when desired, by passing full_response=True to map_reduce(). This returns the full response to the map/reduce command, rather than just the result collection:
- All of the optional map/reduce parameters are also supported, simply pass them as keyword arguments. In this example we use the query parameter to limit the documents that will be mapped over:
- With MongoDB 1.8.0 or newer you can use SON or collections. OrderedDict to specify a different database to store the result collection.

Definitions: Matrix Multiplication

- M is a matrix with element mij in row i and column j.
- N is a matrix with element njk in row j and column k.
- P is a matrix = MN with element pik in row i and column k, where pik =Σj mijnjk

Relational Representations

- M = M(I, J, V), with tuples (i, j, mij)
- N = N(J, K, W), with tuples (j, k, njk)

Explanation

- The product MN is almost a natural join followed by grouping and aggregation. That is, the natural join of M(I, J, V) and N(J, K, W), having only attribute J in common, would produce tuples (i, j, k, v, w) from each tuple (i, j, v) in M and tuple (j, k, w) in N.
- This five-component tuple represents the pair of matrix elements (m_{ij} , n_{jk}). What we want instead is the product of these elements, that is, the four-component tuple (i, j, k, v × w), because that represents the product mijnjk.
- Once we have this relation as the result of one map-reduce operation, we can perform grouping and aggregation, with I and K as the grouping attributes and the sum of V ×W as the aggregation. That is, we can implement matrix multiplication as the cascade of two MapReduce operations, as follows.
- UPDATE: Please note that the Mapper function does not have access to the Row Number (in this case i, and k) directly. An extra MapReduce Job has to be run initially in order to add the Row Number as Key to every row. The process of generating the Row Number is explained in the next post.

First Iteration

The Map Function:

- For each matrix element m_{ij} emit the key value pair j, (M, i, m_{ij}).
- For each matrix element n_{jk} emit the key value pair j, (N, k, n_{jk}).
- The Reduce Function:
- For each key j, for each value that comes from M, say (M, i, m_{ij}), and each value that comes from N, say (N, k, n_{jk}), emit the key value pair j, (i, k, $m_{ij}n_{jk}$).
- The output of this Reduce function is used as the input to the Map function in the next iteration.

Second Iteration

The Map Function

- For each key value pair j, (i, k, $m_{ij}n_{jk}$), emit the key value pair (i, k), $m_{ij}n_{jk}$.

The Reduce Function:

- For each key (i, k), emit the key value pair (i, k), v, where v is the sum of the list of values associated with this key and is the value of the element in row i and column k of the matrix P = MN
- When processing large data sets the need for joining data by a common key can be very useful, if not essential. By joining data you can further gain insight such as joining with timestamps to correlate events with a time a day. The need for joining data are many and varied. We will be covering 3 types of joins, Reduce-Side joins, Map-Side joins and the Memory-Backed Join over 3 separate posts. This installment we will consider working with Reduce-Side joins.

Reduce Side Joins

- Of the join patterns we will discuss, reduce-side joins are the easiest to implement. What makes reduce-side joins straight forward is the fact that Hadoop sends identical keys to the same reducer, so by default the data is organized for us.
- To perform the join, we simply need to cache a key and compare it to incoming keys. As long as the keys match, we can join the values from the corresponding keys. The trade off with reduce-side joins is performance, since all of the data is shuffled across the network.
- Within reduce-side joins there are two different scenarios we will consider: one-to-one and one-to-many. We'll also explore options where we don't need to keep track of the incoming keys; all values for a given key will be grouped together in the reducer.

One-To-One Joins

- A one-to-one join is the case where a value from dataset 'X' shares a common key with a value from dataset 'Y'. Since Hadoop guarantees that equal keys are sent to the same reducer, mapping over the two datasets will take care of the join for us. Since sorting only occurs for keys, the order of the values is unknown.
- We can easily fix the situation by using secondary sorting. Our implementation of secondary sorting will be to tag keys with either a "1" or a "2" to determine order of the values. We need to take a couple extra steps to implement our tagging strategy.
- Implementing a Writable Comparable

MAP

1. Spitting our data and creating a List of the values
2. Remove the join key from the list
3. Re-join the data back into a single String
4. Set the join key, join order and the remaining data
5. Write out the data

Reduce

1. First, we create a Guava Splitter on line 5 that will split strings by a "/".
2. Then on lines 8-10 we are setting the index of our join key and the separator used in the files.
3. In lines 12-17 we setting the tags for the input files to be joined. The order of the file names on the command line determines their position in the join. As we loop over the file names from the command line, we split the whole file name and retrieve the last value (the base filename) via the Guava Iterables.getLast() method. We then call config.set() with the filename as the key and we use i + 1 as the value, which sets the tag or join order. The last value in the args array is skipped in the loop, as that is used for the output path of our MapReduce job. On the last line of the loop we append each file path in a StringBuilder which is used later to set the input paths for the job.
4. We only need to use one mapper for all files, the JoiningMapper, which is set.
5. Set our custom partitioner and group comparator (respectively) which ensure the arrival order of keys and values to the reducer and properly group the values with the correct key

```
import java.io.IOException;
import java.util.*;
import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class TwoStepMatrixMultiplication {
public static class Map extends Mapper<LongWritable, Text,
Text, Text> {
public void map(LongWritable key, Text value, Context
context) throws IOException, InterruptedException {
String line = value.toString();
String[] indicesAndValue = line.split(",");
Text outputKey = new Text();
Text outputValue = new Text();
if (indicesAndValue[0].equals("A")) {
outputKey.set(indicesAndValue[2]);
outputValue.set("A," + indicesAndValue[1] + "," +
indicesAndValue[3]);
context.write(outputKey, outputValue);
} else {
outputKey.set(indicesAndValue[1]);
outputValue.set("B," + indicesAndValue[2] + "," +
indicesAndValue[3]);
context.write(outputKey, outputValue);
}
}
}
public static class Reduce extends Reducer<Text, Text, Text,
Text> {
public void reduce(Text key, Iterable<Text> values,
Context context) throws IOException, InterruptedException {
String[] value;
ArrayList<Entry<Integer, Float>> listA = new
ArrayList<Entry<Integer, Float>>();
ArrayList<Entry<Integer, Float>> listB = new
ArrayList<Entry<Integer, Float>>();
for (Text val : values) {
```

```
value = val.toString().split(",");
if (value[0].equals("A")) {
listA.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
} else {
listB.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));

}
}
String i;
float a_ij;
String k;
float b_jk;
Text outputValue = new Text();
for (Entry<Integer, Float> a : listA) {
i = Integer.toString(a.getKey());
a_ij = a.getValue();
for (Entry<Integer, Float> b : listB) {
k = Integer.toString(b.getKey());
b_jk = b.getValue();
outputValue.set(i + "," + k + "," +
Float.toString(a_ij*b_jk));
context.write(null, outputValue);
}
}
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = new Job(conf,
"MatrixMatrixMultiplicationTwoSteps");
job.setJarByClass(TwoStepMatrixMultiplication.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path("hdfs://
127.0.0.1:9000/matrixin"));
FileOutputFormat.setOutputPath(job, new Path("hdfs://
127.0.0.1:9000/matrixout"));
job.waitForCompletion(true);
}
}
```

- Creating Jar file for the Matrix Multiplication. $ jar -cvf MatrixMultiply.jar
- Uploading the M, N file which contains the matrix multiplication data to HDFS.
- $ hadoop fs -mkdir Matrix/ $ hadoop fs -copyFromLocal M Matrix/ $ hadoop fs -copyFromLocal N Matrix/
- _Executing the jar file using hadoop command and thus how fetching record from HDFS and storing output in HDFS.
- $ hadoop jar MatrixMultiply.jar www.ehadoopinfo.com.MatrixMultiply Matrix/* result/

Experiment No. 6

Create HIVE Database and Descriptive analytics-basic statistics

Databases in Hive

- The Hive concept of a database is essentially justa catalog or namespace of tables. However, they are very useful forlarger clusters with multiple teams and users, as a way of avoidingtable name collisions. It's also common to use databases to organizeproduction tables into logical groups.If you don't specify a database, the default database is used.
- The simplest syntax for creating a database is shown in thefollowing example:

```
hive> CREATE DATABASE financials;
```

- Hive will throw an error if financials already exists.

```
hive> CREATE DATABASE IF NOT EXISTS financials;
```

- While normally you might like to be warned if a database of the same name already exists, the IF NOT EXISTS clause is useful for scripts that should create a database on-the-fly, if necessary, before proceeding.
- You can also use the keyword SCHEMA instead of DATABASE in all the database-related commands. At any time, you can see the databases that already exist as follows:

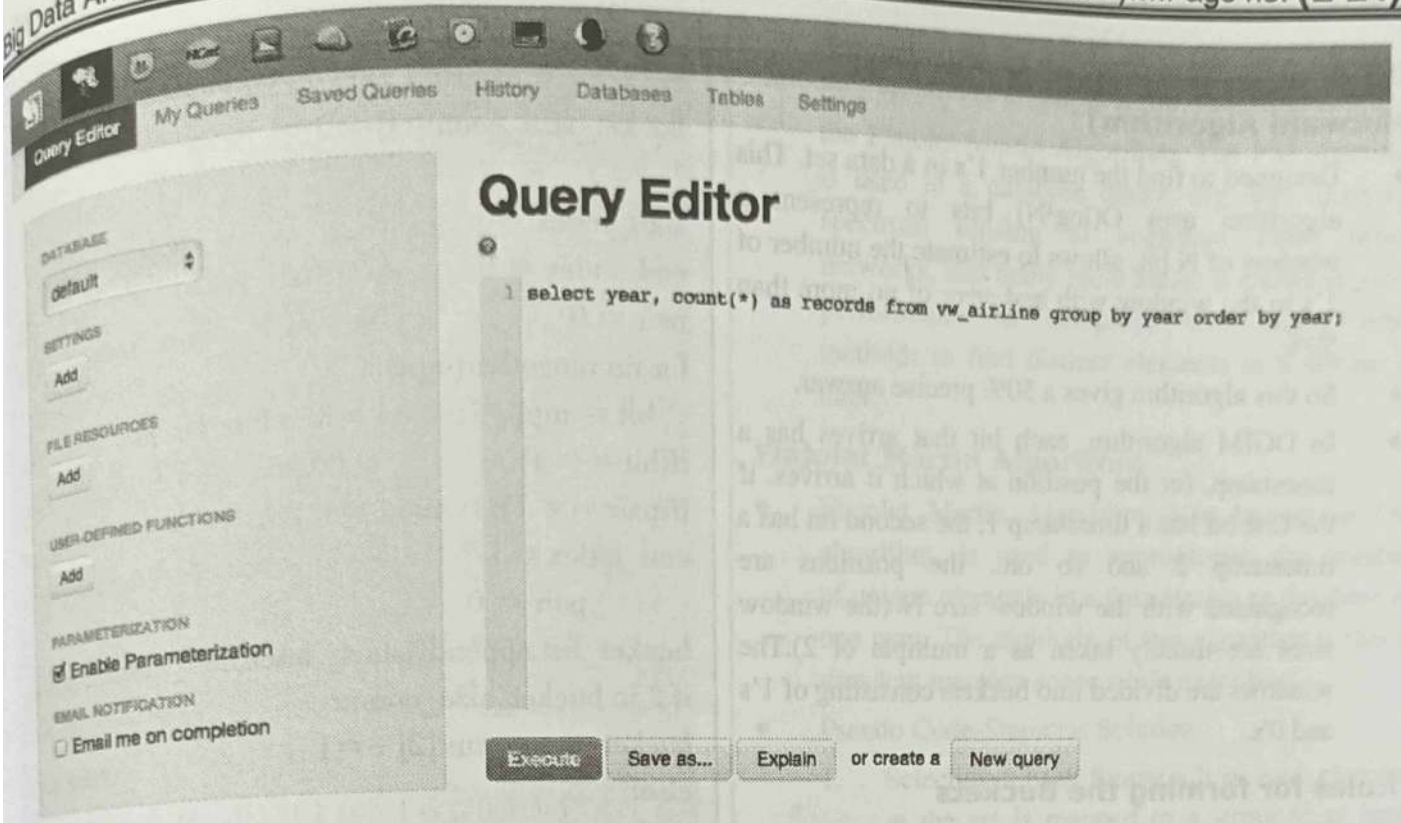
```
hive> SHOW DATABASES;
default
financials
```

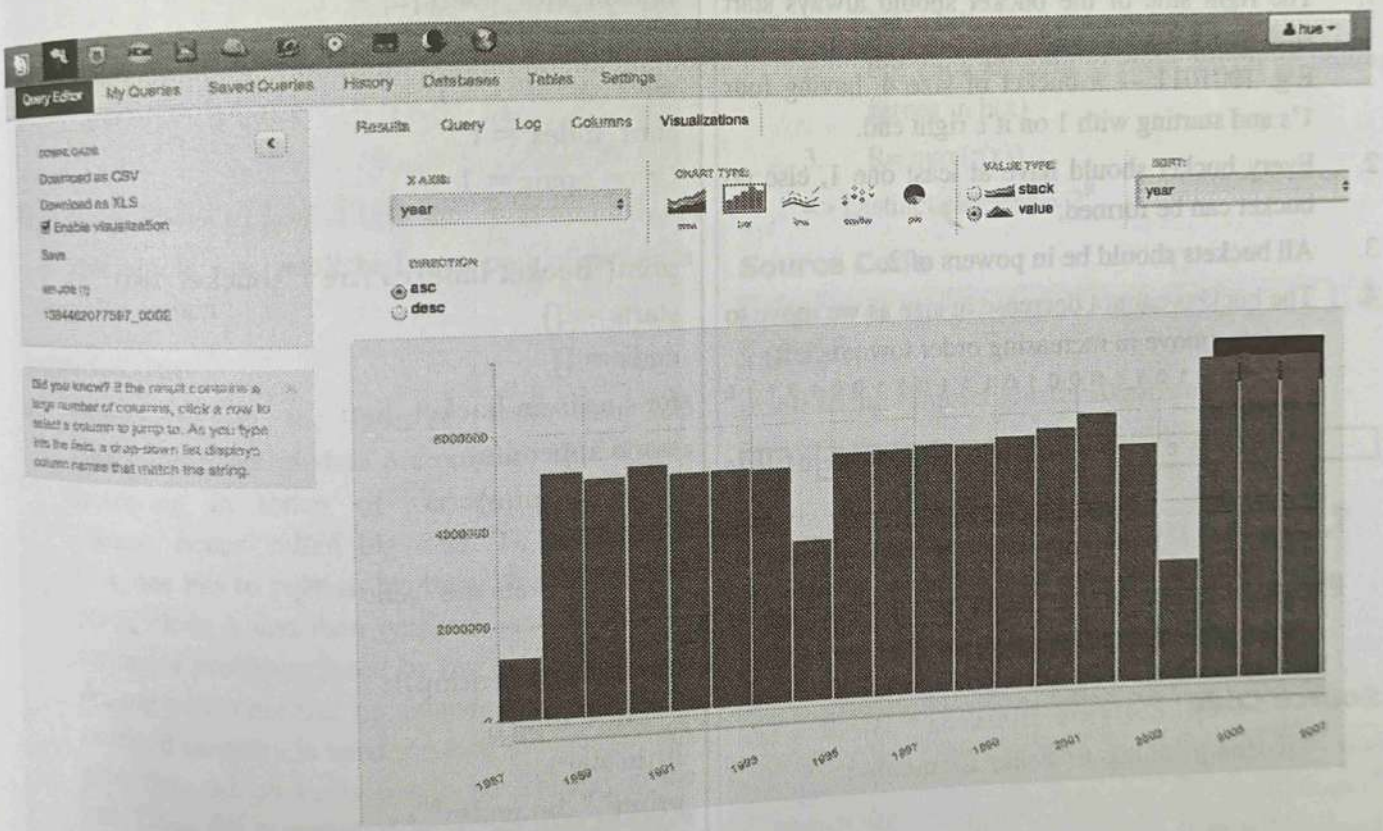
- You can override this default location for the new directory as shown in this example:

```
hive> CREATE DATABASE financials
> LOCATION '/my/preferred/directory';
```

- You can add a descriptive comment to the database, which will be shown by the DESCRIBE DATABASE <database> command.

```
hive> CREATE DATABASE financials
> COMMENT 'Holds all financial tables';
hive> DESCRIBE DATABASE financials;
financials Holds all financial tables
hdfs://master-server/user/hive/warehouse/financials.db
```





Experiment No. 7

Data Stream Algorithms (any one):

- - Implementing DGIM algorithm using any Programming Language
- - Implement Bloom Filter using any programming language
- Implement Flajolet Martin algorithm using any programming language

DGIM algorithm (Datar-Gionis-Indyk-Motwani Algorithm)

- Designed to find the number 1's in a data set. This algorithm uses $O(\log^2 N)$ bits to represent a window of N bit, allows to estimate the number of 1's in the window with and error of no more than 50%.
- So this algorithm gives a 50% precise answer.
- In DGIM algorithm, each bit that arrives has a timestamp, for the position at which it arrives. if the first bit has a timestamp 1, the second bit has a timestamp 2 and so on.. the positions are recognized with the window size N (the window sizes are usually taken as a multiple of 2).The windows are divided into buckets consisting of 1's and 0's.

Rules for forming the Buckets

1. The right side of the bucket should always start with 1. (if it starts with a 0,it is to be neglected) E.g. 1001011 → a bucket of size 4 ,having four 1's and starting with 1 on it's right end.
2. Every bucket should have at least one 1, else no bucket can be formed.
3. All buckets should be in powers of 2.
4. The buckets cannot decrease in size as we move to the left. (move in increasing order towards left)

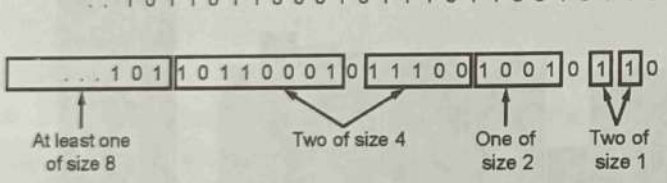


Fig. 4.2 : A bit-stream divided into buckets following the DGIM rules

Source Code

```
inp = list(map(int,input("Enter Elements :
").split()))
bucket_list = []
bucket_size_count = {}

def checker():
for ctin bucket_size_count.keys():
if(bucket_size_count[ct]>2):
        s2,e2,size2 = bucket_list.pop(-2)
        s1,e1,size1 = bucket_list.pop(-2)
bucket_list.insert(-1,(s1,e2,size1*2))
bucket_size_count[ct]-=2

start_index =0
end_index = 0
pair = 0
for iin range(len(inp)):
   bit = inp[i]
if(bit == 1):
if(pair == 1):
end_index = i
        pair = 0
bucket_list.append((start_index,end_index,2))
if 2 in bucket_size_count:
bucket_size_count[2]+=1
else:
bucket_size_count[2] = 1
checker()
else:
start_index = i
        pair = 1

print("Bucket Indexes Are : ",bucket_list)
starts = []
ends = []
for s,e,sizein bucket_list:
starts.append(s)
ends.append(e)

print("Buckets are ",end="")

for iin range(len(inp)):
   bit = inp[i]
if(iin starts):
print(" " ,bit,end="")
elif(iin ends):
print(bit,end= " ")
else:
print(bit,end= " ")

k = int(input("Enter k : "))
length = len(inp)
```

```
bound1 = length-1-k
bound2 = length-1

ones_count = 0

for s,e,sizein bucket_list[::-1]:
if(s<bound1 and e < bound1):
break
elif(s<=bound1 and e >= bound1):
ones_count +=int(size/2)
elif(s>=bound1 and e >= bound1):
ones_count += size

print("Number of Onces in Last",k,"bits are
",ones_count)
```

Output:

```
Enter Elements : 1 1 0 1 0 1 1 0 0 0 1 1 1
Bucket Indexes Are : [(0, 5, 4), (6, 10, 2), (11, 12,
2)]
Buckets are 1 1 0 1 0 1 1 0 0 0 1 1 1
Enter k : 3
Number of Onces in Last 3 bits are 3
```

This article was published as a part of the Data Science Blogathon

Introduction

- Every day on the internet, more than 2.5 quintillion bytes of data are created. This data is increasing in terms of variety, velocity and volume, hence called big data. To analyze this data, one has to collect this data, store it in a safe place, clean it and then perform analysis. One of the major problems faced by big data engineers is dealing with unuseful or redundant data. A lot of time and memory is used to store and analyze this extra data which turns out to be fruitless in the end. Thus, the removal of duplicate data becomes extremely essential to cut the analysis cost and reduce redundancy.
- Data cleaning can be done using various techniques but before cleaning the data, it is necessary to know the amount of useful data present in the dataset. Therefore, before the removal of duplicate data from a data stream or database, it is necessary to have knowledge of distinct or unique data present. A way to do so is by hashing the elements of the universal set using the Flajolet Martin Algorithm. The FM algorithm is used in a database query, big data analytics, spectrum sensing in cognitive radio sensor networks, and many more areas. It shows superior performance as compared with many other methods to find distinct elements in a stream of data.

Flajolet Martin Algorithm

- Flajolet Martin Algorithm, also known as FM algorithm, is used to approximate the number of unique elements in a data stream or database in one pass. The highlight of this algorithm is that it uses less memory space while executing.
- Pseudo Code-Stepwise Solution:
  1. Selecting a hash function h so each element in the set is mapped to a string to at least $\log_2 n$ bits.
  2. For each element x, r(x)= length of trailing zeroes in h(x)
  3. R= max(r(x))

  => Distinct elements= 2^R

Source Code

```
stream = [1, 2, 3, 4, 5, 6, 4, 2, 5, 9, 1, 6, 3, 7, 1, 2,
2, 4, 2, 1]
print('Using Flajolet Martin Algorithm:')
import time

start_time = time.time()

maxnum = 0
for iin range(0, len(stream)):
val = bin((1 * stream[i] + 6) % 32)[2:]

  sum = 0
for j in range(len(val) - 1, 0, -1):

if val[j] == '0':
      sum += 1
else:
break
```

```
    if sum >maxnum:
maxnum = sum

print('distict elements', 2 ** maxnum)
print("--- %s seconds ---" % (time.time() -
start_time))
```

Output:

```
Using Flajolet Martin Algorithm:
distict elements 8
--- 0.0 seconds ---
```

Experiment No. 8

Social Network Analysis using R (for example: Community Detection Algorithm)

Social Network Analysis (SNA) is the process of exploring or examining the social structure by using graph theory. It is used for measuring and analyzing the structural properties of the network. It helps to measure relationships and flows between groups, organizations, and other connected entities.

Before we start let us see some network analysis terminology

- A network is represented as a graph, which shows links (if any) between each vertex (or node) and its neighbors.
- A line indicating a link between vertices is called an edge.
- A group of vertices that are mutually reachable by following edges on the graph is called a component.
- The edges followed from one vertex to another are called a path.

The following software is required in order to perform network analysis

- R software
- Packages:
  - igraph
  - sna (social network analysis)

Community Detection Algorithm

- A community, with respect to graphs, can be defined as a subset of nodes that are densely connected to each other and loosely connected to the nodes in the other communities in the same graph.
- Detecting communities in a network is one of the most important tasks in network analysis. In a large-scale network, such as an online social network, we could have millions of nodes and edges. Detecting communities in such networks becomes a herculean task.
- Therefore, we need community detection algorithms that can partition the network into multiple communities.

Girvan-Newman Algorithm for Community Detection

- Under the Girvan-Newman algorithm, the communities in a graph are discovered by iteratively removing the edges of the graph, based on the edge betweenness centrality value.
- The edge with the highest edge betweenness is removed first. We will cover this algorithm later in the article, but first, let's understand the concept of "edge betweenness centrality".

Edge Betweenness Centrality (EBC)

The edge betweenness centrality (EBC) can be defined as the number of shortest paths that pass through an edge in a network. Each and every edge is given an EBC score based on the shortest paths among all the nodes in the graph.

Source Code

```
# Social Network Analysis
library(igraph)
g <- graph(c(1,2,2,3,3,4,4,1),
       directed = F,
       n=7)

g1 <- graph(c("Amy", "Ram", "Ram", "Li", "Li", "Amy",
         "Amy", "Li", "Kate", "Li"),
       directed=T)

# Network measures
degree(g1, mode='all')
degree(g1, mode='in')
```

```
degree(g1, mode='out')

diameter(g1, directed=F, weights = NA)
edge_density(g1, loops = F)
ecount(g1)/(vcount(g1)*(vcount(g1)-1))
reciprocity(g1)
closeness(g1, mode='all', weights = NA)
betweenness(g1, directed=T, weights=NA)
edge_betweenness(g1, directed=T, weights=NA)

# Read data file
data <-
read.csv('https://raw.githubusercontent.com/bkrai/R-files-
from-YouTube/main/networkdata.csv', header=T)
y <- data.frame(data$first, data$second)

# Create network
net <- graph.data.frame(y, directed=T)
V(net)$label <- V(net)$name
V(net)$degree <- degree(net)

# Histogram of node degree
hist(V(net)$degree)

# Network diagram
plot(net)

# Highlighting degrees & layouts
plot(net,
vertex.color = rainbow(52),
vertex.size = V(net)$degree*0.4,
edge.arrow.size = 0.1,
    layout=layout.fruchterman.reingold)
# Hub and authorities
hs<- hub_score(net)$vector
as <- authority.score(net)$vector
par(mfrow=c(1,2))
set.seed(123)
plot(net,
vertex.size=hs*30,
    main = 'Hubs',
vertex.color = rainbow(52),
edge.arrow.size=0.1,
    layout = layout.kamada.kawai)

plot(net,
vertex.size=as*30,
    main = 'Authorities',
vertex.color = rainbow(52),
edge.arrow.size=0.1,
    layout = layout.kamada.kawai)
par(mfrow=c(1,1))

# Community detection
net <- graph.data.frame(y, directed = F)
cnet<- cluster_edge_betweenness(net)
plot(cnet)
```

Output

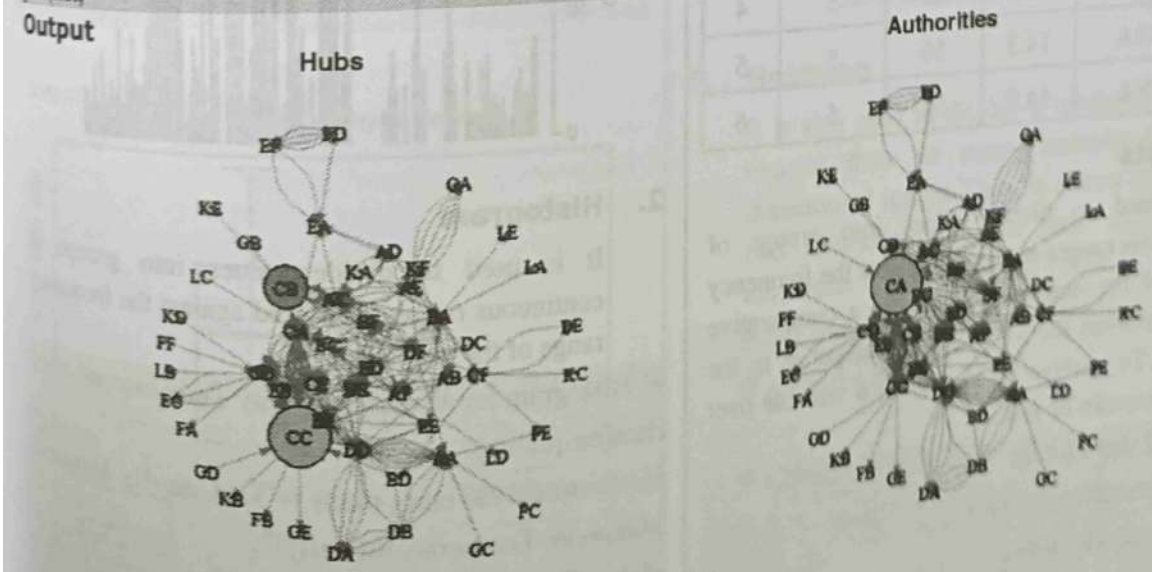


Fig. L1

Experiment No. 9

Data Visualization using Hive/PIG/R/Tableau

Data visualization is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

Data Visualization in R Programming Language

- The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana.
- R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.

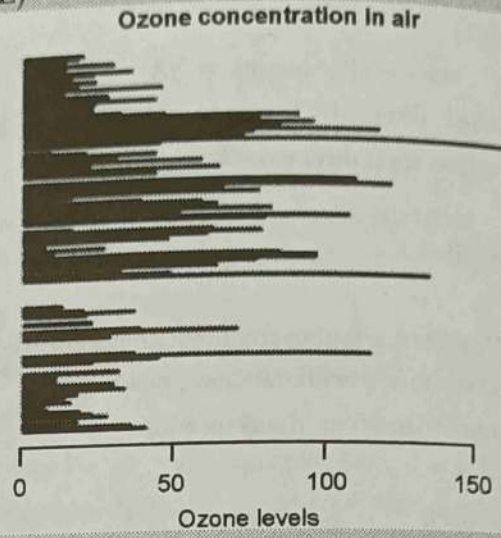
Consider the following airquality data set for visualization in R:

Ozone	Solar R.	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

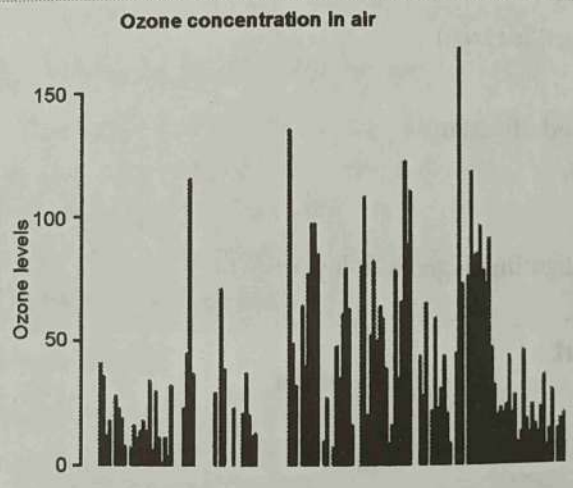
1. Barplots

It is used to divide values into groups of continuous ranges measured against the frequency range of the variable.To perform a comparative study between the various data categories in the data set.To analyze the change of a variable over time in months or years.

```
# Horizontal Bar Plot for
# Ozone concentration in air
barplot(airquality$Ozone,
        main = 'Ozone Concenteration in air',
xlab = 'ozone levels', col = "green", horiz =
TRUE)
```




```
# Vertical Bar Plot for
# Ozone concentration in air
barplot(airquality$Ozone,
        main = 'Ozone Concenteration in air',
ylab = 'ozone levels', col = "blue", horiz = FALSE)
```

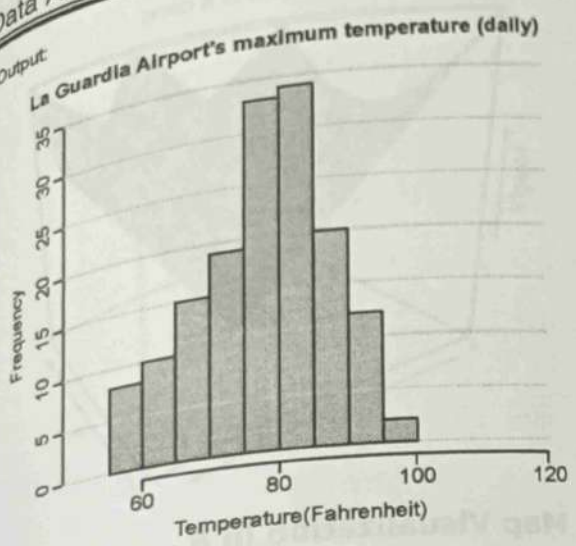


2. Histogram

It is used to divide values into groups of continuous ranges measured against the frequency range of the variable.

```
# Histogram for Maximum Daily Temperature
data(airquality)
hist(airquality$Temp, main ="La Guardia Airport's\
Maximum Temperature(Daily)",
xlab ="Temperature(Fahrenheit)",
xlim = c(50, 125), col ="cyan",
freq = TRUE)
```

Output:

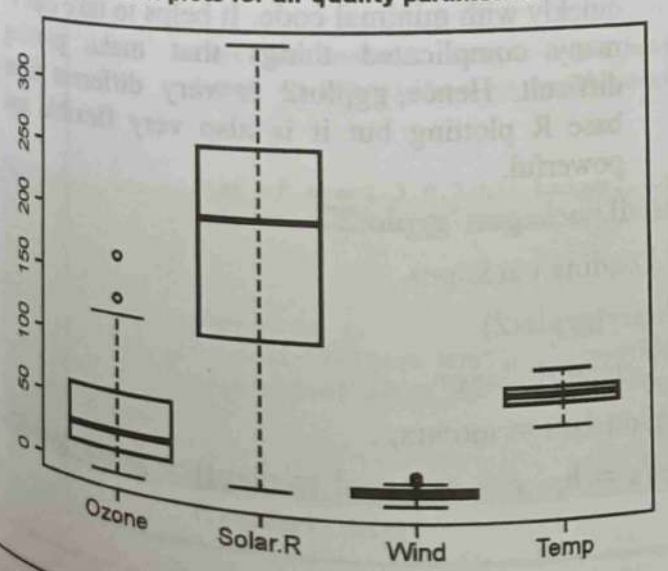


3. Boxplot

- The statistical summary of the given data is presented graphically using a boxplot. A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.
- To give a comprehensive statistical description of the data through a visual cue. To identify the outlier points that do not lie in the inter-quartile range of data.

```
# Multiple Box plots, each representing
# an Air Quality Parameter
boxplot(airquality[, 0:4],
    main ='Box Plots for Air Quality Parameters',
col = "orange")
```

Output:

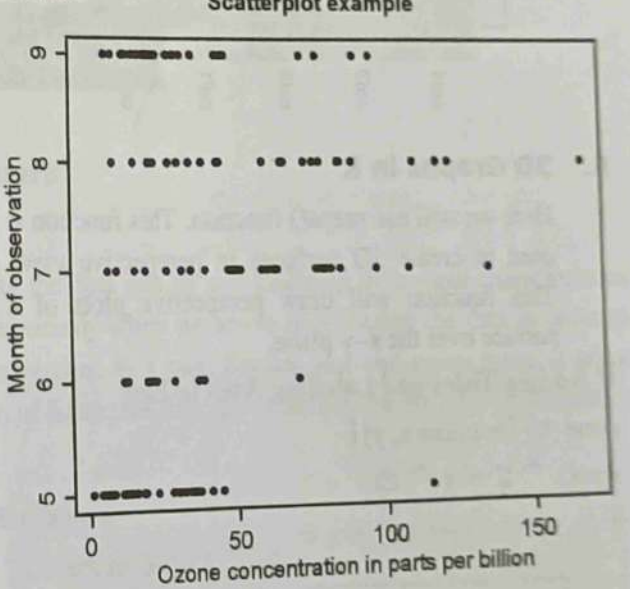


4. Scatterplot

A scatter plot is composed of many points on a Cartesian plane. Each point denotes thevalue taken by two parameters and helps us easily identify the relationship between them.To show whether an association exists between bivariate data.To measure the strength and direction of such a relationship.

```
# Scatter plot for Ozone Concentration per month
data(airquality)

plot(airquality$Ozone, airquality$Month,
    main ="Scatterplot Example",
xlab ="Ozone Concentration in parts per billion",
ylab =" Month of observation ", pch = 19)
```

Output:

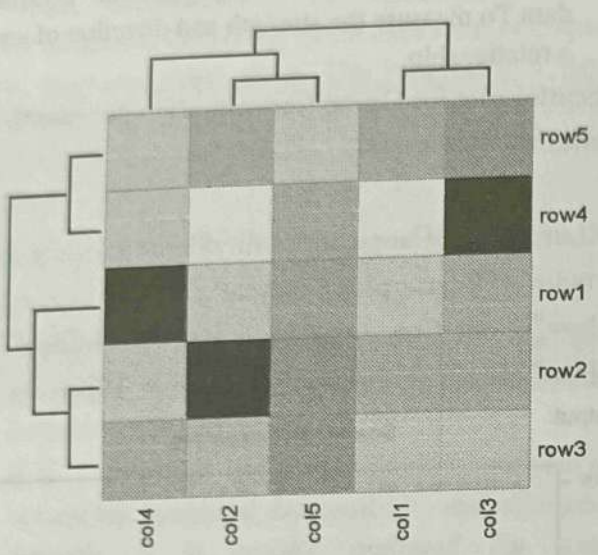


5. Heatmap

It is also used to display a relationship between two or three or many variables in a two-dimensional image. Thus, it allows us to explore two dimensions of the axis and the third dimension by an intensity of colour.

```
# Set seed for reproducibility
# set.seed(110)

# Create example data
data <- matrix(rnorm(50, 0, 5), nrow = 5, ncol
= 5)
```

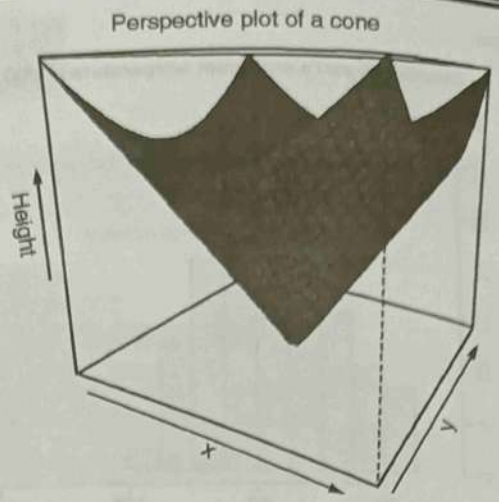
```
# Column names
colnames(data) <- paste0("col", 1:5)
rownames(data) <- paste0("row", 1:5)
# Draw a heatmap
heatmap(data, col = cm.colors(256))
```



6. 3D Graphs in R

Here we will use preps() function, This function is used to create 3D surfaces in perspective view. This function will draw perspective plots of a surface over the x–y plane.

```
# Adding Titles and Labeling Axes to Plot
cone <- function(x, y){
sqrt(x ^ 2 + y ^ 2)
}

# prepare variables.
x <- y <- seq(-1, 1, length = 30)
z <- outer(x, y, cone)

# plot the 3D surface
# Adding Titles and Labeling Axes to Plot
persp(x, y, z,
    main="Perspective Plot of a Cone",
zlab = "Height",
    theta = 30, phi = 15,
    col = "purple", shade = 0.4)
```



7. Map Visualization in R

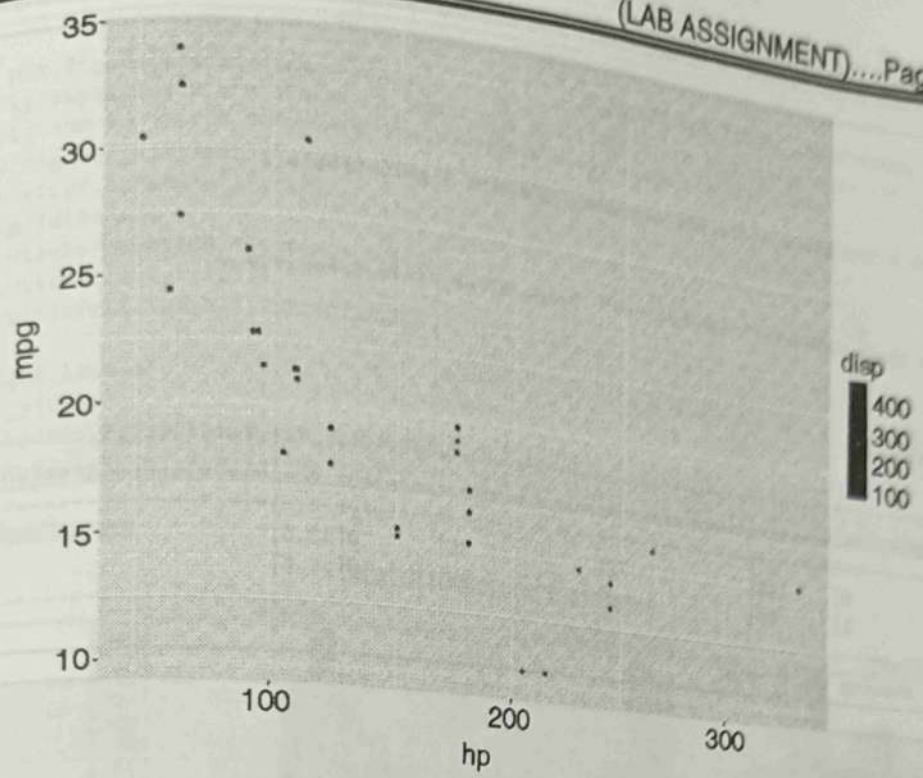
Here we are using maps package to visualize and display geographical maps using an R programming language.

```
install.packages("maps")
# Load the required libraries
library(maps)
map(data = "world")
```

8. ggplot2

It is a plotting system. We use it to build professional-looking graphs. Also, use plots quickly with minimal code. It helps to take care of many complicated things that make plotting difficult. Hence, ggplot2 is very different from base R plotting but it is also very flexible and powerful.

```
install.packages("ggplot2")
# Loading packages
library(ggplot2)
# Geometric layer
ggplot(data = mtcars,
aes(x = hp, y = mpg, col = disp)) + geom_point()
```



Experiment No. 10

Exploratory Data Analysis using Spark/ Pyspark

What is Exploratory Data Analysis ?

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modeling the data in order to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important.

Exploratory Data Analysis with Diabetes Database

The following three cells must be ran in order to use PySpark in Google Colab

Spark 3.0.2

```
In [ ]:
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-3.0.2/spark-3.0.2-bin-hadoop2.7.tgz

In [ ]:
!tar xf spark-3.0.2-bin-hadoop2.7.tgz
!pip install -q findspark

In [ ]:
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.2-bin-hadoop2.7"

In [ ]:
import findspark
findspark.init()
```

```
In [ ]: from pyspark.sql import SparkSession

In [ ]: spark = SparkSession.builder.appName("diabetes").getOrCreate()

In [ ]: df = spark.read.csv("diabetes.csv", header=True, inferSchema=True)

In [ ]: df.show(2)
```

```
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin| BMI|DiabetesPedigreeFunction|Age|Outcome|
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
|          6|    148|           72|           35|      0|33.6|                   0.627| 50|      1|
|          1|     85|           66|           29|      0|26.6|                   0.351| 31|      0|
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
only showing top 2 rows
```

```
In [ ]: df.printSchema()
```

```
root
 |-- Pregnancies: integer (nullable = true)
 |-- Glucose: integer (nullable = true)
 |-- BloodPressure: integer (nullable = true)
 |-- SkinThickness: integer (nullable = true)
 |-- Insulin: integer (nullable = true)
 |-- BMI: double (nullable = true)
 |-- DiabetesPedigreeFunction: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Outcome: integer (nullable = true)
```

```
In [ ]: df.describe().toPandas()
```

Out[ ]:

	summary	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
0	count	768	768	768	768	768	
1	mean	3.8450520833333335	120.89453125	69.10546875	20.536458333333332	79.79947916666667	31.992578124999
2	stddev	3.369578062698870	31.97261819513622	19.355807170644777	15.952217567727642	115.24400235133803	7.884160320375
3	min	0	0	0	0	0	
4	max	17	199	122	99	846	

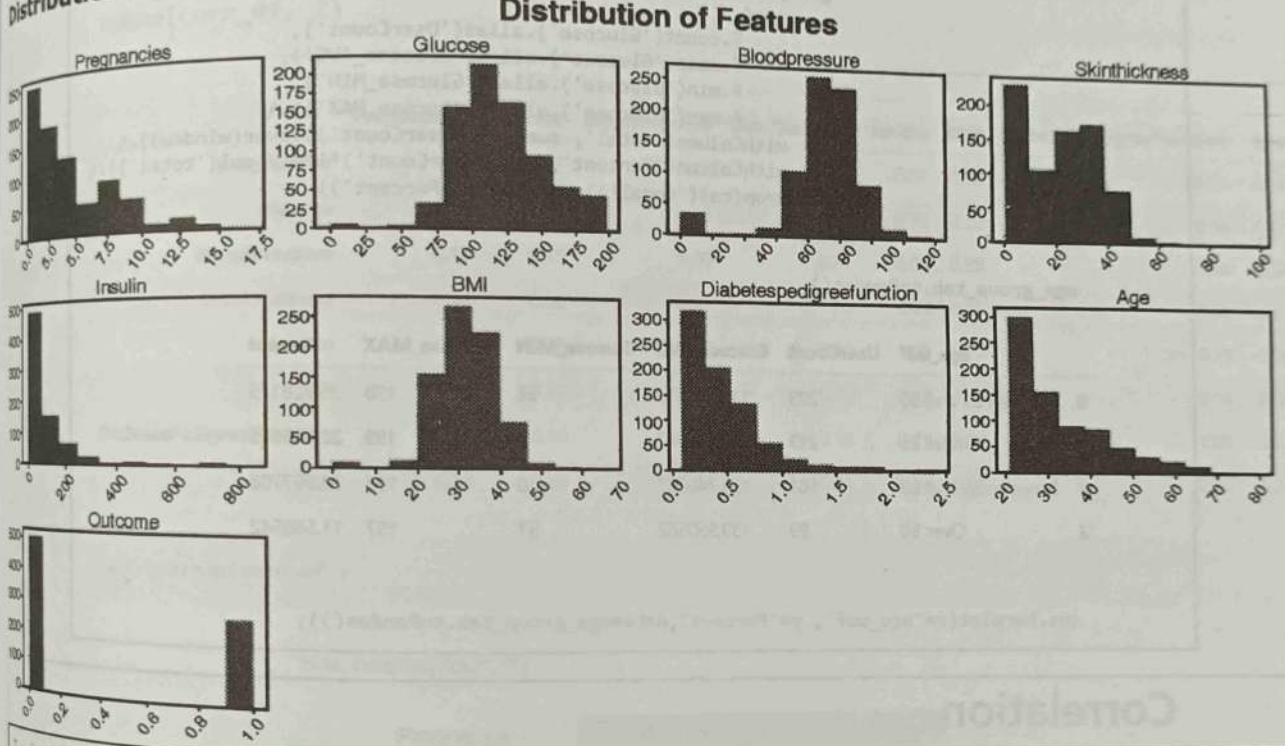
```
In [ ]: df.groupby('Outcome').count().show()
```

```
+-------+-----+
|Outcome|count|
+-------+-----+
|      1|  268|
|      0|  500|
+-------+-----+
```

```
In [ ]: import matplotlib.pyplot as plt
        import seaborn as sns
```

```
fig = plt.figure(figsize=(25, 15))
st = fig.suptitle("Distribution of Features", fontsize=50, verticalalignment="center")
for col, num in zip(df.toPandas().describe().columns, range(1,11)):
  ax = fig.add_subplot(3,4, num)
  ax.hist(df.toPandas()[col])
  plt.grid(False)
  plt.xticks(rotation=45, fontsize=20)
  plt.yticks(fontsize=15)
  plt.title(col.upper(), fontsize=20)

plt.tight_layout()
st.set_y(0.95)
fig.subplots_adjust(top=0.85, hspace=0.4)
plt.show()
```

Distribution of features




```
from pyspark.sql.functions import isnan, when, count, col

df.select([count(when(isnan(c),c)).alias(c) for c in df.columns]).toPandas().head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0	0	0	0	0	0	0	0	0

User Defined Functions (UDF)

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

y_udf = udf(lambda y: "no" if y == 0 else "yes", StringType())
```

```
In [ ]: df = df.withColumn("HasDiabetes",y_udf('Outcome')).drop("Outcome")

In [ ]: df.show(3)
```

```
+-----------+-------+-------------+-------------+-------+----+------------------------+---+----------+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin| BMI|DiabetesPedigreeFunction|Age|HasDiabetes|
+-----------+-------+-------------+-------------+-------+----+------------------------+---+----------+
|          6|    148|           72|           35|      0|33.6|                   0.627| 50|       yes|
|          1|     85|           66|           29|      0|26.6|                   0.351| 31|        no|
|          8|    183|           64|            0|      0|23.3|                   0.672| 32|       yes|
+-----------+-------+-------------+-------------+-------+----+------------------------+---+----------+
only showing top 3 rows
```

```
In [ ]: age_group_tab = df.select(["age_udf","Glucose"]).\
                    groupBy('age_udf').\
                        agg(
                            F.count('Glucose').alias('UserCount'),
                            F.mean('Glucose').alias('Glucose_AVG'),
                            F.min('Glucose').alias('Glucose_MIN'),
                            F.max('Glucose').alias('Glucose_MAX')).\
                        withColumn('total', sum(col('UserCount')).over(window)).\
                        withColumn('Percent', col('UserCount')*100 / col('total')).\
                        drop(col('total')).sort(desc('Percent'))

In [ ]: age_group_tab.toPandas()
```

Out[ ]:

	age_udf	UserCount	Glucose_AVG	Glucose_MIN	Glucose_MAX	Percent
0	Between 25 and 35	279	119.677419	68	198	36.328125
1	Under 25	219	110.858447	0	199	28.515625
2	Between 36 and 50	181	125.740331	0	197	23.567708
3	Over 50	89	139.550562	57	197	11.588542

```
In [ ]: sns.barplot(x="age_udf", y="Percent",data=age_group_tab.toPandas());
```

Correlation

```
In [ ]: numeric_features = [t[0] for t in df.dtypes if t[1] !='string']
        numeric_features_df = df.select(numeric_features)
        numeric_features_df.toPandas().head()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

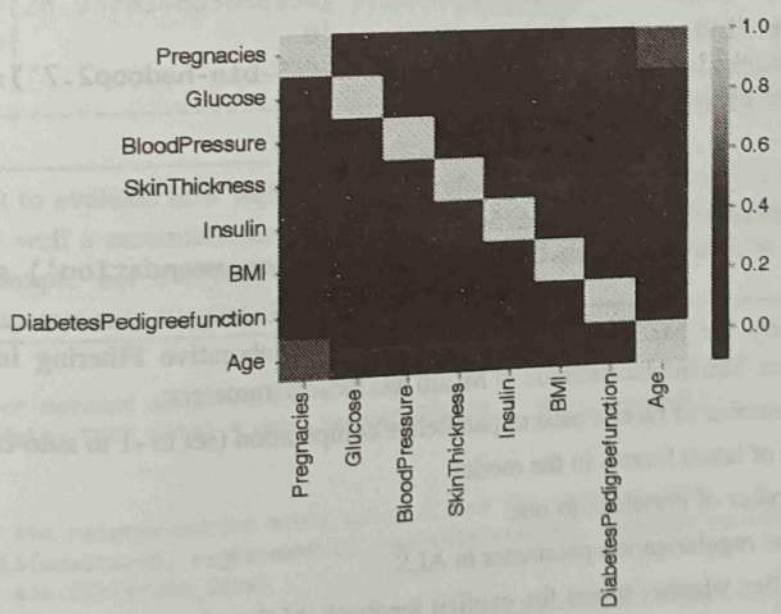
```
In [ ]: col_names = numeric_features_df.columns
        features = numeric_features_df.rdd.map(lambda row: row[0:])

In [ ]: from pyspark.mllib.stat import Statistics
        import pandas as pd

In [ ]: corr_mat = Statistics.corr(features, method="pearson")
        corr_df = pd.DataFrame(corr_mat)
        corr_df.index = col_names
        corr_df.columns = col_names
        round(corr_df, 2)
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Pregnancies	1.00	0.13	0.14	-0.08	-0.07	0.02	-0.03	0.54
Glucose	0.13	1.00	0.15	0.06	0.33	0.22	0.14	0.26
BloodPressure	0.14	0.15	1.00	0.21	0.09	0.28	0.04	0.24
SkinThickness	-0.08	0.06	0.21	1.00	0.44	0.39	0.18	-0.11
Insulin	-0.07	0.33	0.09	0.44	1.00	0.20	0.19	-0.04
BMI	0.02	0.22	0.28	0.39	0.20	1.00	0.14	0.04
DiabetesPedigreeFunction	-0.03	0.14	0.04	0.18	0.19	0.14	1.00	0.03
Age	0.54	0.26	0.24	-0.11	-0.04	0.04	0.03	1.00

```
In [ ]: sns.heatmap(corr_df);
```

Sns. heatmap(corr_df);



Experiment No. 11

Mini Project: One real life large data application to be implemented (Use standard Datasets available on the web).

- Streaming data analysis – use flume for data capture, HIVE/PYSpark for analysis of
- twitter data, chat data, weblog analysis etc.
- Recommendation System (for example: Health Care System, Stock Market
- Prediction, Movie Recommendation, etc.)
- SpatioTemporalDataAnalytics

Movie Recommender Systems using Spark with PySpark

This project is about building a movie recommendation system using explicit feedback of movie ratings by user's aka Alternating Least Squares (ALS) - Collaborative Filtering

Two main approaches for recommendation system

- **Collaborative Filtering:** we make predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption is that if a user A has the same opinion as a user B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a user chosen randomly.
- **Content Based Filtering** methods are based on a description of the item and a profile of the user's preferences. In a content-based recommender system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.

- In this project we will be focusing on a type of Collaborative Filtering called Alternating Least Sqaures (ALS)
- **ALS:** ALS is one of the low rank matrix approximation algorithms for collaborative filtering. ALS decomposes user-item matrix into two low rank matrixes: user matrix and item matrix. In collaborative filtering, users and products are described by a small set of latent factors that can be used to predict missing entries. And ALS algorithm learns these latent factors by matrix factorization.

```
In [1]:
import findspark
findspark.init('/home/sonal/spark-2.4.5-bin-hadoop2.7')
import pyspark

In [2]:
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('recommendation').getOrCreate()
```

Spark MLlib library for Machine Learning provides a Collaborative Filtering implementation by using Alternating Least Squares. The implementation in MLlib has these parameters:

- numBlocks is the number of blocks used to parallelize computation (set to -1 to auto-configure).
- rank is the number of latent factors in the model.
- iterations is the number of iterations to run.
- lambda specifies the regularization parameter in ALS.
- implicitPrefs specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data.

- alpha is a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations.

```
In [3]: from pyspark.ml.evaluation import RegressionEvaluator
        from pyspark.ml.recommendation import ALS

In [4]: data = spark.read.csv('movielens_ratings.csv',inferSchema=True,header=True)

In [5]: data.head()

Out[5]: Row(movieId=2, rating=3.0, userId=0)

In [6]: data.printSchema()

root
 |-- movieId: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- userId: integer (nullable = true)

In [7]: data.describe().show()

+-------+------------------+------------------+------------------+
|summary|           movieId|            rating|            userId|
+-------+------------------+------------------+------------------+
|  count|              1501|              1501|              1501|
|   mean| 49.40572951365756|1.7741505662891406|14.383744170552964|
| stddev|28.937034065088994| 1.187276166124803| 8.591040424293272|
|    min|                 0|               1.0|                 0|
|    max|                99|               5.0|                29|
+-------+------------------+------------------+------------------+
```

- We can do a split to evaluate how well our model performed, but keep in mind that it is very hard to know conclusively how well a recommender system is truly working for some topics. Especially if subjectivity is involved, for example not everyone that loves Star Wars is going to love star trek, even though a recommendation system may suggest otherwise.

```
In [8]: # Smaller dataset so we will use 0.8 / 0.2
        (train_data, test_data) = data.randomSplit([0.8, 0.2], seed=42)

In [9]: # Build the recommendation model using ALS on the training data
        als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating")
        model = als.fit(train_data)
```

```
In [10]: # Evaluate the model by computing the RMSE on the test data
         predictions = model.transform(test_data)
```

```
In [11]: predictions.show()
```

```
+-------+------+------+----------+
|movieId|rating|userId|prediction|
+-------+------+------+----------+
|     31|   1.0|    26| -0.8328631|
|     31|   1.0|     5| 0.91361725|
|     31|   1.0|     4|  1.8772194|
|     31|   2.0|    25| -1.0943574|
|     31|   1.0|    18| -0.2835476|
|     85|   3.0|     1|  3.0139313|
|     85|   1.0|    13|  2.2822132|
|     85|   3.0|     6| -1.7273896|
|     85|   1.0|    25| -1.3156357|
|     65|   1.0|    16|-0.07137105|
|     65|   1.0|     2|  2.3545816|
|     78|   1.0|     1|  1.1508821|
|     78|   1.0|    19| 0.69761777|
|     78|   1.0|    24|   1.624121|
|     78|   1.0|     2|  1.4557397|
|     34|   1.0|    28|  2.2461605|
|     34|   1.0|    16|-0.91016656|
|     34|   1.0|    19| 0.17841205|
|     81|   1.0|     6|   2.857556|
|     81|   2.0|     5|-0.10178676|
+-------+------+------+----------+
only showing top 20 rows
```

```
In [12]: evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",predictionCol="prediction")
         rmse = evaluator.evaluate(predictions)
         print("Root-mean-square error = " + str(rmse))
```

```
Root-mean-square error = 1.874538232312848
```

- The RMSE described our error in terms of the stars rating column.
- So now that we have the model, how would you actually supply a recommendation to a user?
- The same way we did with the test data! For example:

```
In [22]: single_user = test_data.filter(test_data['userId']==12).select(['movieId','userId'])
```

```
In [23]: # User had 10 ratings in the test data set
         # Realistically this should be some sort of hold out set!
         single_user.show()
```

```
+-------+------+
|movieId|userId|
+-------+------+
|     18|    12|
|     24|    12|
|     25|    12|
|     30|    12|
|     38|    12|
|     45|    12|
|     50|    12|
|     54|    12|
|     57|    12|
|     79|    12|
|     83|    12|
```

```
[25]: reccomendations.orderBy('prediction',ascending=False).show()

+-------+------+----------+
|movieId|userId|prediction|
+-------+------+----------+
|     50|    12|  4.368422|
|     79|    12| 3.8063822|
|     18|    12| 3.4928117|
|     30|    12|  3.491328|
|     25|    12| 3.3623793|
|     54|    12| 1.2524549|
|     24|    12|0.97046584|
|     38|    12| 0.7397181|
|     57|    12|0.47302982|
|     45|    12|0.08336234|
|     83|    12| -0.785038|
+-------+------+----------+
```

- We can recommend movie with id 50 to user. The user might like those based on previous history. However, we say don't watch movies with id 86,66 and 81. You might don't like it.
- What if a user has never watched any movie or a new user, what we can recommend, it's called a cold start in the recommendation system. Well, in that case, we can ask the user to take a survey and get an idea of his interest in movies. Or we can give other users recommendations. Cold start is a problem for the recommendation system problem in general.

Chapter Ends...
❑❑❑

Multiple Choice Questions

Chapter 1 : Introduction to Big Data and Hadoop

Q. 1 The important 3V's in big data are ?
(a) Volume, Vulnerability, Variety.
(b) Volume, Velocity, Variety.
(c) Velocity, Vulnerability, Variety.
(d) Volume, Vulnerability, Velocity.
✓**Ans. : (b)**

Q. 2 What are the different features of Big data?
(a) Open source. (b) Scalability.
(c) Data recovery. (d) All of the above.
✓**Ans. : (d)**

Q. 3 Which one is false about big data analytics?
(a) It collects data.
(b) It looks for pattern.
(c) It does not organize data.
(d) It analyzes data. ✓**Ans. : (d)**

Q. 4 Which of the following would NOT use big data analytics?
(a) Healthcare companies.
(b) Public agencies.
(c) Retail companies.
(d) High school students doing homework. ✓**Ans. : (d)**

Q. 5 What are the main components of big data?
(a) YARN (b) HDFS
(c) MapReduce (d) All of the above
✓**Ans. : (d)**

Q. 6 The data node and name node in HADOOP are
(a) Worker Node and Master Node respectively
(b) Master Node and Worker Node respectively
(c) Both Worker Nodes
(d) Both Master Nodes ✓**Ans. : (a)**

Q. 7 Message generated by a data node to indicate its connectivity with NameNode?
(a) Beep (b) Heartbeat
(c) Analog pulse (d) Map
✓**Ans. : (b)**

Q. 8 The total forms of big data is ____
(a) 1 (b) 2 (c) 3 (d) 4
✓**Ans. : (c)**

Q. 9 In which language is Hadoop written?
(a) C++ (b) Java
(c) Rust (d) Python ✓**Ans. : (b)**

Q. 10 Which of the following is not a part of the data science process.
(a) Communication building
(b) Discovery
(c) Operationalize
(d) Model planning ✓**Ans. : (a)**

Chapter 2 : Hadoop HDFS and MapReduce

Q. 11 Mapper class is
(a) Final
(b) Static type
(c) Generic type
(d) Abstract type ✓**Ans. : (c)**

Q. 12 Fixed-size pieces of MapReduce job is known as _____
(a) Splits (b) Tasks
(c) Maps (d) Records ✓**Ans. : (a)**

Q. 13 The location of files in MapReduce controlled by:
(a) Hadoop file manager
(b) MapReduce files list
(c) Hadoop configuration files
(d) Hadoop cluster
✓**Ans. : (c)**

Q. 14 The log directory also has a subdirectory called:
(a) Log users
(b) Directory users
(c) Userlogs
(d) Useful logs
✓**Ans. : (c)**

Q. 15 Input to the _____ is the sorted output of the mappers.
(a) Reducer
(b) Mapper
(c) Shuffle
(d) All of the mentioned ✓**Ans. : (a)**

Q. 16 A _____ node acts as the Slave and is responsible for executing a Task assigned to it by the JobTracker.
(a) MapReduce
(b) Mapper
(c) TaskTracker
(d) JobTracker ✓**Ans. : (c)**

Q. 17 _____ function is responsible for consolidating the results produced by each of the Map() functions/tasks.
(a) Reduce
(b) Map
(c) Reducer
(d) All of the mentioned ✓**Ans. : (a)**

Q. 18 _____ is a utility which allows users to create and run jobs with any executables as the mapper and/or the reducer.
(a) Hadoop Strdata
(b) Hadoop Streaming
(c) Hadoop Stream
(d) None of the mentioned
✓**Ans. : (b)**

Q. 19 Which of the following terms is used to denote the small subsets of a large file created by HDFS?
(a) NameNode
(b) DataNode
(c) Blocks
(d) Namespace
✓**Ans. : (c)**

Q. 20 Running a _____ program involves running mapping tasks on many or all of the nodes in our cluster.
(a) MapReduce
(b) Map
(c) Reducer
(d) All of the mentioned ✓**Ans. : (a)**

Chapter 3 : NoSQL

Q. 21 Which of the following is a disadvantage of relational database?
(a) Concurrency
(b) Impedance mismatch
(c) ACID transactions
(d) Normalization ✓**Ans. : (b)**

Q. 22 Which of the following is a disadvantage of NoSQL database?
(a) Scalability
(b) Performance
(c) High availability
(d) Complex transaction support
✓**Ans. : (d)**

Q. 23 Which of the following is not a NoSQL data model?
(a) object relational
(b) column-oriented store
(c) document-oriented store
(d) graph database ✓**Ans. : (a)**

Q. 24 Which type of format cannot be stored or retrieved in the document data model?
(a) JSON (b) XML
(c) DOC (d) BSON ✓Ans. : (c)

Q. 25 Which of the following data models can be used for social network mining?
(a) document data model
(b) schema-less data model
(c) key/value data model
(d) graph data model
✓Ans. : (d)

Q. 26 Sharding is type of _____ partitioning.
(a) vertical
(b) horizontal
(c) both vertical and vertical
(d) none of these ✓Ans. : (b)

Q. 27 Which of the following nodes does MapReduce use to distribute the tasks to worker nodes?
(a) job tracker
(b) pivot table
(c) metadata
(d) root node ✓Ans. : (a)

Q. 28 All NoSQL databases are similar.
(a) True (b) False ✓Ans. : (b)

Q. 29 Can NoSQL replace SQL?
(a) Yes (b) No ✓Ans. : (b)

Q. 30 Which of the following is a characteristic of a NoSQL database?
(a) Uses JSON
(b) Requires JOINs
(c) Needs a schema
(d) Uses tables for storage
✓Ans. : (a)

Chapter 4 : Mining Data Streams

Q. 31 In Flajolet-Martin algorithm if the stream contains n elements with m of them unique, this algorithm runs in
(a) O(n) (b) O(1)
(c) O(2n) (d) O(3n)
✓Ans. : (a)

Q. 32 Real-time data stream is _____
(a) sequence of data items that arrive in some order and may be seen only once
(b) sequence of data items that arrive in some order and may be seen twice.
(c) sequence of data items that arrive in same order
(d) sequence of data items that arrive in different order ✓Ans. : (a)

Q. 33 In sliding window of size w an element arriving at time t expires at
(a) w (b) t
(c) t+w (d) t-w ✓Ans. : (c)

Q. 34 What are DGIM's maximum error boundaries?
(a) DGIM either underestimates or overestimates the true count; at most by 25%
(b) DGIM always underestimates the true count; at most by 25%
(c) DGIM either underestimates or overestimates the true count; at most by 50%
(d) DGIM always overestimates the count; at most by 50%
✓Ans. : (b)

Q. 35 In FM algorithm we shall use estimate.....for the number of distinct elements seen in the stream
(a) 2 to the power R
(b) 3 to the power R
(c) 2R
(d) 3R ✓Ans. : (a)

Q. 36 Which algorithm we will implement to know how many distinct users visited the website till now or in last 2 hours.

(a) DGIM (b) Clustering
(c) SVM (d) FM ✓**Ans. : (d)**

Q. 37 A Bloom filter consists of_____

(a) An array of n bits, initially all 0's.
(b) An array of 2 bits, initially all 0's
(c) An array of 1 bits, initially all 0's.
(d) An array of n bits, initially all 1's.

✓**Ans. : (a)**

Q. 38 The purpose of the Bloom filter is to allow_____

(a) through all stream elements whose keys are in Set
(b) through all stream elements whose keys are in class
(c) through all data elements whose keys are in Set
(d) through all touple elements whose keys are in Set ✓**Ans. : (a)**

Q. 39 Which of the following statements about the standard DGIM algorithm are false?

(a) DGIM operates on a time-based window.
(b) In DGIM, the size of a bucket is always a power of two.
(c) The maximum number of buckets has to be chosen beforehand
(d) DGIM reduces memory consumption through a clever way of storing counts ✓**Ans. : (c)**

Q. 40 In DGIM, whenever forming a bucket then_____

(a) Every bucket should have at least one 1, else no bucket can be formed
(b) Every bucket should have at least two 1, else no bucket can be formed
(c) Every bucket should have at least three 1, else no bucket can be formed
(d) Every bucket should have at least four 1, else no bucket can be formed ✓**Ans. : (a)**

Chapter 5 : Real- Time Big data Models

Q. 41 User-Based Collaborative Filtering methods -

(a) Are based on user's similarity only
(b) In such methods complexity grows linearly with the number of customers and items
(c) Suffers the problem of sparsity of recommendations on the data set
(d) a,b and c ✓**Ans. : (d)**

Q. 42 Identify the correct recommendation system's algorithm(s) fromgiven options.

(a) Collaborative page ranking
(b) Collaborative filtering
(c) Item based recommendation system
(d) Object based recommendation system ✓**Ans. : (b)**

Q. 43 Recommender systems can be defined as

(a) Systems that evaluate quality based on the preferences of others with a similar point of view
(b) Systems that evaluate quality based on the purchase history of any particular person only
(c) Systems that evaluate quality based on the demand of items
(d) Systems that evaluate quality based on the association rule mining techniques ✓**Ans. : (a)**

Q. 44 What is the goal of collaborative filtering?

(a) Collaborating with other websites
(b) Filtering your audience based on demographics
(c) Determining what new customers do in their spare time

(d) Delivering recommended products or services ✓Ans. : (d)

Q. 45 Identify the correct statements related to Collaborative Filtering

(a) Predict the opinion the user will have on the different items

(b) Recommend the 'best' items based on the user's previous likings and the opinions of like-minded users whose ratings are similar

(c) The problem of collaborative filtering is to predict how well a user will like an item that he has not rated given a set of historical preference judgments for a community of users.

(d) All of the above options

✓Ans. : (d)

Q. 46 The search or content based methods

(a) Given the user's purchased and rated items, constructs a search query to find other popular items

(b) It may use, same author, artist, director, or similar keywords/subjects

(c) It is impractical to base a query on all the items in content based method

(d) All of the above options

✓Ans. : (d)

Q. 47 Item-to-Item Collaborative Filtering methods -

(A) It is less scalable than User-Based Collaborative Filtering methods

(B) Amazon.com uses this method

(C) Scales independently of the catalog size or the total number of customers

(D) Acceptable performance by creating the expensive similar-itemtable of fline

(a) Only A (b) A and B

(c) B,C and D (d) A,B,C and D

✓Ans. : (c)

Q. 48 Identify all challenges of collaborative filtering

(A) Modularity

(B) Data sparsity

(C) Scalability

(D) Shilling attacks

(a) Only A (b) A and B

(c) B,C and D (d) A,B,C and D

✓Ans. : (c)

Q. 49 The memory based collaborative filtering uses

(A) It generally uses Naive bayesian system.

(B) This mechanism uses user rating data to compute similarity between users or items.

(C) Typical examples of this mechanism are neighborhood based CF and item-based/user-based top-N recommendations.

(a) Only A (b) B and C

(c) A and D (d) A,B and C

✓Ans. : (b)

Q. 50 Identify the correct statements related to recommender system

(a) An information filtering technology, commonly used on ecommerce Web sites that uses a collaborative filtering to present information on items and products that are likely to be of interest to the reader.

(b) In presenting the recommendations, the recommender system will use details of the registered user's profile and opinions and habits of their whole community of users and compare the information to reference characteristics to present the recommendations.

(c) An example of a recommender system isWhatShouldIReadNext.com, a site where users can enter a title of a recent book they have read and enjoyed to see recommended books that they are likely to also enjoy.

(d) All of the above options

✓Ans. : (d)

Chapter 6 : Data Analytics with R

Q. 51 How many types of R objects are present in R data type?

(a) 4 (b) 5 (c) 6 (d) 7

✓Ans. : (c)

Q. 52 R is an_____programming language?

(a) Closed source

(b) GPL

(c) Open source

(d) Definite source ✓Ans. : (c)

Q. 53 Which of the following plots is circular

(a) index (b) histogram

(c) bar (d) pie ✓Ans. : (d)

Q. 54 R is technically much closer to the Scheme language than it is to the original _____ language.

(a) B (b) S (c) C (d) C++

✓Ans. : (b)

Q. 55 unlike writing out a table or CSV file, dump() and dput() preserve the _____ so that another user doesn't have to specify the all over again.

(a) metadata

(b) backup data

(c) attribute data

(d) normal data

✓Ans. : (a)

Q. 56 _____ generate summary statistics of different variables in the data frame, possibly within strata.

(a) rename (b) summarize

(c) set (d) subset

✓Ans. : (b)

Q. 57 _____ splits a data frame and results an array (hence the da). Hopefully you're getting the idea here.

(a) apply (b) daply

(c) stats (d) plyr ✓Ans. : (b)

Q. 58 Which of the following lists names of variables in a data.frame?

(a) quantile() (b) names()

(c) barchart() (d) par()

✓Ans. : (a)

Q. 59 Which function is used for seeing currently active libraries?

(a) Curlib() (b) Currlib()

(c) .libpaths() (d) Pathlibr()

✓Ans. : (c)

Q. 60 Which function is used for loading packages?

(a) Library (b) Interface

(c) Loader (d) Linker

✓Ans. : (a)