SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)
CSE (DATA SCIENCE)

LECTURE NOTES

Subject Name: BIG DATA ANALYTICS

Year / Branch: IV B.TECH – VII SEMESTER

Regulation: R23

Prepared By: Mr. G.YUVARAJU

Assistant Professor

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(Autonomous)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

IV B. TECH.-VII SEMESTER

23CSD471T	BIG DATA ANALYTICS	L T P C
		3 - - 3

PRE-REQUISITES: Data Engineering

COURSE EDUCATIONAL OBJECTIVES:

1. Provide an overview of an exciting growing field of big data analytics.
2. Introduce the tools required to manage and analyze big data like Hadoop, NoSQL, MapReduce, HIVE, Cassandra, Spark.
3. Teach the fundamental techniques and principles in achieving big data analytics with scalability and streaming capability.
4. Optimize business decisions and create competitive advantage with Big Data analytics

UNIT–1: INTRRODUCTION TO BIG DATA (9)

Introduction to Big Data platform- What is Big Data? Big Data Sources-Acquisition-Nuts and Bolts of Big data-Features of Big Data-Security - Compliance - auditing and protection-Evolution of Big Data- Best practices for Big Data Analytics-Big Data characteristics- Volume - Veracity - Velocity - Variety- Structure of Big Data- Exploring the opportunities with Big Data.

UNIT–2: APACHE HADOOP (9)

Introduction to Hadoop- A brief history of Hadoop - Apache Hadoop and The Hadoop Ecosystem AND Hadoop Features - fault tolerance , with data replication, Distributed, parallel processing , High availability, Data locality - The Design of HDFS-HDFS concepts.

UNIT–3: INTRODUCTION TO HIVE AND PIG (9)

Introduction to Hive, data types and file formats, HiveQL data definition- Table creation-Internal table -External table, HiveQL data manipulation, Logical joins, Window functions-rank-dense-rank- row number- Table partitioning-static partitioning -dynamic partitioning - Bucketing.

Pig- Pig Latin-Statements-Expressions-Types- Pig Architecture-Data processing Operators-Loading and storing Data-Filtering Data-Grouping and Joining Data-Sorting Data-Combining and Splitting Data

UNIT–4: HBASE- ZOOKEEPER – SQOOP (9)

HBasics – Concepts – Example-HBase Versus RDBMS HBase, Hbase architecture, data model, operations—Introduction to Zookeeper-Zookeeper Services-Building applications with Zookeeper - Introduction to Sqoop-Database Imports-Working with Imported data- Importing large objects-performing exports.

UNIT–5: APACHE SPARK (9)

Apache spark- Advantages over Hadoop, lazy evaluation, In memory processing, DAG, Spark context, Spark Session, RDD, Transformations- Narrow and Wide, Actions, Data frames ,RDD to Data frames, Catalyst optimizer, Data Frame Transformations, Working with Dates and Timestamps, Working with Nulls in Data, Working with Complex Types, Working with JSON, Grouping, Window Functions, Joins, Data Sources, Broadcast Variables, Accumulators.

Total Hours:45

UNIT-1: INTRRODUCTION TO BIG DATA

Unit Overview

This unit introduces the fundamentals of Big Data, its concepts, terminology, and significance in today's data-driven world. It covers the Big Data platform, data sources, data acquisition methods, and the fundamental building blocks of Big Data. The unit explains the key features and characteristics of Big Data, including the 5Vs—Volume, Velocity, Variety, Veracity, and Value. It also discusses data security, compliance, auditing, and protection, along with the evolution of Big Data technologies and best practices for Big Data Analytics. Finally, the unit explores the structure of Big Data and its applications across various domains.

Learning Outcomes

After completing this unit, students will be able to:

1. Define Big Data and explain its importance in modern computing.
2. Describe Big Data platforms, data sources, and data acquisition techniques.
3. Explain the fundamental building blocks and features of Big Data.
4. Understand the evolution of Big Data technologies and analytics.
5. Describe the characteristics of Big Data, including Volume, Velocity, Variety, Veracity, and Value.
6. Explain the concepts of data security, compliance, auditing, and data protection.
7. Analyze the structure of Big Data and identify opportunities for applying Big Data Analytics in real-world scenarios.

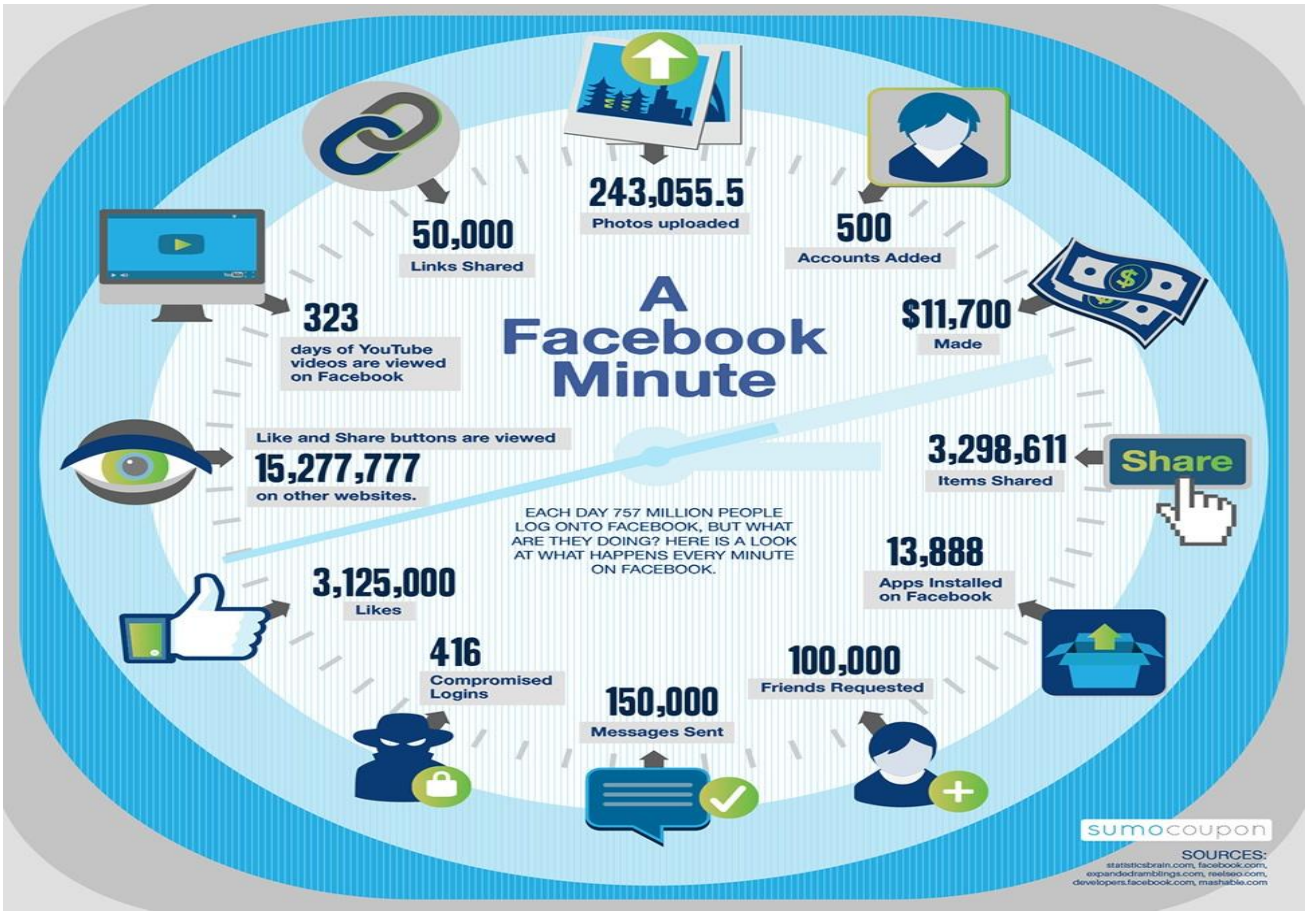
Importance of Studying this Unit

The rapid growth of digital technologies has resulted in an unprecedented increase in data generation from social media, IoT devices, healthcare, finance, education, e-commerce, and many other domains. Organizations rely on Big Data technologies to store, process, and analyze massive datasets for informed decision-making. Studying this unit enables students to understand the foundations of Big Data, its infrastructure, security challenges, and analytical capabilities. The knowledge gained forms the basis for advanced topics such as Hadoop, Spark, Machine Learning, Artificial Intelligence, Cloud Computing, and Business Intelligence, making it essential for careers in Data Science and Data Engineering.

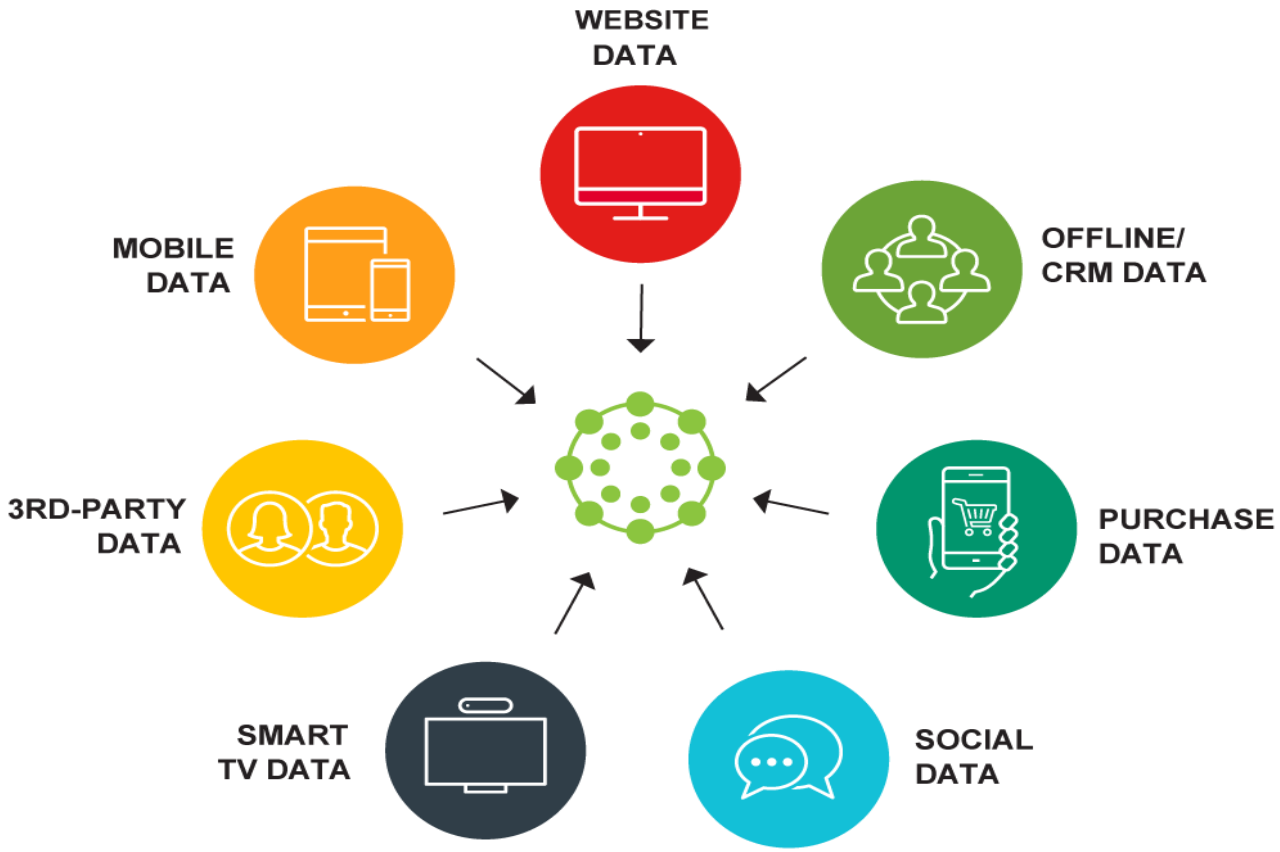
What is Big Data

Big Data refers to extremely large and complex datasets that cannot be processed efficiently using traditional data processing tools and techniques. (or) Data that satisfies 5 V's is called Big Data

What makes data as "Big" Data?



Big Data Sources



Big Data Platform

A Big Data Platform is an integrated ecosystem of hardware, software, networking, storage, and analytical tools designed to collect, store, process, manage, analyze, and visualize massive volumes of structured, semi-structured, and unstructured data. It enables organizations to handle data that is too large, complex, or fast-changing for traditional database systems.

A Big Data platform supports distributed storage, parallel processing, real-time analytics, and scalable computing, making it suitable for applications such as social media analytics, fraud detection, healthcare, e-commerce, scientific research, and IoT.

Objectives of a Big Data Platform

The primary objectives of a Big Data platform are to:

- Collect data from multiple sources.
- Store massive datasets efficiently.
- Process data at high speed.
- Analyze data to extract meaningful insights.
- Present results through reports and dashboards.
- Support informed decision-making

Components of a Big Data Platform

1. Data Sources

Data sources are the origins from which data is generated.

Examples

- Social media (Facebook, Twitter, Instagram)
- IoT sensors
- Mobile applications
- Websites
- Banking systems

- Healthcare systems
- E-commerce platforms
- Enterprise applications (ERP, CRM)

Importance

- Provides raw data for analysis.
- Supports business intelligence.
- Enables predictive analytics.

2. Data Acquisition

Data acquisition refers to collecting data from various sources and transferring it into the Big Data platform.

Activities

- Data collection : Data collection is the process of gathering raw data from various internal and external source
- Data ingestion : Data ingestion is the process of importing or transferring collected data into a Big Data storage system or data lake.
- Data cleaning : Data cleaning is the process of detecting and correcting errors, removing duplicate records, and handling missing or inconsistent data to improve data quality.
- Data integration : Data integration is the process of combining data from multiple heterogeneous sources into a unified and consistent dataset.
- Data transformation : Data transformation is the process of converting data into a suitable format or structure for storage, processing, and analysis.

Methods

- Batch processing
- Real-time streaming
- APIs
- Web scraping

- Sensor data collection

3. Data Storage

Data storage involves storing massive amounts of data efficiently and securely.

Storage Technologies

- Hadoop Distributed File System (HDFS)
- Cloud Storage
- NoSQL Databases
- Data Lakes

Characteristics

- Distributed storage
- Fault tolerance
- High scalability
- Cost-effective

4. Data Processing

Data processing transforms raw data into useful information.

Types

Batch Processing

Processes large volumes of stored data at scheduled intervals.

Example: Monthly salary processing.

Stream Processing

Processes data continuously as it arrives.

Example: Credit card fraud detection.

Processing Frameworks

- Apache Spark
- Hadoop MapReduce
- Apache Flink

5. Data Analytics

What is Data Analytics

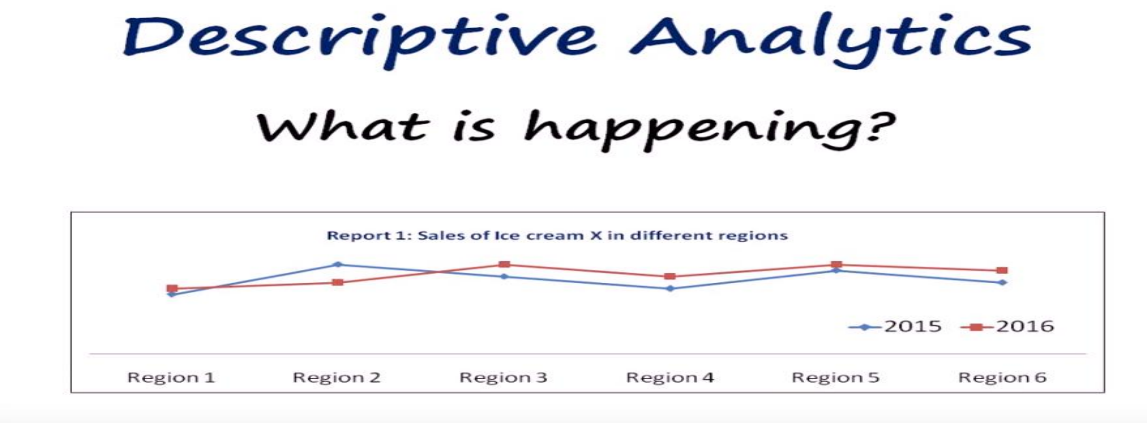
- Data analytics is a broader term that encompasses data analysis.
- Data analytics is a discipline that includes the management of the complete data lifecycle, which encompasses collecting, cleaning, organizing, storing, analyzing and governing data.
- Following figure shows the symbol used to represent analytics.

There are four general categories of analytics that are distinguished by the results they produce:

1. **Descriptive Analytics**
2. **Diagnostic Analytics**
3. **Predictive Analytics**
4. **Prescriptive Analytics**

Descriptive Analytics

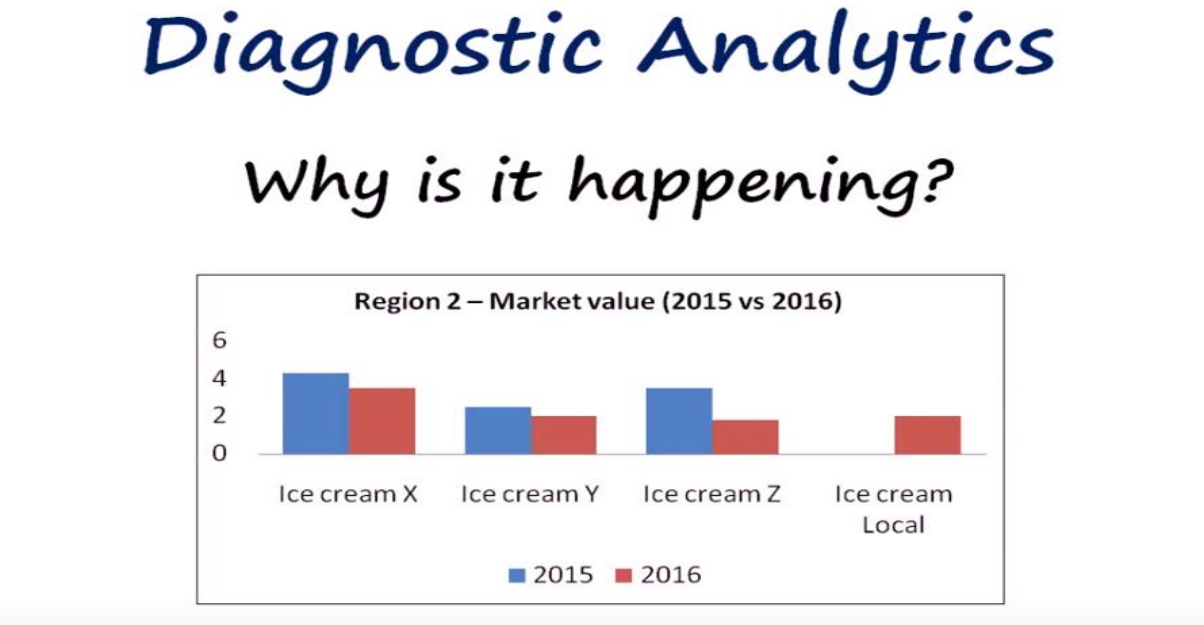
Descriptive Analytics gives you information on what is happening. Let us see a particular Example



This report gives us information about sales of ice cream x in different regions and its also compared these sales for years 2015 and 16.This is a typical descriptive analytics support which gives you information on what is happening to sales of this ice cream so this chart could infer that except for the region 2 the sales have increased from 2015 to 16 so this is the level of information one could obtain from descriptive analytics now that we know the sales have dropped in region 2 .

Diagnostic Analytics

Information on why is it happening . so our job is to deep dive into region 2 and see what exactly is going wrong for us so this is a typical report for the market share of region 2



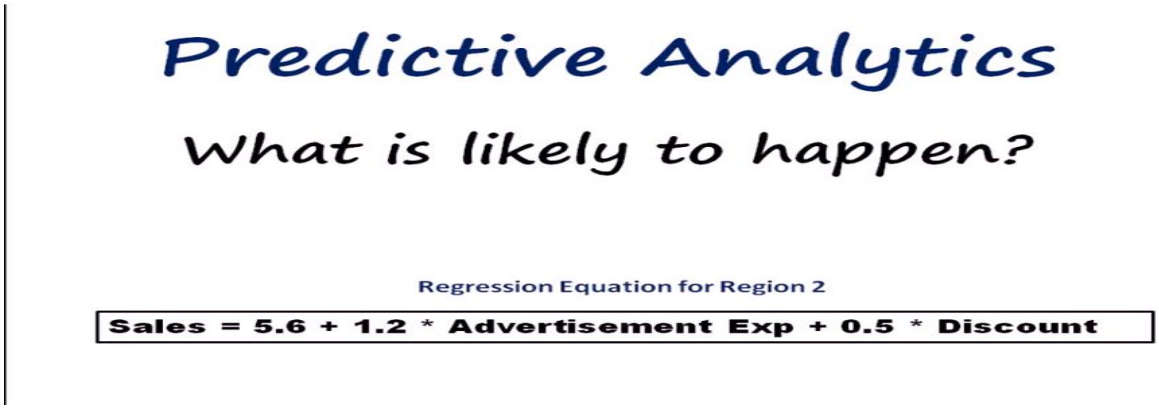
From the graph , it is clear that the sales of ice cream brands x, y and z is dropped from 2015 to 2016 and it is because the local brand ice cream has come up in 2016.In the first step we found what is happening and here we found why it is happening. here we have diagnosed the reason of sales drop in region 2

Predictive Analytics

Once we identified the reason of happenings , then we can take measures to improve things i.e what should be done to make things happen correct. In our example, first we identified region 2 has less sales in 2016 next, we found the reason of why region 2 sales got down and now it is the time to find the solution to increase the sales and that is called predictive data analytics

For this example what we can do is to build a regression for region 2 sales the different factors that would affect the sales of region 2 this would help us change the different factors say for example the

advertisement expense or discount and see what would be the impact of the changes on sales so these are some things that u can do so using predictive analytics which essentially tells you what is likely to happen in future now we know that what had happened and why it had happened and our next job is to find what is likely to happen i.e predictive analytics



Prescriptive Analytics

- Our next step is to move to next topic i. e Prescriptive analysis there you suggest the best course of action to solve the problem.
- for this particular example we can give solutions like increase the advertising expense by 10% or give 5 % discount for two months
- if we are able to find out using the regression equation that these factors like advertising expense or discount has an impact on sales
- so these are the four different levels of business analytics .
- Now lets summarize what we have just seen

6. Data Visualization

Visualization converts analytical results into easy-to-understand graphical formats.

Common Visualization Tools

- Tableau
- Microsoft Power BI
- Apache Superset
- Google Data Studio

Visualization Methods

- Bar Charts
- Pie Charts
- Line Graphs
- Heat Maps
- Dashboards

Advantages

- Easy interpretation
- Faster decision-making
- Better communication of insights

Popular Big Data Platforms

1. Hadoop

Hadoop is an open-source framework used for distributed storage and parallel processing of large datasets across clusters of computers.

Components

- HDFS (Storage)
- MapReduce (Processing)
- YARN (Resource Management)
- Hadoop Common (Utilities)

Advantages

- Highly scalable
- Fault tolerant
- Cost-effective
- Open source

Applications

- Data warehousing
- Log analysis
- Search engines

2. Apache Spark

Apache Spark is a fast, open-source distributed computing framework for Big Data processing.

Features

- In-memory processing
- Faster than Hadoop MapReduce
- Real-time analytics
- Supports Machine Learning

Applications

- Real-time analytics
- Machine Learning
- Streaming data
- Graph processing

3. Apache Hive

Apache Hive is a data warehouse system built on Hadoop that allows users to query large datasets using HiveQL, a SQL-like language.

Features

- SQL-like queries
- Easy data analysis
- Batch processing
- Suitable for reporting

4. Apache Pig

Apache Pig is a high-level platform used for analyzing large datasets using the Pig Latin scripting language.

Features

- Easy scripting
- Data transformation
- Large-scale data analysis
- Runs on Hadoop

5. Apache Kafka

Apache Kafka is a distributed messaging system used for real-time data streaming.

Features

- High throughput
- Fault tolerance
- Real-time processing
- Distributed architecture

Applications

- Event streaming
- Log aggregation
- IoT data collection
- Real-time analytics

6. Apache Cassandra

Apache Cassandra is a distributed NoSQL database designed to handle large volumes of data across multiple servers.

Features

- High availability

- Fault tolerance
- Linear scalability
- No single point of failure

Applications

- Social media
- Banking
- IoT
- Online retail

7. MongoDB

MongoDB is a document-oriented NoSQL database that stores data in JSON-like documents.

Features

- Flexible schema
- High performance
- Horizontal scalability
- Easy integration

Applications

- Content management systems
- Mobile applications
- E-commerce websites
- Big Data applications

What is Big Data

Big Data is a collection of data that is huge in volume, yet growing exponentially with time.. (or) Data that satisfies 5 V's is called Big Data

What is Data?

The quantities, characters, or symbols on which operations are performed by a computer.

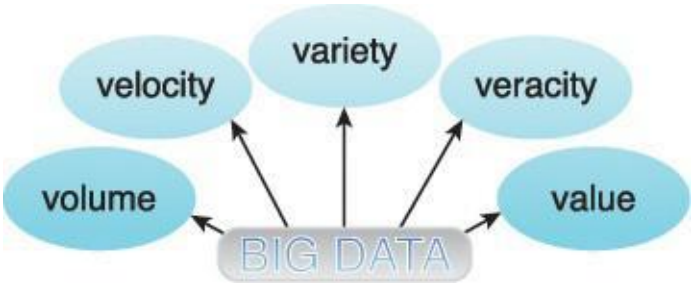
Big Data Characteristics

1) **Volume**
2) **Velocity**
3) **Variety**
4) **Veracity**
5) **Value**

- For a dataset to be considered Big Data, it must possess one or more characteristics like volume, velocity, variety, veracity and value

(or)

- We differentiate Big Data from traditional data by one or more of the five V's: Volume, Velocity, Variety, Veracity and value
- Five Big Data Characteristics that can be used to help differentiate data categorized as "Big" from other forms of data are



Volume

- Refers to the vast amounts of data generated every second.
- we are not talking about Terabytes but zettabytes.
- If we take all the data generated in the world between the beginning of time and 2008, the same amount of data will soon be generated every minute.
- This makes most data sets too large to store and analyse using Traditional database technology.
- Example: Amazon handles 15 million customer click stream user data per day to recommend products.

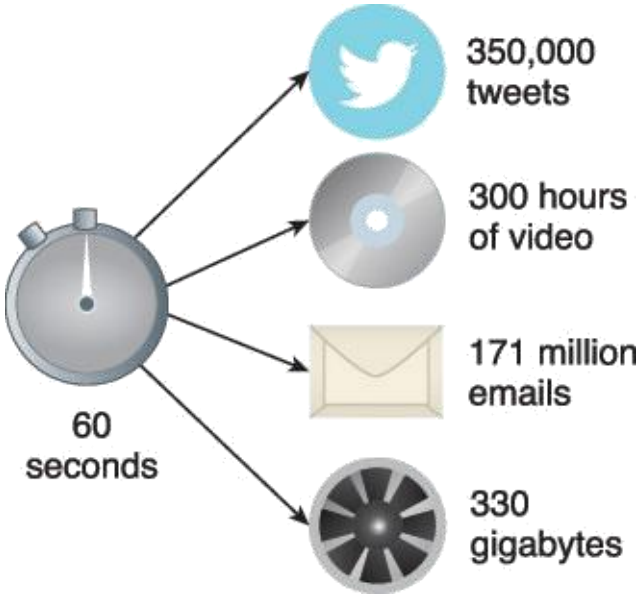
Typical data sources that are responsible for generating high data volumes can include:

- online transactions, such as point-of-sale and banking
- sensors, such as GPS sensors, RFIDs, smart meters and telematics social media, such as Facebook and Twitter

Velocity:

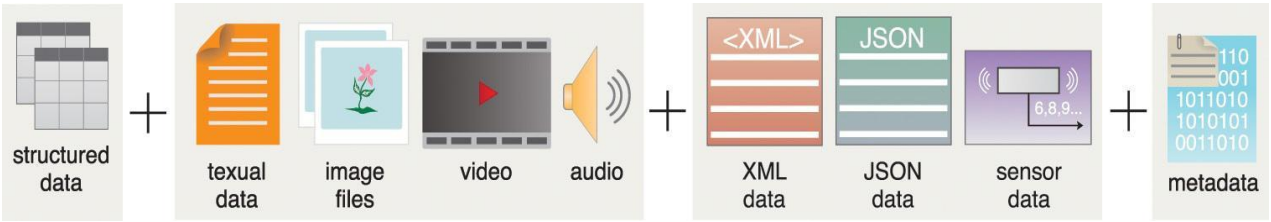
- Refers to the speed at which new data is generated and the speed at which data moves around. Just think of social media messages going viral in seconds

Example: 72 hours of video are uploaded to YouTube every minute this is the velocity. Extremely high velocity of data is another major characteristic of big data.



Variety

Data variety refers to the multiple formats and types of data that need to be supported by Big Data solutions. Data variety brings challenges for enterprises in terms of data integration, transformation, processing, and storage. Following figure provides a visual representation of data variety, which includes structured data in the form of financial transactions, semi-structured data in the form of emails and unstructured data in the form of images.

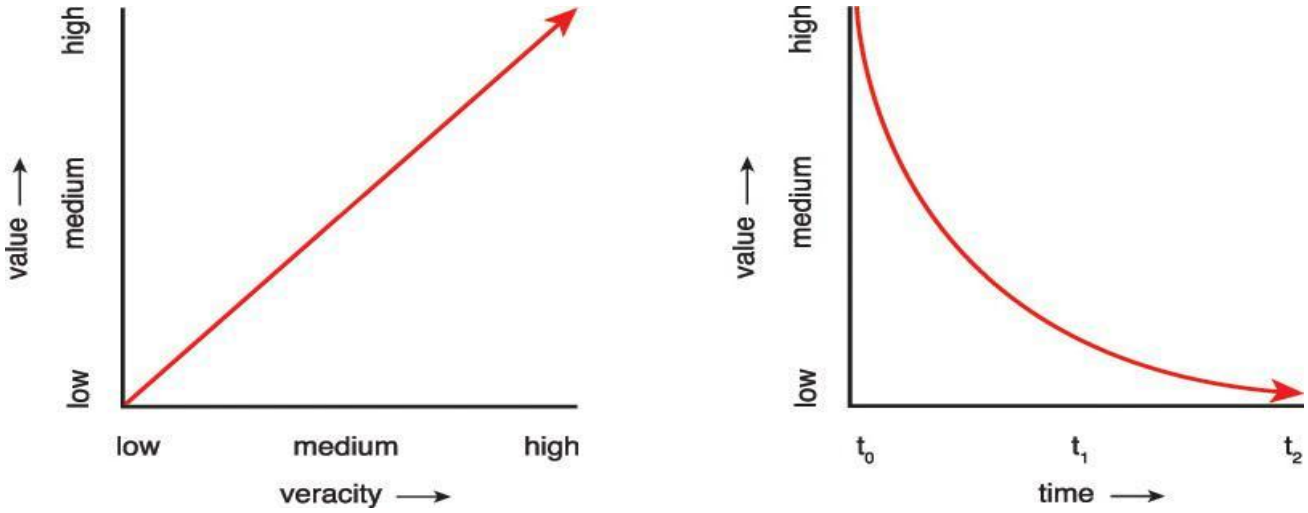


Veracity

Veracity refers to the quality or fidelity of data. Data that enters Big Data environments needs to be assessed for quality, which can lead to data processing activities to resolve invalid data and remove noise. In relation to veracity, data can be part of the signal or noise of a dataset. Noise is data that cannot be converted into information and thus has no value, whereas signals have value and lead to meaningful information. Data with a high signal-to-noise ratio has more veracity than data with a lower ratio. Thus the signal-to-noise ratio of data is dependent upon the source of the data and its type.

Value

Value is defined as the usefulness of data for an enterprise. The value characteristic is intuitively related to the veracity characteristic in that the higher the data fidelity, the more value it holds for the business.



Different Types of Data

The data processed by Big Data solutions can be human-generated or machine-generated, although it is ultimately the responsibility of machines to generate the analytic results. Human-generated data is the result of human interaction with systems, such as online services and digital devices. Following figure shows examples of human-generated data.

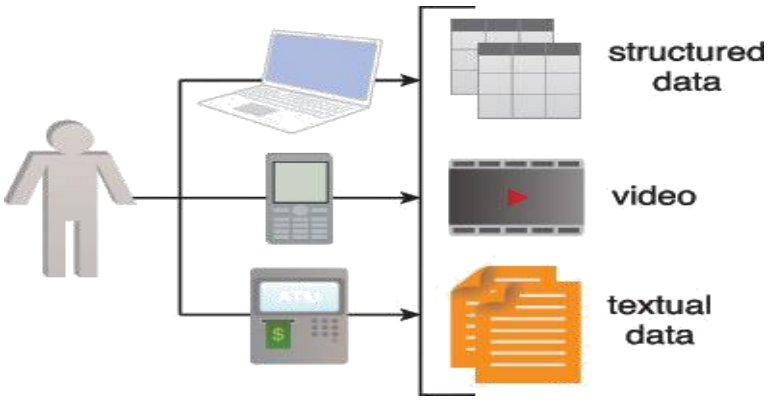


Fig: Examples of human-generated data include social media, blog posts, emails, photo sharing and messaging.

Machine-generated data is generated by software programs and hardware devices in response to real-world events.

As demonstrated, human-generated and machine-generated data can come from a variety of sources and be represented in various formats or types. This section examines the variety of data types that are processed by Big Data solutions. The primary types of data are:

Structured data

Unstructured data

Semi-structured data

Structured Data

Structured data conforms to a data model or schema and is often stored in tabular form. It is used to capture relationships between different entities and is therefore most often stored in a relational database. Structured data is frequently generated by enterprise applications and information systems like ERP and CRM systems. Due to the abundance of tools and databases that natively support structured data, it rarely requires special consideration in regards to processing or storage. Examples of this type of data include banking transactions, invoices, and customer records. Following figure shows the symbol used to represent structured data.

Fig: The symbol used to represent structured data stored in a tabular form.

Unstructured Data

Data that does not conform to a data model or data schema is known as unstructured data. It is estimated that unstructured data makes up 80% of the data within any given enterprise. Unstructured data has a faster growth rate than structured data. Following figure illustrates some common types of unstructured data. This form of data is either textual or binary and often conveyed via files that are self-contained and non-relational.

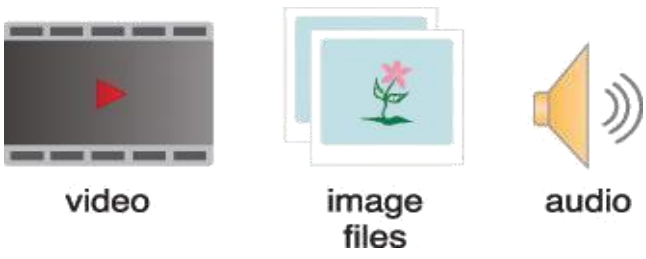


Fig: Video, image and audio files are all types of unstructured data.

Semi-structured Data

Semi-structured data has a defined level of structure and consistency, but is not relational in nature. Instead, semi-structured data is hierarchical or graph-based. This kind of data is commonly stored in files that contain text. For instance following figure shows that XML and JSON files are common forms of semi-structured data. Due to the textual nature of this data and its conformance to some level of structure, it is more easily processed than unstructured data.

Fig: XML, JSON and sensor data are semi-structured.

Best Practices with Big Data

Definition:
Big Data best practices are a set of guidelines and techniques that help organizations collect, store, process, analyze, secure, and manage large volumes of data efficiently. Following these best practices ensures that data is reliable, accessible, secure, and useful for making better business decisions.

1. Know What Data is Important and What it is Not

The first step in managing Big Data is identifying which data is valuable and which data is unnecessary. Organizations collect data from various sources, but not all of it contributes to business goals. Storing irrelevant data increases storage costs and makes analysis more difficult.

By selecting only useful and relevant data, organizations can improve processing speed, reduce storage requirements, and make better decisions.

Key Points

- Identify business-relevant data.
- Remove duplicate or unnecessary data.

- Reduce storage and processing costs.
- Improve decision-making.

2. Ensure High Quality

Data quality is one of the most important factors in Big Data analytics. Poor-quality data leads to incorrect analysis and poor business decisions. High-quality data should be accurate, complete, consistent, reliable, and up-to-date.

Organizations should regularly validate, clean, and update data to maintain its quality.

Key Points

- Data should be accurate and error-free.
- Ensure completeness of records.
- Keep data updated.
- Remove duplicate and inconsistent data.

3. Commit to Proper Data Labeling

Data labeling means assigning meaningful tags or categories to data so that it can be easily identified and used for analysis. Properly labeled datasets are especially important in Artificial Intelligence (AI) and Machine Learning (ML), where algorithms learn from labeled data.

Good data labeling improves searchability, organization, and analytical accuracy.

Key Points

- Label data correctly.
- Improve dataset organization.
- Support AI and Machine Learning.
- Make data easier to search and analyze.

4. Choose Proper Big Data Storage Locations

Selecting the right storage location is essential for managing Big Data efficiently. Organizations may use cloud storage, distributed storage systems, or hybrid storage solutions depending on their requirements.

Data should be stored close to where it is frequently accessed. Metadata should be stored with the actual data, and backups should be maintained in multiple locations.

Key Points

- Use cloud or distributed storage.

- Store metadata with data.
- Keep multiple copies for reliability.
- Ensure easy access to data.

5. Simplify Backup Procedures

Regular backups protect data from accidental deletion, hardware failures, cyber-attacks, and natural disasters. Backup procedures should be simple, automated, and performed frequently.

Organizations should store backup copies at remote locations and regularly test backup integrity to ensure successful recovery.

Key Points

- Perform regular backups.
- Automate backup processes.
- Store backups offsite.
- Verify backup integrity.

6. Implement High Data Security Measures

Big Data often contains confidential and sensitive information. Strong security measures protect data from unauthorized access, cyber threats, and data breaches.

Security measures include encryption, authentication, authorization, firewalls, intrusion detection systems, and regular security audits.

Key Points

- Encrypt sensitive data.
- Use strong authentication.
- Implement access control.
- Monitor security continuously.

7. Plan for the Future by Understanding Trends

Big Data enables organizations to identify patterns and trends that help in future planning. Understanding historical and current data allows businesses to predict customer behavior, market changes, and future opportunities.

Trend analysis supports better strategic planning and informed decision-making.

Key Points

- Analyze historical data.
- Identify future trends.

- Improve business planning.
- Make informed decisions.

8. Use Your Tools Effectively – They Will Make Your Life Easier

Big Data tools such as Hadoop, Spark, Tableau, Power BI, and business intelligence (BI) software simplify data processing and analysis. Organizations should use these tools effectively to gain valuable insights and improve productivity.

Proper use of analytical tools helps understand customer behavior, product performance, and business operations.

Key Points

- Learn available Big Data tools.
- Use BI tools for analysis.
- Improve productivity.
- Gain valuable business insights.

9. Find Ways to Make Your Data More Accessible

Data accessibility ensures that authorized users can easily access the information they need. Poor accessibility delays decision-making and reduces organizational efficiency.

Organizations should use centralized storage, proper indexing, metadata management, and user-friendly interfaces to improve data accessibility while maintaining security.

Key Points

- Improve data sharing.
- Provide secure access.
- Use centralized repositories.
- Make data easily searchable.

Explore the Opportunities with Bigdata

Better information management: Big Data helps companies find, access, and use information more effectively. It can also uncover new types of data that weren't being used before to add value.

Better supply chain tracking: Big Data makes it easier to track and manage supply chains by giving a clearer view of operations in real time, no matter where the data is stored.

Faster response: With Big Data, companies can quickly adjust to changes in the market, speed up product launches, and make supply chains more reliable.

More teamwork and collaboration: Big Data helps different teams within a company and key partners work together more effectively.

Better logistics: Big Data makes it easier to adjust schedules, plan the best routes, change routes when needed, and plan roadside services in real time.

Innovation and product design: Different types of data, like how products are used, sales information, data from devices, customer feedback, and ideas from suppliers, can help create new products and improve existing ones.

Better financial results: Big Data can help businesses save money in the long run, make smarter investments, and understand what drives costs and how they affect the business.

Better product and market strategy: Big Data helps businesses understand their customers better, so they can target the right people and grow more easily. It also improves customer service and boosts sales by using tools like websites and social media.

Better demand and production planning: Big Data helps businesses plan product launches and releases more effectively. It also allows for more detailed planning, which leads to faster and more efficient decision-making.

Security, Compliance, Auditing, and Protection in Big Data

As organizations increasingly depend on Big Data to support business operations and decision-making, protecting data becomes a major concern. Big Data systems process enormous volumes of structured, semi-structured, and unstructured data collected from multiple sources. Therefore, organizations must ensure that the data remains **secure, compliant with regulations, auditable, and protected** against unauthorized access and failures.

1. Security

Security is the process of protecting Big Data from unauthorized access, modification, disclosure, destruction, and cyber-attacks. Due to the distributed nature of Big Data platforms, security is more complex than in traditional database systems.

The main objectives of Big Data security are:

- Protect confidential information.
- Prevent unauthorized access.
- Ensure data integrity.
- Maintain data availability.
- Protect data during storage and transmission.

Security Challenges

The sheer size of Big Data brings significant security challenges. Security involves much more than preventing attackers from accessing the system. It also includes protecting data against corruption, accidental loss, hardware failures, and malicious activities.

Organizations use various security mechanisms such as:

- User Authentication
- Authorization
- Data Encryption
- Firewalls
- Intrusion Detection Systems (IDS)
- Network Security
- Backup and Disaster Recovery

Types of Security Controls

Data Access

Data access determines who can read, modify, or delete data.

- Only authorized users should access sensitive information.
- Organizations implement Role-Based Access Control (RBAC).
- Multi-factor authentication provides additional protection.

Data Availability

Data should always be available to authorized users whenever required.

Availability is achieved through:

- Data replication
- Distributed storage
- Fault tolerance
- Regular backups
- Disaster recovery plans

Performance

Security mechanisms such as encryption and continuous monitoring improve protection but also consume system resources.

Therefore, organizations must maintain a balance between security and system performance.

2. Compliance

Compliance refers to following legal, regulatory, industry, and organizational policies related to data collection, storage, processing, and sharing.

Organizations handling customer information must comply with applicable laws and standards to ensure privacy and proper data usage.

Objectives of Compliance

- Protect customer privacy.
- Meet government regulations.
- Avoid legal penalties.
- Maintain customer trust.
- Ensure ethical use of data.

Compliance Activities

- Data privacy management
- Data retention policies
- Access control policies
- Security policy implementation
- Regular compliance reviews
- Employee awareness programs

Failure to comply with regulations may result in heavy financial penalties and damage to the organization's reputation.

3. Auditing

Auditing is the systematic process of recording, monitoring, and reviewing all activities performed on Big Data systems.

Audit logs help organizations identify who accessed data, when it was accessed, and what operations were performed.

Objectives of Auditing

- Monitor user activities.
- Detect unauthorized access.
- Investigate security incidents.
- Ensure policy compliance.
- Support forensic investigations.

Audit Information Includes

- User login details

- Data accessed
- Changes made to data
- Time and date of access
- Failed login attempts
- Administrative activities

Regular auditing improves accountability and helps detect security threats at an early stage.

4. Protection

Protection refers to safeguarding Big Data against accidental loss, corruption, hardware failures, software failures, cyber-attacks, and natural disasters.

Data protection ensures that information remains confidential, accurate, and available throughout its lifecycle.

Data Protection Techniques

- Data Encryption
- Regular Backups
- Data Replication
- Access Control
- Secure Storage
- Disaster Recovery Planning
- Malware Protection
- Network Security
- Continuous Monitoring

Data should be protected both **at rest** (stored data) and **in transit** (data moving across networks).

Case Study: Big Data in Amazon E-Commerce Platform

Title

Case Study: How Amazon Uses Big Data to Improve Customer Experience and Business Performance

1. Introduction

Amazon is the world's largest online retailer and one of the biggest users of Big Data technologies. Every second, millions of customers search for products, read reviews, compare prices, place orders, and track shipments. Amazon collects and processes this enormous volume of data to provide personalized services, optimize business operations, and increase customer satisfaction.

Big Data helps Amazon understand customer behavior, predict future demand, improve inventory management, recommend products, detect fraud, and ensure fast delivery. The company combines advanced analytics, artificial intelligence (AI), machine learning (ML), cloud computing, and distributed storage systems to analyze petabytes of data every day.

2. Company Background

Company: Amazon

Founded: 1994

Founder: Jeff Bezos

Headquarters: Seattle

Industry: E-commerce, Cloud Computing, Artificial Intelligence

Employees: More than 1.5 million worldwide

Customers: Hundreds of millions across more than 200 countries

Amazon's success depends heavily on data-driven decision-making.

3. Business Challenges

Before implementing advanced Big Data technologies, Amazon faced several challenges:

Huge Volume of Data

Millions of customer visits daily

Billions of product pages

Millions of transactions every hour

Personalized Shopping

Each customer has different interests and purchasing habits.

Inventory Management

Predicting product demand accurately across thousands of warehouses.

Fraud Detection

Identifying fake transactions and suspicious activities.

Fast Delivery

Optimizing delivery routes and warehouse operations.

Dynamic Pricing

Changing product prices according to demand, competition, and stock availability.

4. Types of Data Collected

Amazon collects structured, semi-structured, and unstructured data.

Data Type	Examples
Customer Data	Name, address, payment information
Browsing Data	Searches, clicks, page views
Purchase History	Previous orders, returns
Product Data	Price, category, ratings
Review Data	Customer reviews and comments
Logistics Data	Shipment tracking, warehouse information
Device Data	Mobile, desktop, tablet usage
Seller Data	Product listings, inventory

5. Big Data Characteristics (5Vs)

Volume

- Petabytes of customer and transaction data
- Billions of records

Velocity

- Real-time transactions
- Live product recommendations
- Instant order processing

Variety

- Text
- Images
- Videos
- Customer reviews
- Clickstream data

GPS location

Veracity

Removing duplicate data

Detecting fake reviews

Data validation

Value

Converting raw data into useful business insights.

6. Big Data Architecture Used by Amazon

7. Big Data Technologies Used

Technology	Purpose
Hadoop	Distributed data storage
HDFS	Large-scale file storage
Spark	Fast data processing
Hive	SQL analysis
NoSQL Databases	Product catalog
Machine Learning	Prediction models
AI	Personalized recommendations
Cloud Computing	Scalable infrastructure

8. Applications of Big Data at Amazon

A. Personalized Product Recommendation

Amazon recommends products based on:

Previous purchases

Search history

Recently viewed items

Customer interests

Similar customer behavior

Example

- If a customer purchases:

- Laptop
- Mouse

Amazon recommends:

- Laptop Bag
- Keyboard
- USB Hub
- External Hard Disk
- Antivirus Software

This increases cross-selling and customer satisfaction.

B. Dynamic Pricing

Amazon changes prices automatically based on:

- Customer demand
- Competitor prices
- Stock availability
- Seasonal sales
- Festival offers

Example:

A smartphone priced at ₹25,000 today may become ₹23,999 tomorrow depending on market demand.

C. Demand Forecasting

Machine learning predicts:

- Future sales
- Seasonal demand
- Festival shopping
- Warehouse requirements

Benefits:

- Avoids stock shortages
- Reduces excess inventory
- Improves customer satisfaction

D. Inventory Management

Big Data helps Amazon decide:

- Which products should be stored?
- Which warehouse should store them?
- How much stock is required?

Benefits:

- Faster delivery
- Lower storage costs
- Better inventory utilization

E. Fraud Detection

Big Data identifies:

- Fake transactions
- Suspicious login attempts
- Payment fraud
- Fake seller accounts

Machine learning continuously learns new fraud patterns.

F. Customer Sentiment Analysis

Amazon analyzes:

- Product reviews
- Customer feedback
- Ratings
- Complaints

Natural Language Processing (NLP) identifies:

- Positive opinions
- Negative opinions
- Neutral comments

Example:

"Battery backup is excellent but camera quality is poor."

The review is analyzed into positive and negative sentiments.

G. Supply Chain Optimization

Amazon uses Big Data to optimize:

- Warehouse operations
- Delivery routes
- Transportation
- Packaging
- Delivery scheduling

Benefits:

- Faster delivery
- Lower fuel costs
- Better customer satisfaction
- H. Voice Shopping using Alexa

Amazon Alexa collects customer interactions and provides:

- Voice search
- Product recommendations
- Shopping assistance
- Order tracking

9. Machine Learning Models Used

Amazon uses different ML algorithms.

Algorithm	Application
Collaborative Filtering	Product recommendations
Decision Trees	Customer classification
Random Forest	Fraud detection
Neural Networks	Personalized advertising
Time Series Forecasting	Demand prediction
Deep Learning	Image recognition

10. Big Data Analytics Process

Customer Activity

Data Collection

Data Cleaning

Data Storage

Big Data Processing

Machine Learning

Business Insights

Decision Making

11. Benefits Achieved

Benefit	Description
Better Customer Experience	Personalized shopping
Higher Sales	Improved recommendations
Faster Delivery	Optimized logistics
Inventory Optimization	Reduced stock waste
Fraud Prevention	Secure transactions
Customer Retention	Increased loyalty
Business Growth	Improved profits

12. Challenges Faced

- Managing petabytes of data
- Data privacy and security
- Real-time processing
- High infrastructure cost
- Fake product reviews
- Maintaining recommendation accuracy
- Cybersecurity threats

13. Results

Amazon has achieved:

- Highly personalized shopping experiences
- Faster order processing
- Improved inventory management
- Better customer retention
- Reduced operational costs
- Accurate demand forecasting
- Efficient logistics and delivery
- Increased revenue and profitability

Textbook:

Michael Minelli, Michelle Chambers, Ambiga Dhiraj – *Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses* (Wiley, 2013)

UNIT-2
APACHE HADOOP

Unit Overview

This unit introduces **Apache Hadoop**, one of the most widely used open-source frameworks for storing and processing large-scale data in a distributed computing environment. It begins with an introduction to Hadoop, its history, and the need for distributed data processing. The unit explains the **Apache Hadoop ecosystem**, including its core components and supporting tools. It covers the important features of Hadoop such as **fault tolerance through data replication, distributed and parallel processing, high availability, and data locality**, which enable efficient processing of massive datasets. The unit also focuses on the **Hadoop Distributed File System (HDFS)**, its architecture, design principles, and fundamental concepts, providing a foundation for understanding distributed storage in Big Data environments.

Learning Outcomes

After completing this unit, students will be able to:

1. **Explain the need for Hadoop** and its role in Big Data processing.
2. **Describe the history and evolution of Apache Hadoop** and its significance in distributed computing.
3. **Identify the components of the Hadoop ecosystem** and explain their functions.
4. **Explain the key features of Hadoop**, including fault tolerance, data replication, distributed processing, parallel processing, high availability, and data locality.
5. **Describe the architecture and design principles of the Hadoop Distributed File System (HDFS).**
6. **Explain the fundamental concepts of HDFS**, including blocks, NameNode, DataNode, replication, and fault tolerance.
7. **Analyze how Hadoop provides scalable, reliable, and cost-effective storage and processing** for Big Data applications.

Importance of Studying this Unit

The exponential growth of data generated from social media, IoT devices, cloud applications, scientific research, healthcare, banking, and e-commerce has created the need for efficient storage and processing frameworks. Traditional systems are unable to manage such massive datasets effectively. **Apache Hadoop** addresses this challenge by providing a scalable, fault-tolerant, and distributed computing platform that can process Big Data across clusters of commodity hardware.

Studying this unit enables students to understand the architecture and working principles of Hadoop and HDFS, which form the backbone of many modern Big Data platforms. The knowledge gained serves as a foundation for advanced topics such as **MapReduce, Apache Spark, Hive, Pig, HBase, YARN, Data Analytics, Cloud Computing, Artificial Intelligence, and Data Engineering**. It also prepares students for careers in **Big Data Analytics, Data Engineering, Cloud Computing, and Distributed Systems**.

Introduction to Apache Hadoop

- It provides a software framework for distributed storage and processing of big data using the MapReduce programming model.
- The Apache Hadoop is an Open Source Framework. ?Hadoop can easily handle a large amount of data on low cost, simple hardware cluster.
- Hadoop is also a scalable and Fault –tolerant framework.
- Hadoop is not only a storage system as well as data processing using framework.

Introduction to Apache Hadoop

- Originally designed for computer clusters built from commodity hardware.
- It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.
- The library itself is designed to detect and handle failures at the application layer.
  - *If Hardware failures are common occurrences and should be automatically handled by the framework.*

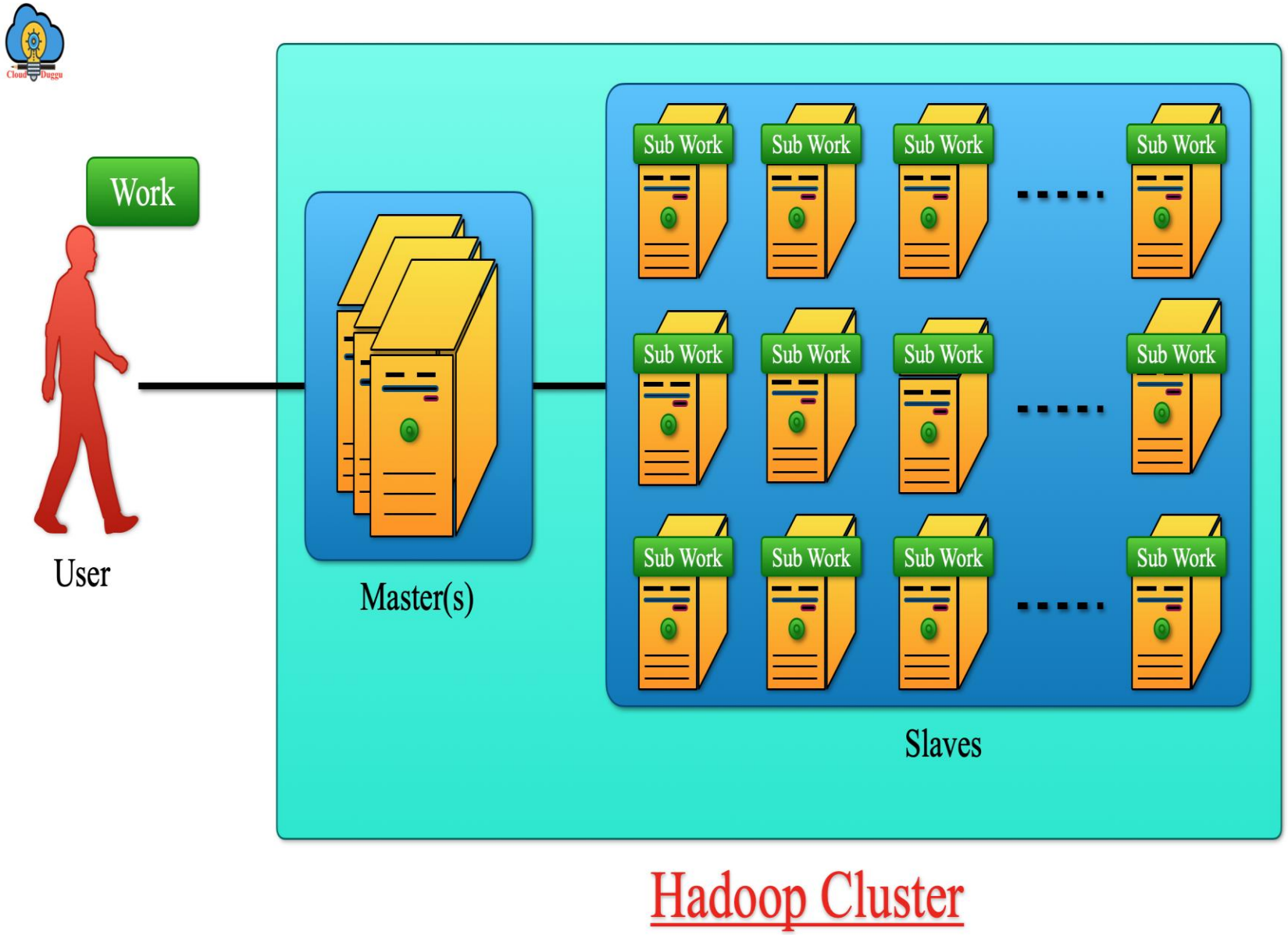
Introduction to Apache Hadoop

- The Core of Apache Hadoop consists of
  - Storage part
    - ***Hadoop Distributed File System (HDFS),***
  - Processing part
    - ***MapReduce programming model.***

Hadoop Technology

- Hadoop is open source tool from the Apache Software Foundation.
- Hadoop codes are written by Yahoo, IBM, Cloudera etc…
- Hadoop provides parallel processing through different commudity h/w simultaneously.

Hadoop Cluster



What is Hadoop?

- Framework for Large-Scale Data Processing
- Inspired by Google's Architecture:
  - **Google File System (GFS)**
  - **MapReduce**
- Open-source Apache project
  - **Nutch search engine project**
  - **Apache Incubator**
- Written in Java and shell scripts

Why we should use Hadoop?

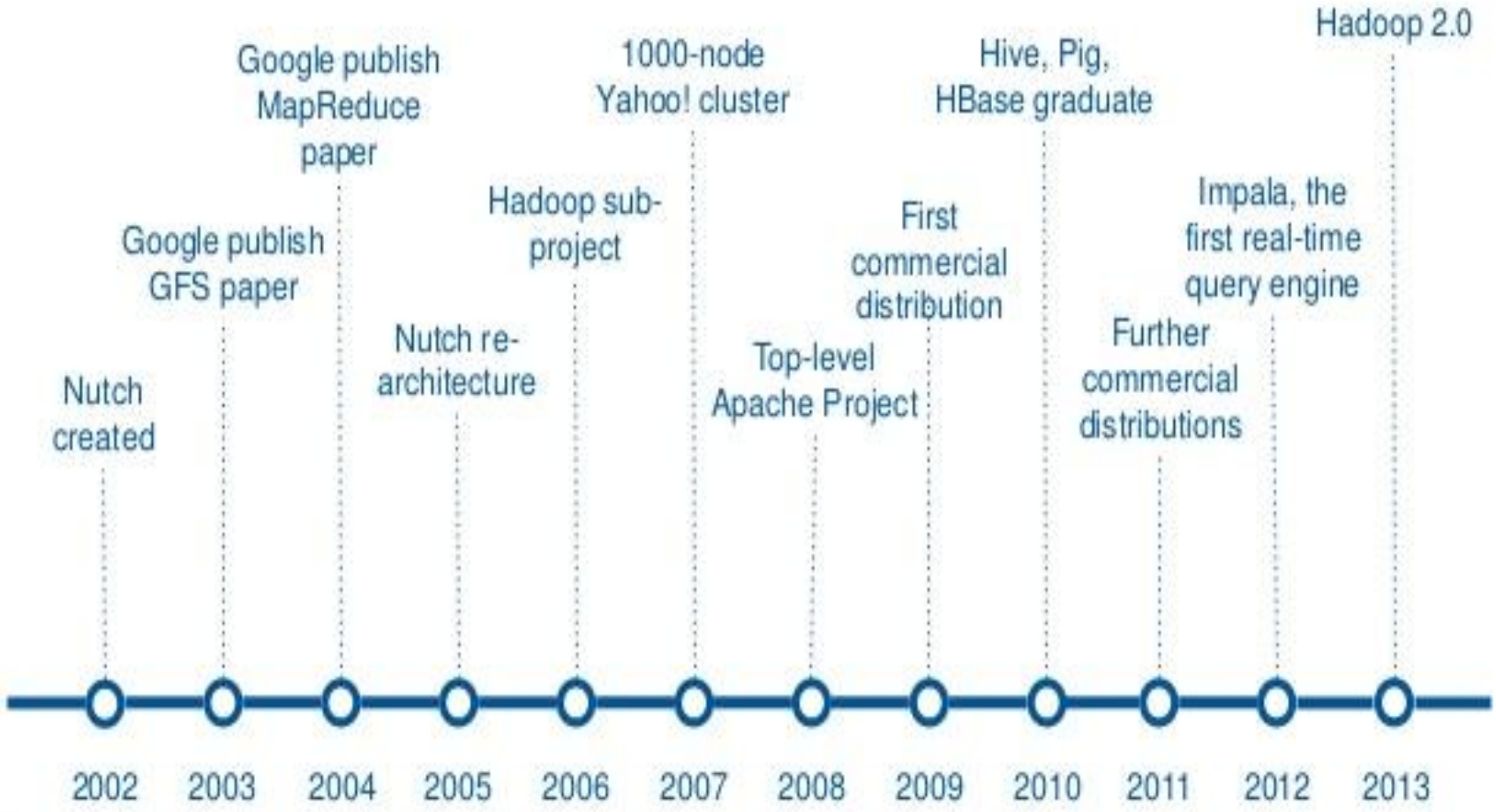
- Hadoop solution is very popular. It captured at least 90% of bigdata market.
- Hadoop splits files into large blocks and distributes them across nodes in a cluster.
- Then, transfers packaged code into nodes to process the data in parallel.
- Its fault tolerant soluction. So we can use no. of commodity h/w easily.
- Data can be stored as structured, unstructured and semi-structured mode. So its more flexible.

Who use Hadoop?

When should you choose Hadoop?

- Need to process a lot of unstructured data
- Processing needs are easily run in parallel
- It uses the concept of MapReduce which enables it to divide the query into small parts and process them in parallel.
- Hadoop has ability to analyze the data present in diff. machines at diff. location very quickly and in a very cost effective way.

A Brief History of Hadoop



Brief History of Hadoop

In 2002, **Doug Cutting and Mike Cafarella** started to work on Apache Nutch project.

Nutch is a open source web crawler software project

After a lot of Research on Nutch, they concluded that this project is very expensive

0.5 million dollars for hardware , Running cost/month $30,000 approximately

In 2003, They came across Google's white paper on GFS(Google File System)

This paper described the architecture of Google's distributed file system, for storing the large data sets.

But it was just the half solution to their problem

In 2004, Google published one more paper on the Mapreduce Technique

This paper was another half solution and now they had complete solution for their Nutch Project

Brief History of Hadoop

So, they started implementing Google's Techniques (GFS &Mapreduce) as open-source in the Apache Nutch Project

In 2005, Cutting found that nutch is limited to only 20-to-40 node clusters

He soon realized 2 problems

Nutch wouldn't achieve its potential until it can reliably on the larger clusters

And that was looking impossible with just two people

Doug Cutting joined Yahoo along with Nutch Project

He wanted to provide the world with an open-source,reliable,scable,computing framework, with the help of yahoo

So at yahoo first, he separates the distributed computing parts from Nutch and formed a new project "Hadoop"

In 2007, Yahoo successfully tested Hadoop on a 1000node cluster and Started using it

In 2008, In January, Yahoo released Hadoop as an open source project to AFS(Apache Software Foundation)

Brief History of Hadoop

In July, Apache Software Foundation successfully tested a 4000 node cluster with Hadoop

In 2009, Hadoop was successfully tested to sort a PB (Petabyte) of data in less than 17 hours

Doug Cutting left the Yahoo and joined Cloudera to fulfill the challenge of spreading Hadoop to other industries

In December 2011

Apache Software Foundation released Apache Hadoop version 1.0

In August 2013

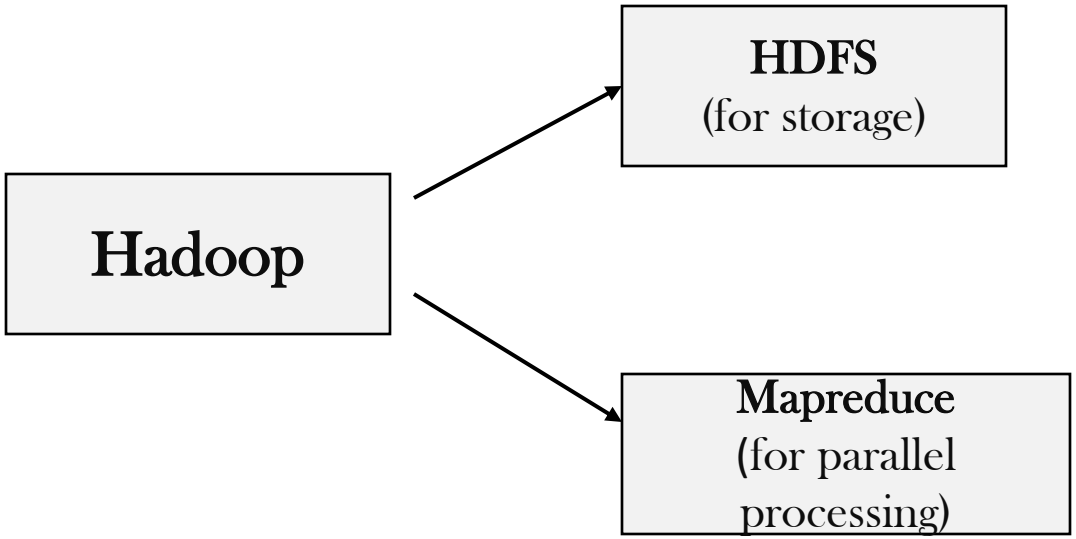
Version 2.0.6 was available

In December 2017

Apache Hadoop version 3.0 released which is currently we are having

Understanding Hadoop Features

- Doug cutting" and "mike cafarella" introduced Hadoop in 2006.
- Hadoop is a open source framework for "storing and processing" huge dataset.
- Hadoop is a open source tool that allows to store and process big data in a distributed environment across cluster of computer using simple programming model for processing.
- Core components of Hadoop are: "HDFS" and "Mapreduce"



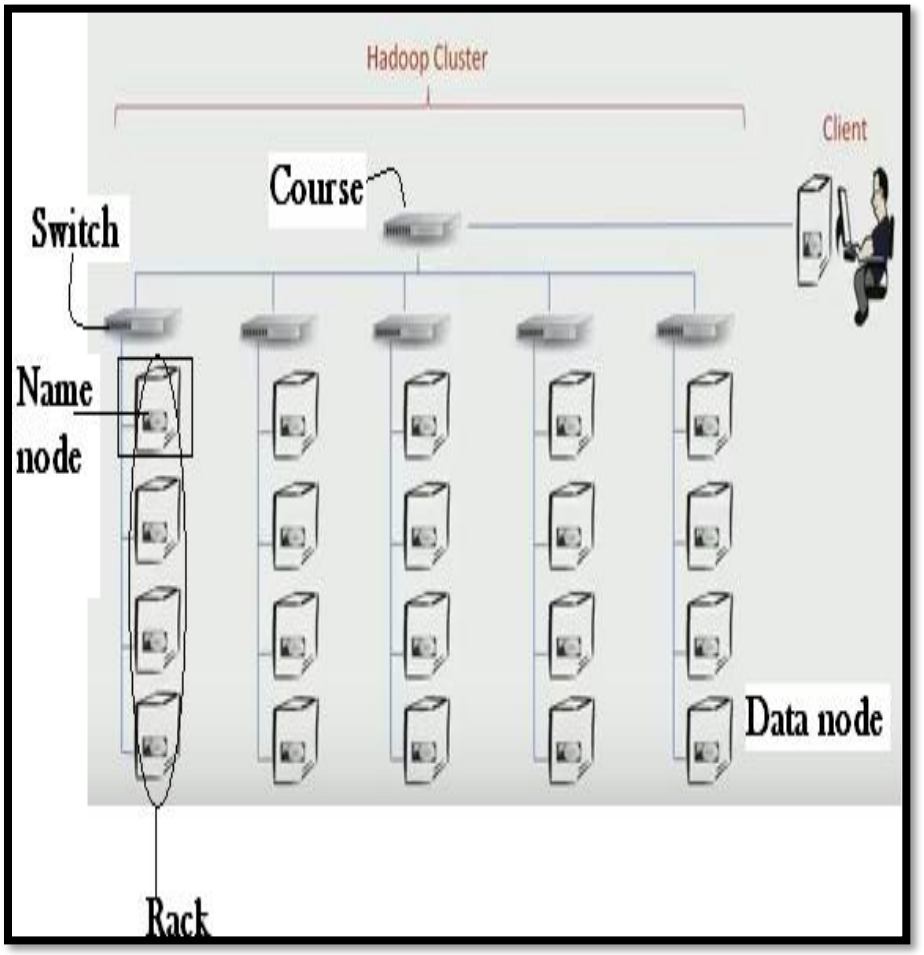
- The name “Hadoop” is not an acronym, it is madeup name. The projects creator “Doug cutting” explains how the name came about.
- The name, my kid gave to the stuffed yellow elephant. Short, relatively easy to pronounce, meaningless and not used else anywhere. Those are my naming criteria. Kids are good at generating search names

6.1 UNDERSTANDING HDFS

DISTRIBUTED STORAGE

Data is stored in a Distributed Manner

- Instead of relying on a single large machine , HDFS uses a network of several smaller machine and add them to a cluster this approach is called Horizontal scaling.
- So, we use a software that combines the storage capacity of the entire network into a single large system i.e network based file system and that is HDFS.



Hadoop client will send a request to name node that it want to create a file along **with target directory name and file name**

By receiving the request the **name node will perform various checks like is the file already exist and client has right permission to create a file.**

Name node does all this because it maintains an image of entire HDFS namespace into memory we call **it file system image.**

If all test pass the name node will **create an entry for new file and return success to client.**

Now, empty file is created.

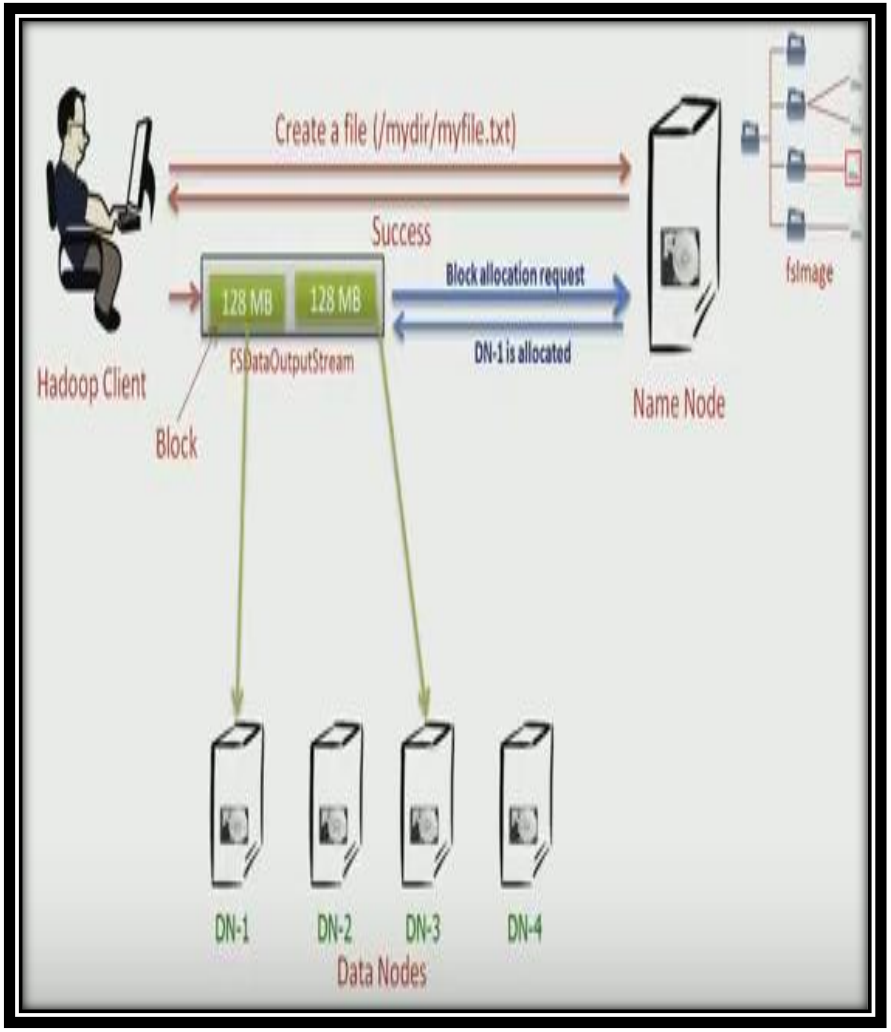
The client start writing **fsdata output stream.**

This stream internally buffers data until you accumulate the reasonable amount of data with default size **128 MB as a block**.

Each block send **block allocation request** to name node asking for location for storing that block name node will allocate a data node and send back data node name to the streamer.

Now streamer will send the **data to the data node.**

If file is large it creates a new block allocation and does the same work.



SCALABILITY

- As HDFS stores the large size data over multiple nodes, so when the requirement of data storing is increased or decreased the number of nodes can be scaled up or scaled down in a cluster.
- The vertical and Horizontal scalability is the 2 different mechanisms available to provide scalability in the cluster.
- Vertical scalability means adding the resource like Disk space, RAM on the existing node of the cluster.
- On another hand in Horizontal scaling, we increase the number of nodes in the cluster and it is more preferable since we can have hundreds of nodes in a cluster.

SCALABILITY

Scalability

- The primary benefit of Hadoop is its **Scalability.**
- One can easily scale the cluster by adding more nodes.
- There are two types of Scalability in Hadoop: **Vertical and Horizontal**

Vertical scalability

- It is also referred as "**scale up".** In vertical scaling, you can increase the **hardware capacity of the individual machine.**
- In other words, you can add more RAM or CPU to your existing system to make it more robust and powerful.

Horizontal scalability

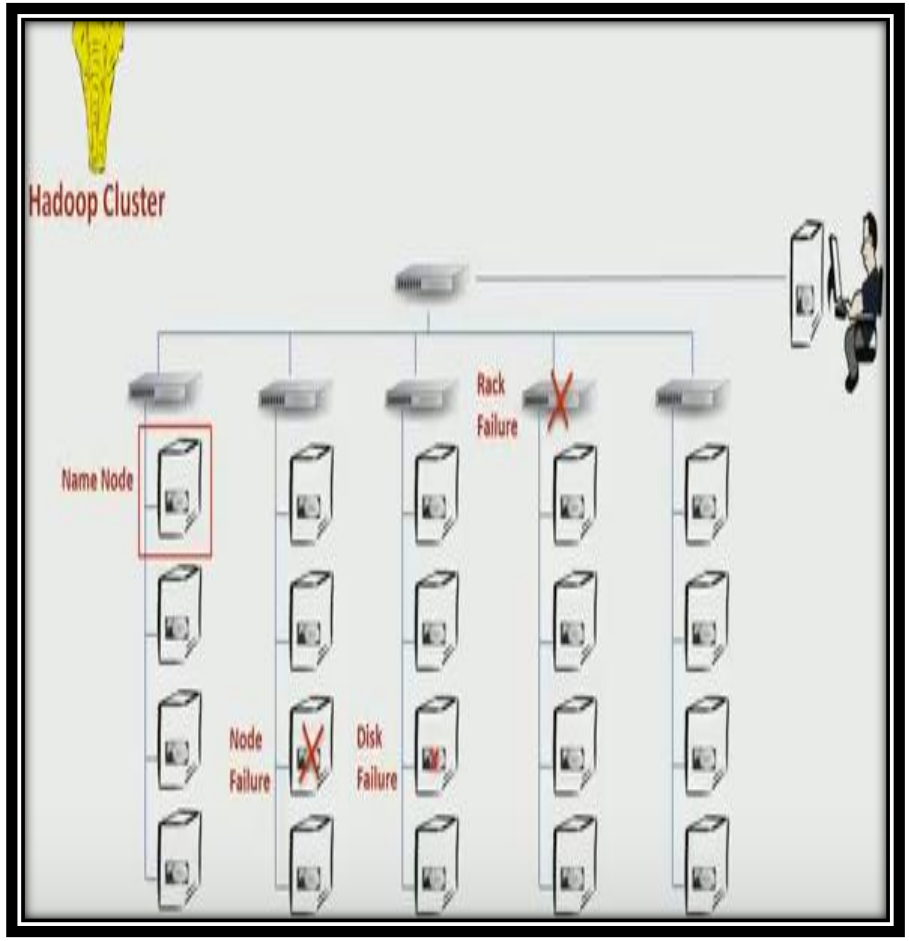
- It is also referred as "**scale out"** is basically the addition of more machines or setting up the cluster.
- In horizontal scaling instead of increasing hardware capacity of individual machine **you add more nodes to existing cluster**

FAULT TORELANCE

- 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 Data Nodes containing the replicas of that data blocks are available.
- ***Thus ensuring no loss of data and makes the system reliable even in unfavorable conditions.***

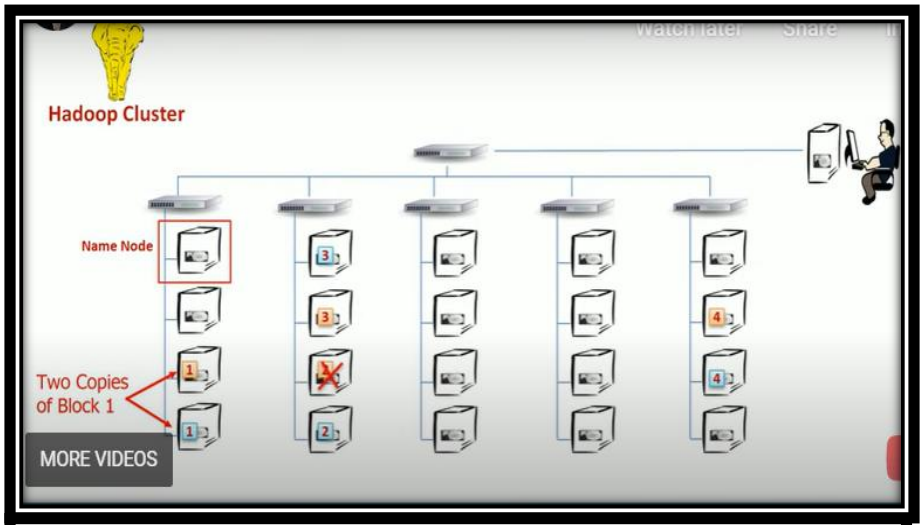
Types of Failures

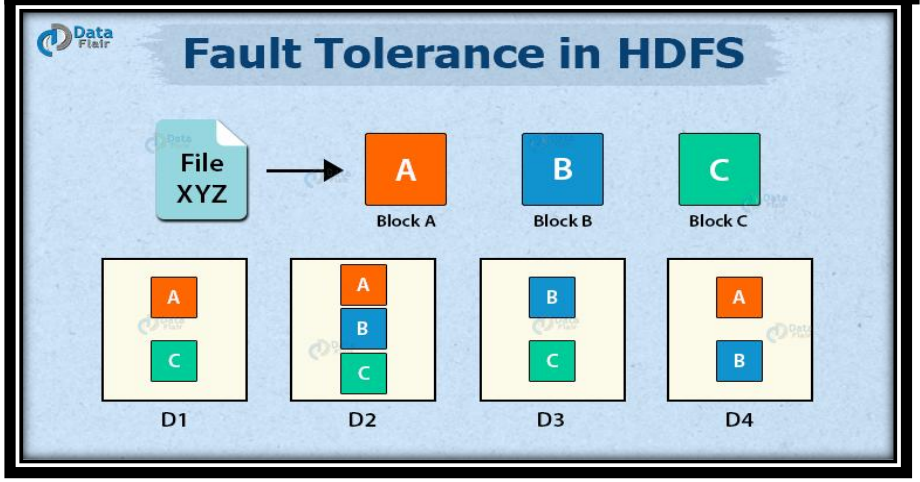
- Different Types of Failures that can happen in Hadoop Cluster are
  - ✓ Disk Failure
  - ✓ Node Failure
  - ✓ Rack Failure
  - ✓ Name Node Failure
- Replication helps to protect Data from **Disk and Node Failure**



FAULT TOLERANCE

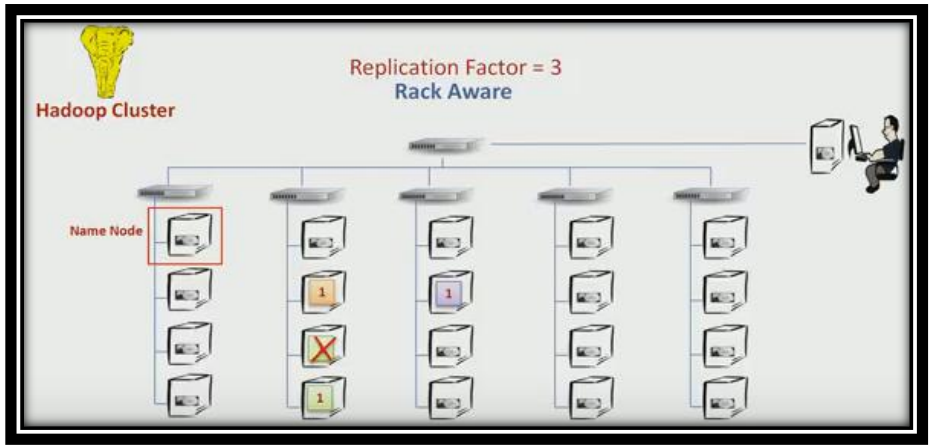
- **Fault tolerance refers to the ability of the system to work or operate even in case of unfavorable conditions (like components failure).**
- HDFS is **highly fault-tolerant**. it handles faults by the process of **replica creation**.
- It creates a replica of users' data on different machines in the HDFS cluster.
- So whenever if any machine in the cluster goes down, then data is accessible from other machines in which the same copy of data was created.
- We can set this replication on file to file basic, (by default its value is 3)
- if we set replication value as 3, HDFS will automatically makes 3 copies of each block for a file.
- Hadoop will also ensure that it keeps these three copies on rack Aware by placing atleast one node in different rack, so that if a complete rack get failed the data can be backed up easily.

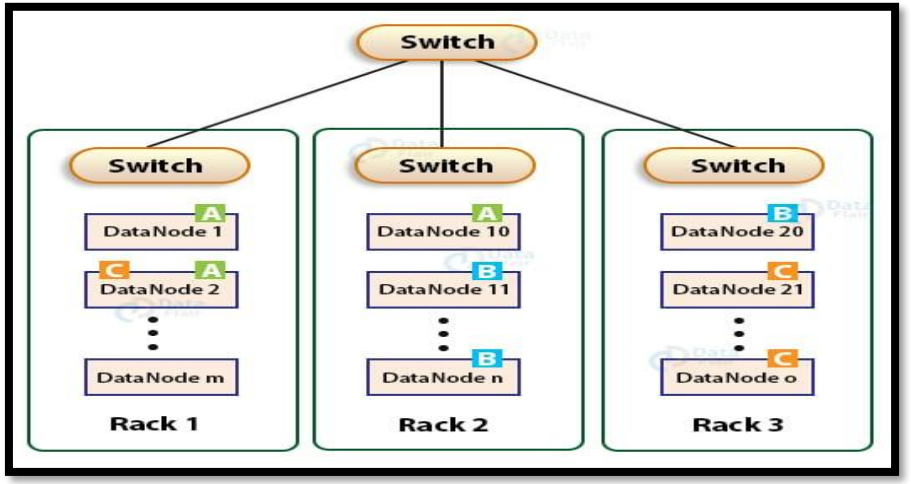




Replica placement via Rack awareness in Hadoop

- HDFS stores replicas of data blocks of a file to provide **fault tolerance and high availability**.
- ***If we store replicas on different nodes on the same rack, then it improves the network bandwidth, but if the rack fails (rarely happens), then there will be no copy of data on another rack.***
- Again, if we store replicas on unique racks, then due to the transfer of blocks to multiple racks while writes increase the cost of writes.
- Therefore, **NameNode on multiple rack cluster maintains block replication by using inbuilt Rack awareness policies which says:**
  - ✓ **Not more than one replica be placed on one node.**
  - ✓ **Not more than two replicas are placed on the same rack.**
- For the common case where the replication factor is three, the block replication policy put the first replica on the local rack, a second replica on the different DataNode on the same rack, and a third replica on the different rack.





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.

HIGH AVAILABILITY

Although Rack, node and disk failure happens, these faults doesn't bring the entire system down your Hadoop cluster remains available, which shows Hadoop is rich in high availability.

What happens when the name node fails?

- We know that name node maintains the file system namespace.
- It also manages the file to block mapping.
- Every client interaction start with name node, so if name node fails we can't use Hadoop cluster.
- Name node is a single point of failure in cluster.
- To achieve high availability we need to protect it against name node failure

How to protect against name node(NN) failure?

- The solution is to back up the namespace content (HDFS namespace information)

HDFS namespace information:

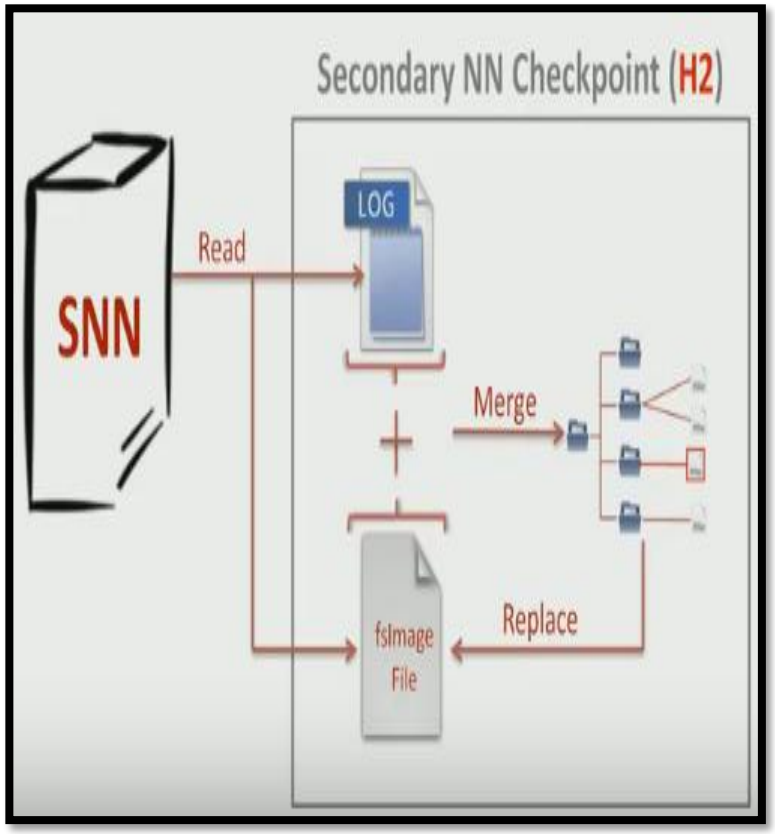
- All the information that name node maintains, should be continuously backed up at some other place.
- so that in the case of failure you have all necessary information to start a new name node on different machine.

SECONDARY NAME NODE

- Secondary NN differs from standby NN.
- Standby NN is a backup for Active NN, whereas secondary is not for that.
- Whenever we restart the name node we will lose the fsImage because it is an in-memory image (In-memory fsImage is the latest and updated picture of Hadoop file system namespace).
- Of course we have editlog to create new fsImage, but it may take more time to read the editlog because the log keeps growing bigger & bigger everyday this directly impact the restart time for a name node which may take hours of time.
- So to solve this problem we came with secondary NN.

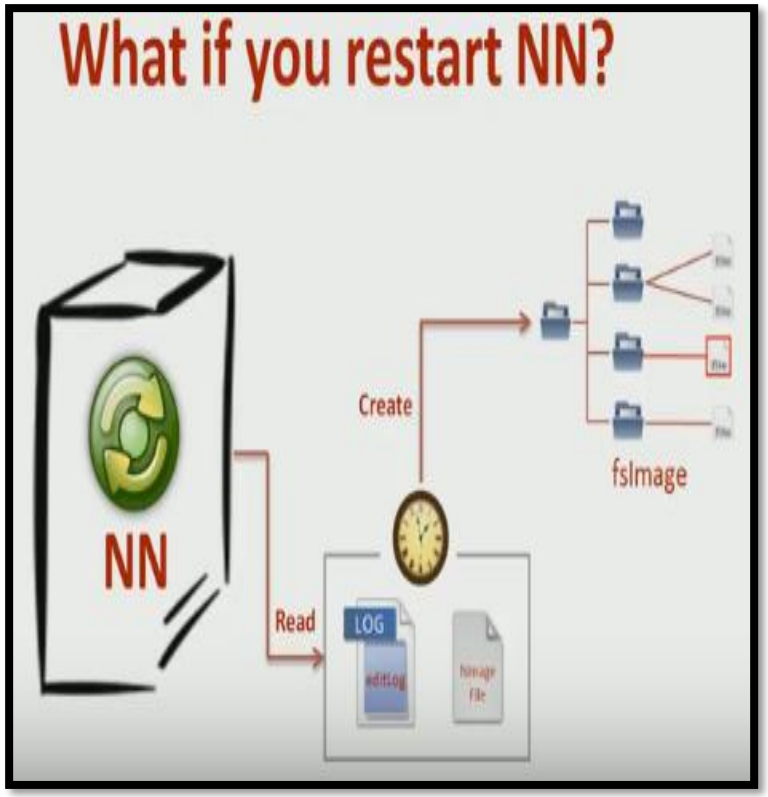
Secondary Name Node

- Secondary NN performs a **checkpoint activity every hour**.
- During the check point, the secondary NN will read the editlog and create the **latest file system image and save it to disk.**
- This state is exactly same as the In-memory fsImage. We call it on-disk fsimage.
- Once we have this on-disk fsImage the secondary NN truncate the edit log because all the changes are already applied, After an hour secondary NN will read the on-disk fsImage and apply all the changes from the editlog that we accumulated during the last 1 hour.
- It will replace the code on-disk fsImage with the new one and truncate the edit log once again. So the checkpoint activity is noting but the merging of an ondisk fsImage and editlog.



Secondary Name Node

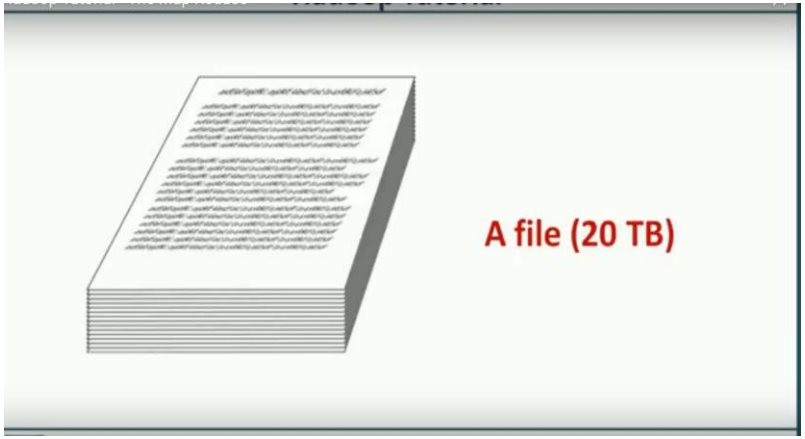
- The checkpoint doesn't take much time because we have just an hour old editlog to apply to the ondisk fsImage.
- The time to read fsImage is short because the fsImage is small compared to the editlog.
- The editlog is like a general ledger that records every transaction.
- The fsImage is like a balance sheet that shows the final state.
- So, the fsImage is small. In case of restart the name node also performs the same checkpoint activity that it can finish in a short time.
- Therefore the whole purpose of secondary NN and checkpoint is to minimize and restart time for the name node.



DATA LOCALITY

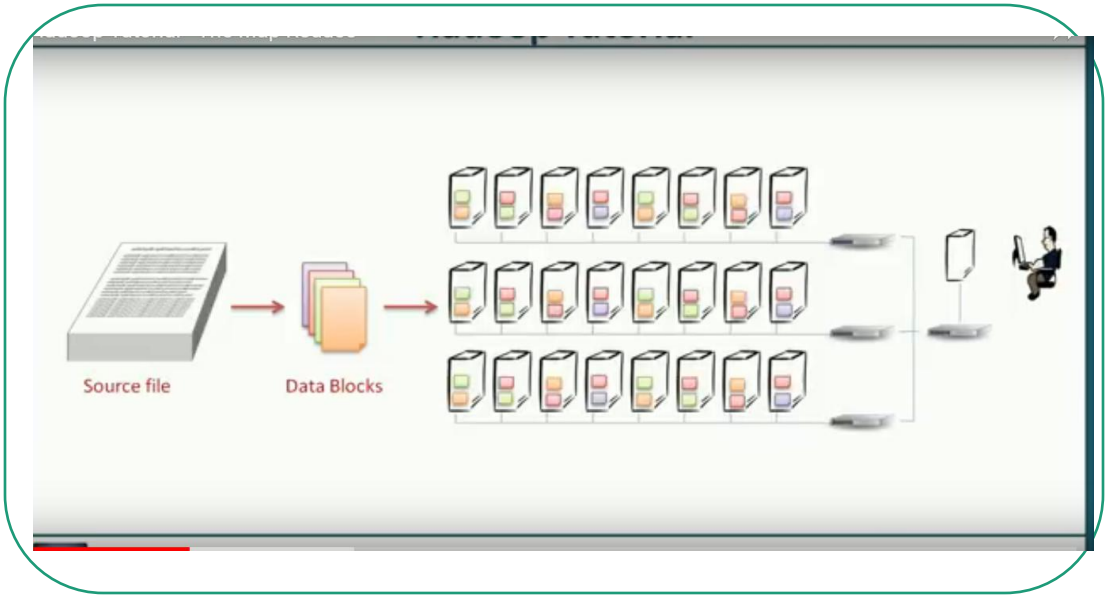
- In Hadoop, Data locality is the process of moving the computation close to where the actual data resides on the node, instead of moving large data to computation.

- This minimizes network congestion and increases the overall throughput of the system. .

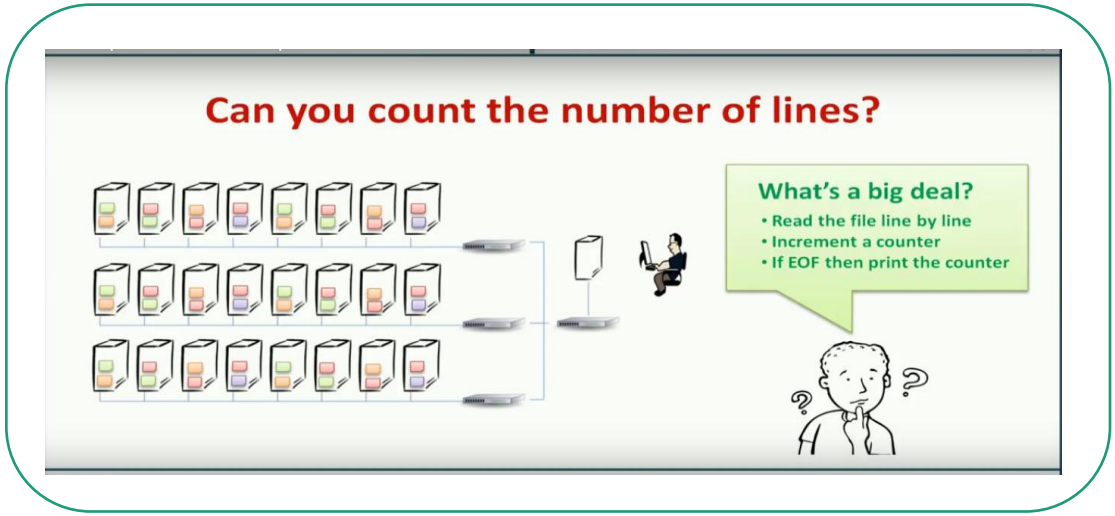


Lets us take a file of 20TB. We can't store the file in single machine .

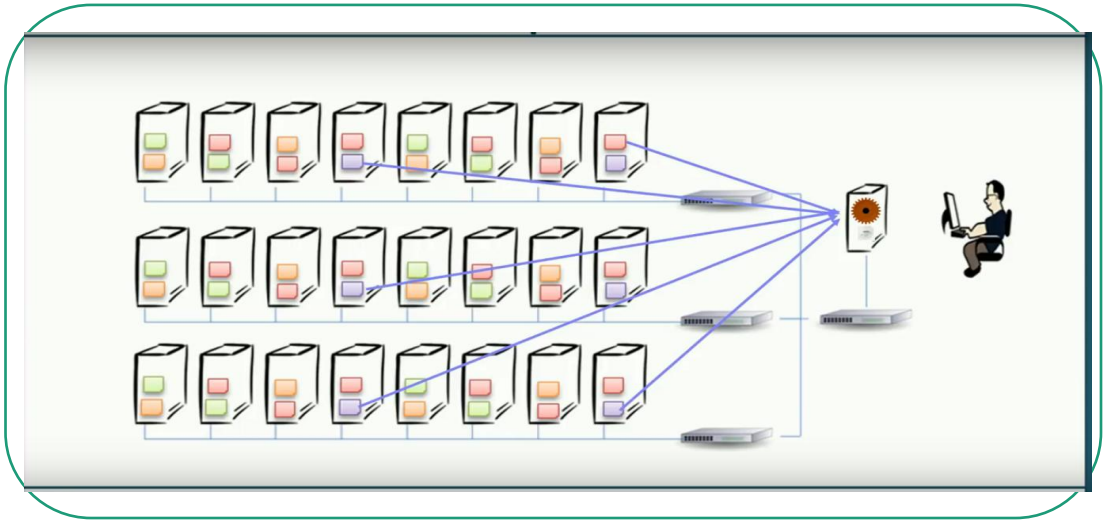
So we decided to store it in Hadoop Cluster



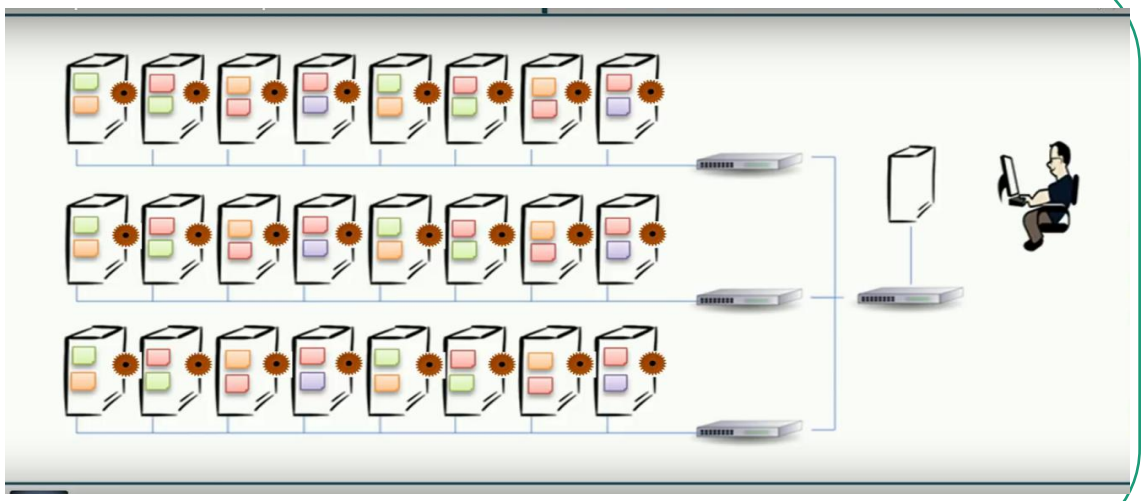
Divide the file into blocks . The Default Block size is 128 MB. Then store it in HDFS



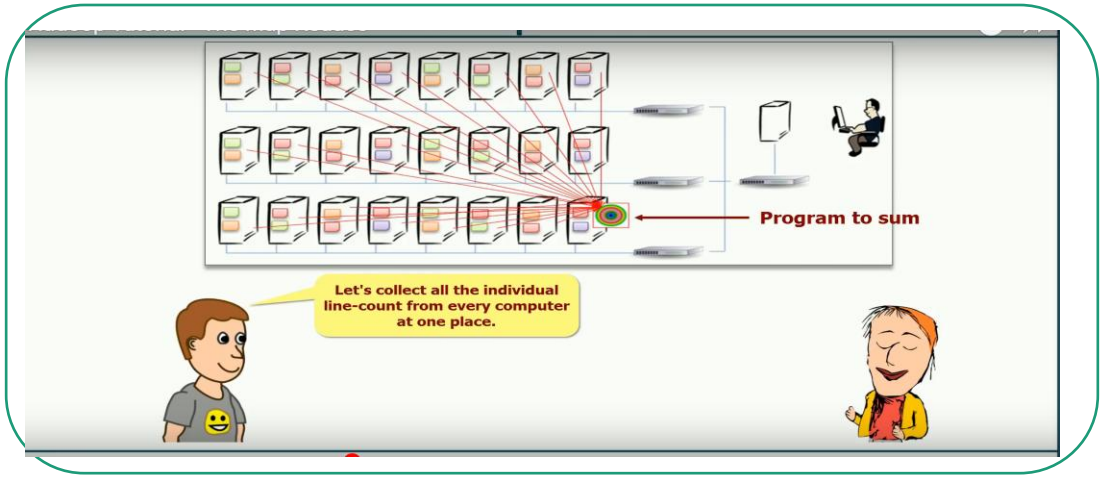
One way to count number of lines is to read a line and then increment the counter . Do the process until the EOF is Reached.



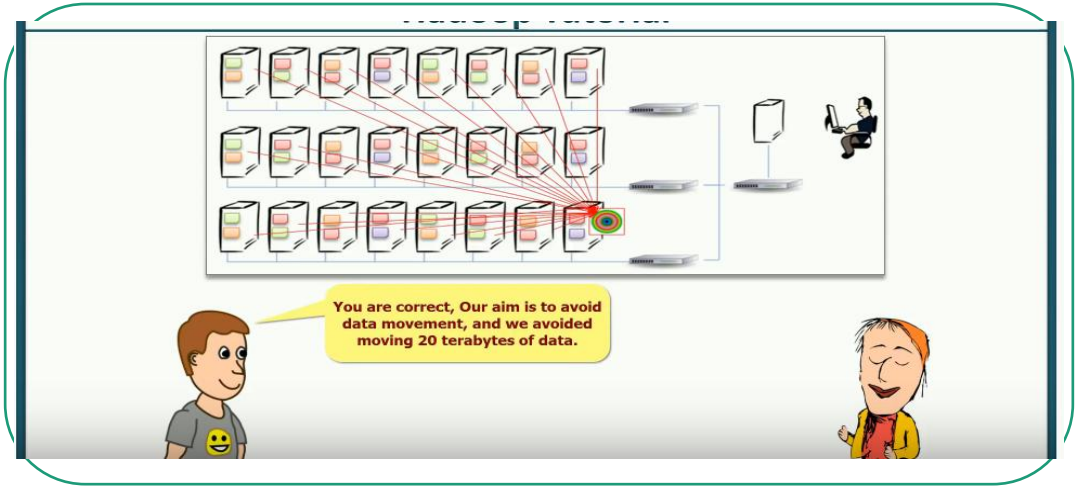
Bring all the Data to a single machine and execute the Java Program that counts the number of lines. This Process takes more Band width and more processing time .



Instead of Bringing data to processing , send processing to data and that is parallel processing



Collect all the Individual Line-count from every computer at one Place

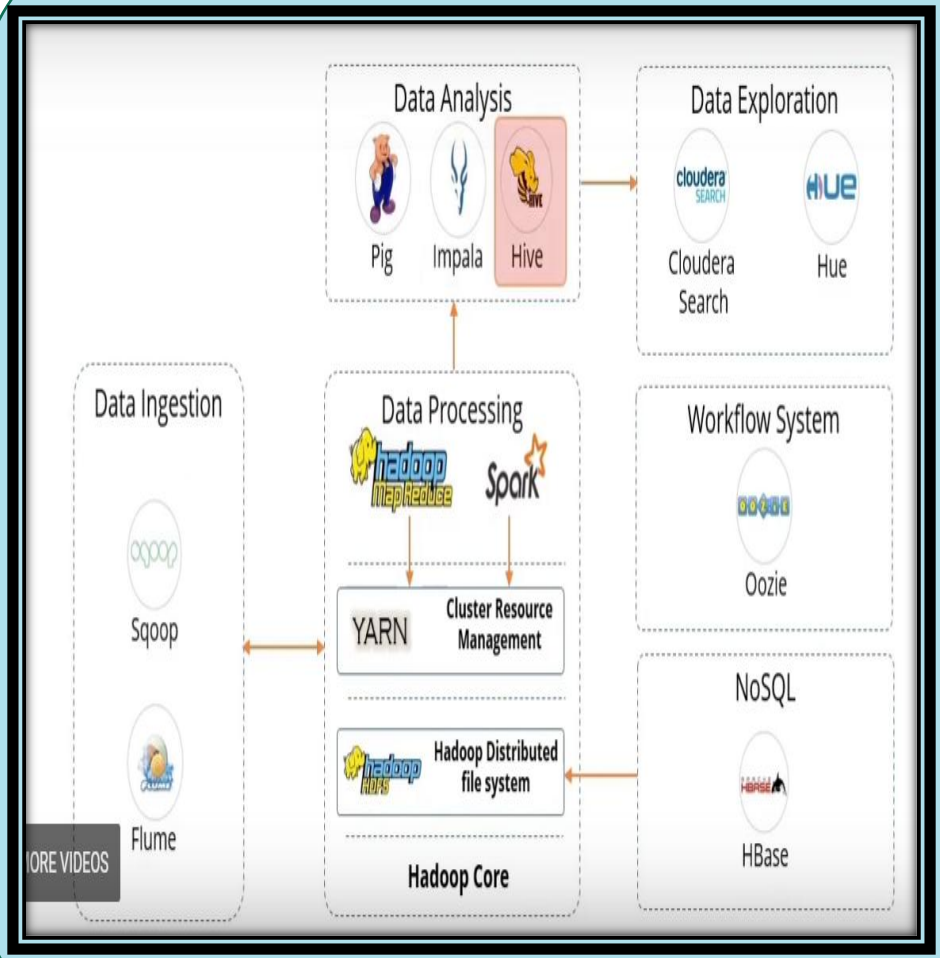


Our aim is to avoid Data Movement and has avoided moving 20 terabytes of Data

Apache Hadoop and Hadoop Ecosystem

- Hadoop is a Big Data Processing Tool (or)
- Hadoop is s a open source framework that allows to store and process big data in a distributed environment across clusters of computer using simple programming models
- Two major components of Hadoop are
    1) **Hadoop Distributed File System** **(For Distributed Storage)**
    2) **Map Reduce** **(For Parallel Processing)**
- Besides these two components, many components have been introduced to deal with Big Data
- And these components are collectively called as Hadoop Ecosystem

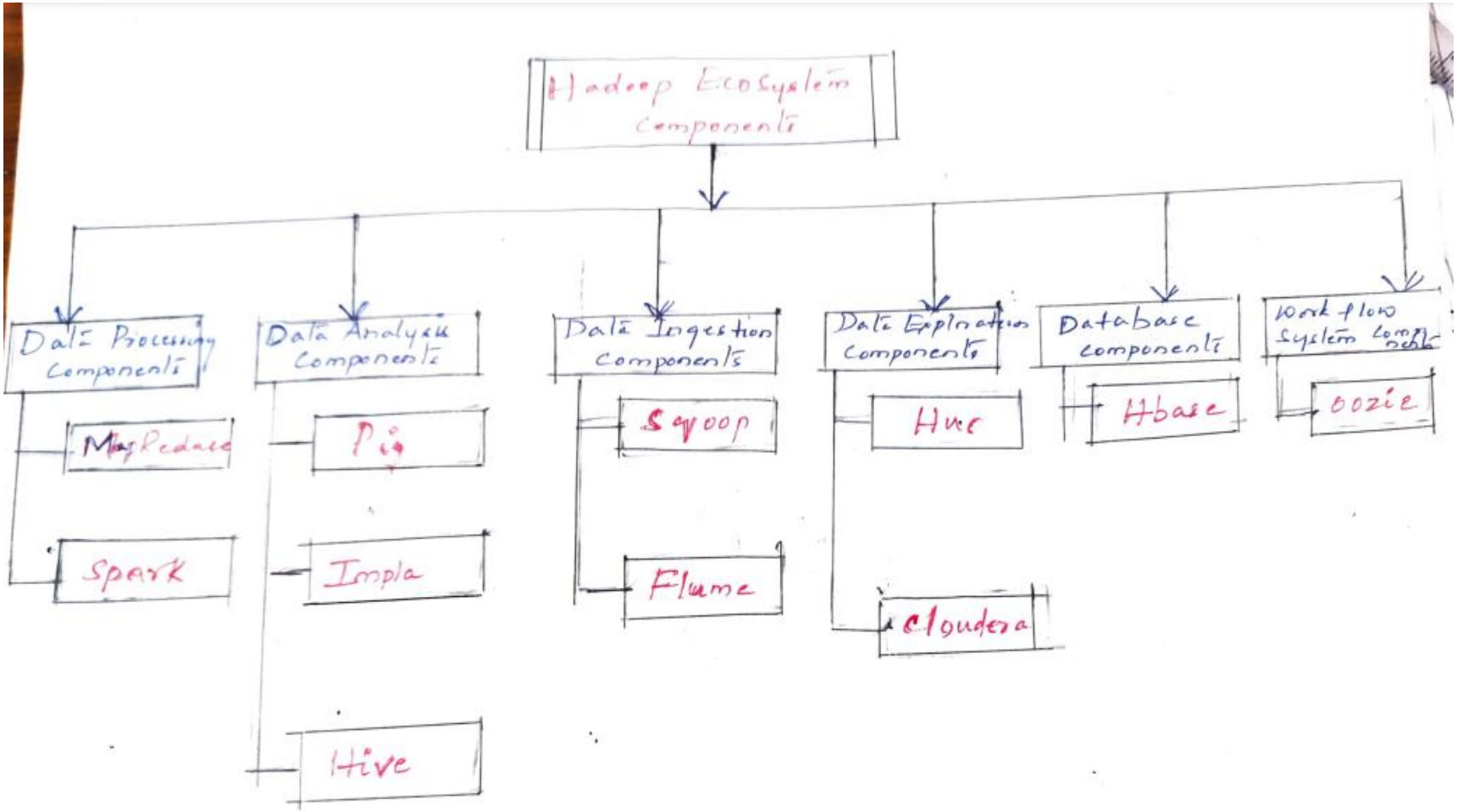
Apache Hadoop and Hadoop Ecosystem



Hadoop Ecosystem components are categorized into

1) Data Processing Components
2) **Data Analysis Components**
3) Data Ingestion Components
4) **Data Exploration Components**
5) Data Base Components
6) **Work flow system Components**

Apache Hadoop and Hadoop Ecosystem



Apache Hadoop and Hadoop Ecosystem

1) Data Processing Components

1.1) Map Reduce

- ✓ It is a parallel programming model for processing large amounts of structured, semi-structured, and unstructured data on large clusters of commodity hardware.

1.2) Spark

- ✓ Spark is an open-source distributed engine for processing and analzing huge volumes of real time data
- ✓ Provides 100 times faster performance as compared to MapReduce
- ✓ Provides in-memory computation of data
- ✓ Used to process and analyze real time streaming data such as stock market and banking data
- ✓ Supports Machine Learning, Business Intelligence, Streaming and Batch Processing

Apache Hadoop and Hadoop Ecosystem

2) Data Analysis Components

2.1) Pig

- ✓ Pig is used to analyze data in Hadoop.
- ✓ It Provides a high level data processing language to perform numerous operations on the data.
- ✓ Language used in Pig is Pig Latin
- ✓ Pig Includes Pig Latin, a scripting language.
- ✓ Pig translates Pig Latin Scripts into MapReduce, which can ten process data in the HDFS cluster.
- ✓ Less Number of code is required to write in Pig when compared to MapReduce i.e 10 lines of pig latin script is around 200 lines of map Reduce job

2.2) Impla

- ✓ Impla is the highest performing SQL Engine which provides the fastest way to access data that I stored in Hadoop Cluster.
- ✓ Impla is Ideal for Interactive Analysis
- ✓ The Latency is very low and it is in measured in milli seconds

2.3) Hive

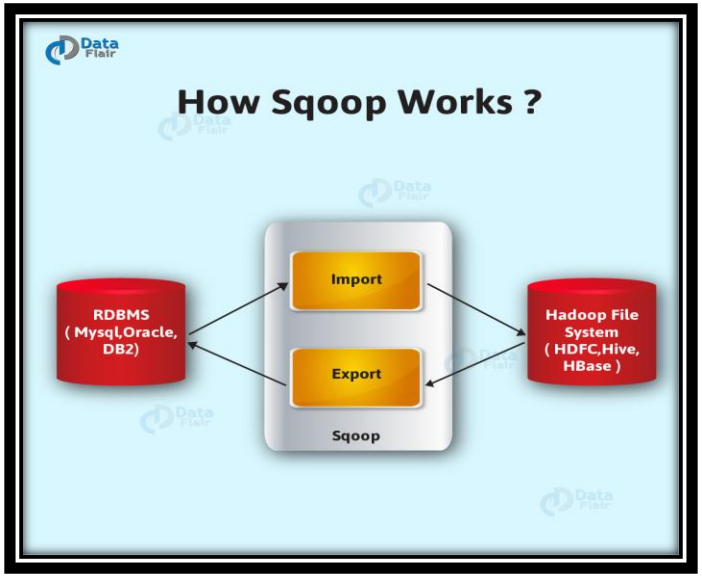
- ✓ The Hadoop ecosystem component, Apache Hive, is an open source data warehouse system for querying and analyzing large datasets stored in Hadoop files. (acts like a OLTP Tool)
- ✓ Hive do three main functions: *data summarization, query, and analysis*.
- ✓ Hive use language called HiveQL (HQL), which is similar to SQL. HiveQL automatically translates SQL-like queries into MapReduce jobs which will execute on Hadoop

Apache Hadoop and Hadoop Ecosystem

3) **Data Ingestion**

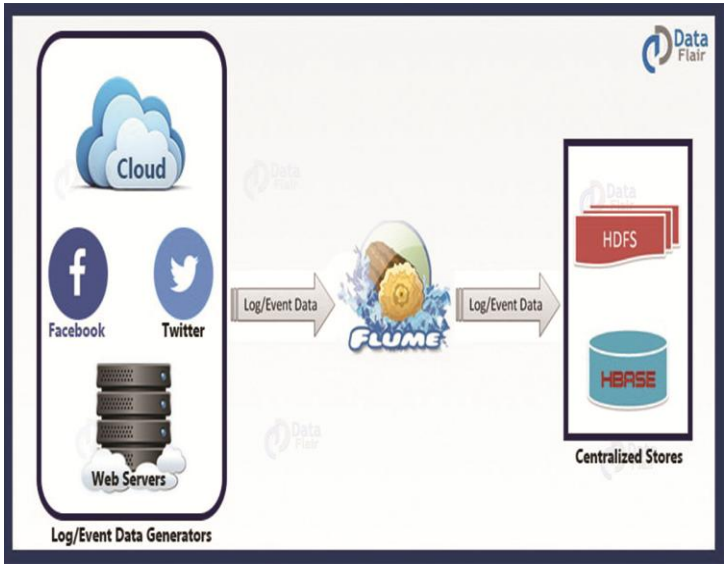
3.1) Sqoop

- ✓ Sqoop is a tool designed to transfer data between Hadoop and Relational Database Servers
- ✓ It is used to import data from relational databases such as Oracle, MYSQL to HDFS and Export data from HDFS to Relational databases



3.2) Flume

- ✓ To Ingests online streaming data from social media, log files, web server into HDFS
- ✓ **Flume** efficiently collects, aggregate and moves a large amount of data from its origin and sending it back to HDFS.

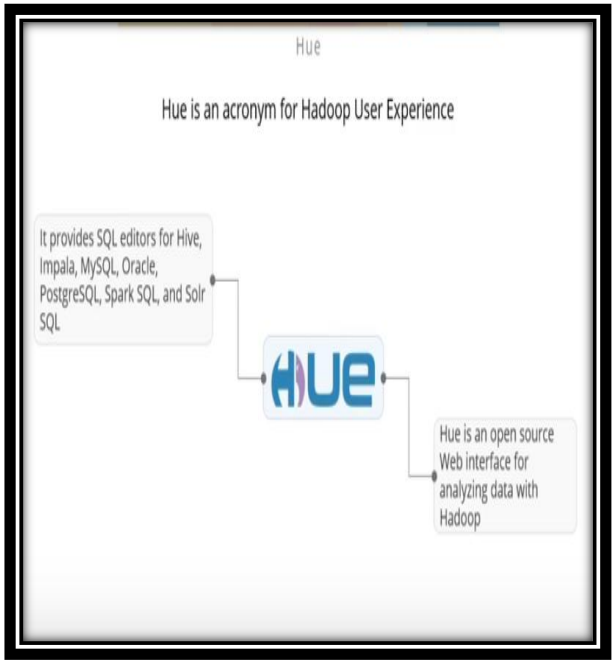


Apache Hadoop and Hadoop Ecosystem

4)Data Exploration

4.1) Hue

- ✓ Hue is an Acronym for Hadoop User Experience
- ✓ It provides SQL Editors fro Hive, Impla, MySQL, Oracle, PostgreSQL etc..
- ✓ Hue is an Open source web interface for analyzing data with Hadoop



4.2) Cloudera

- ✓ One of cloudera's near real time access products
- ✓ Users do not need SQL or Programming skills to use Cloudera Search
- ✓ Enables non-technical users to search and explore data stored in or ingested into Hadoop and HBase.

Apache Hadoop and Hadoop Ecosystem

5) Data Base Components

5.1) Hbase

- ✓ Stores data in HDFS
- ✓ A NoSQL database or Non-relational database
- ✓ Mainly used when you need random. Real-time, read/write access to your Big Data.
- ✓ Provides support to high volume of data and high throughput

Limitations of Hadoop

- ❑ Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one has to search the entire dataset even for the simplest of jobs.
- ❑ A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of time (random access).

Hadoop Random Access Databases

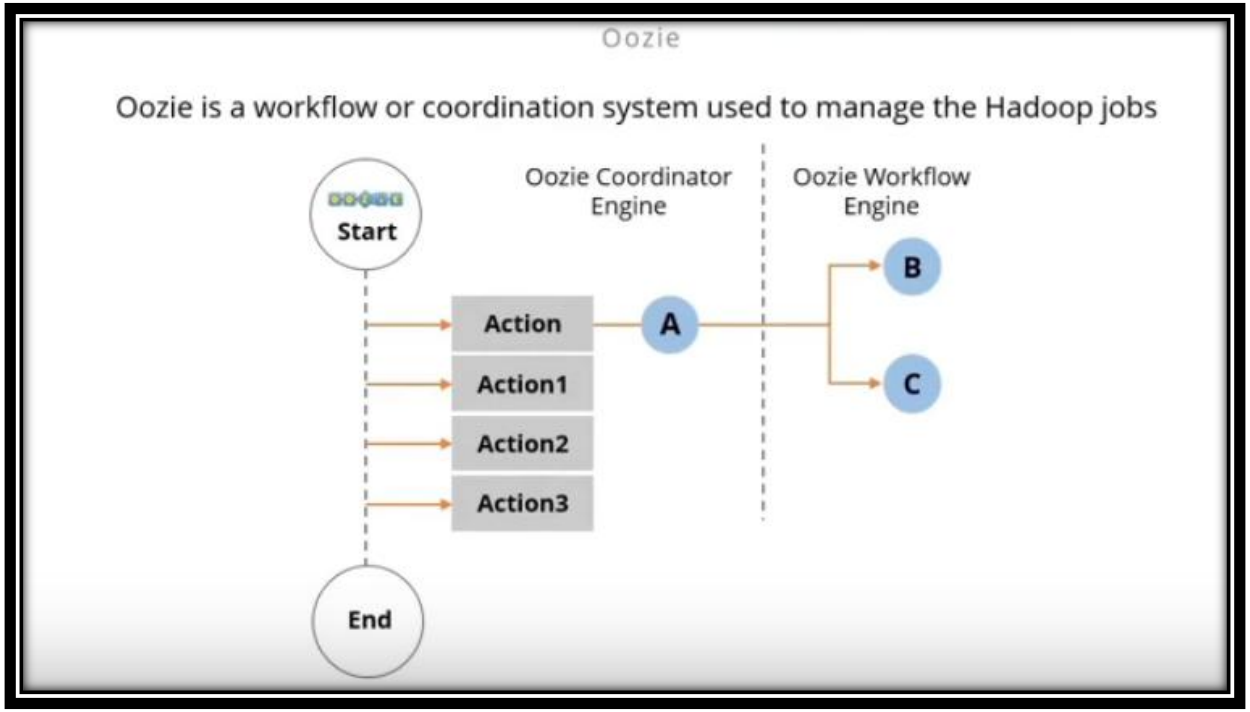
- ❑ Hbase stores huge amounts of data and access the data in a random manner.
- ❑ HBase is a distributed column-oriented database built on top of the Hadoop file system.
- ❑ One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.

Apache Hadoop and Hadoop Ecosystem

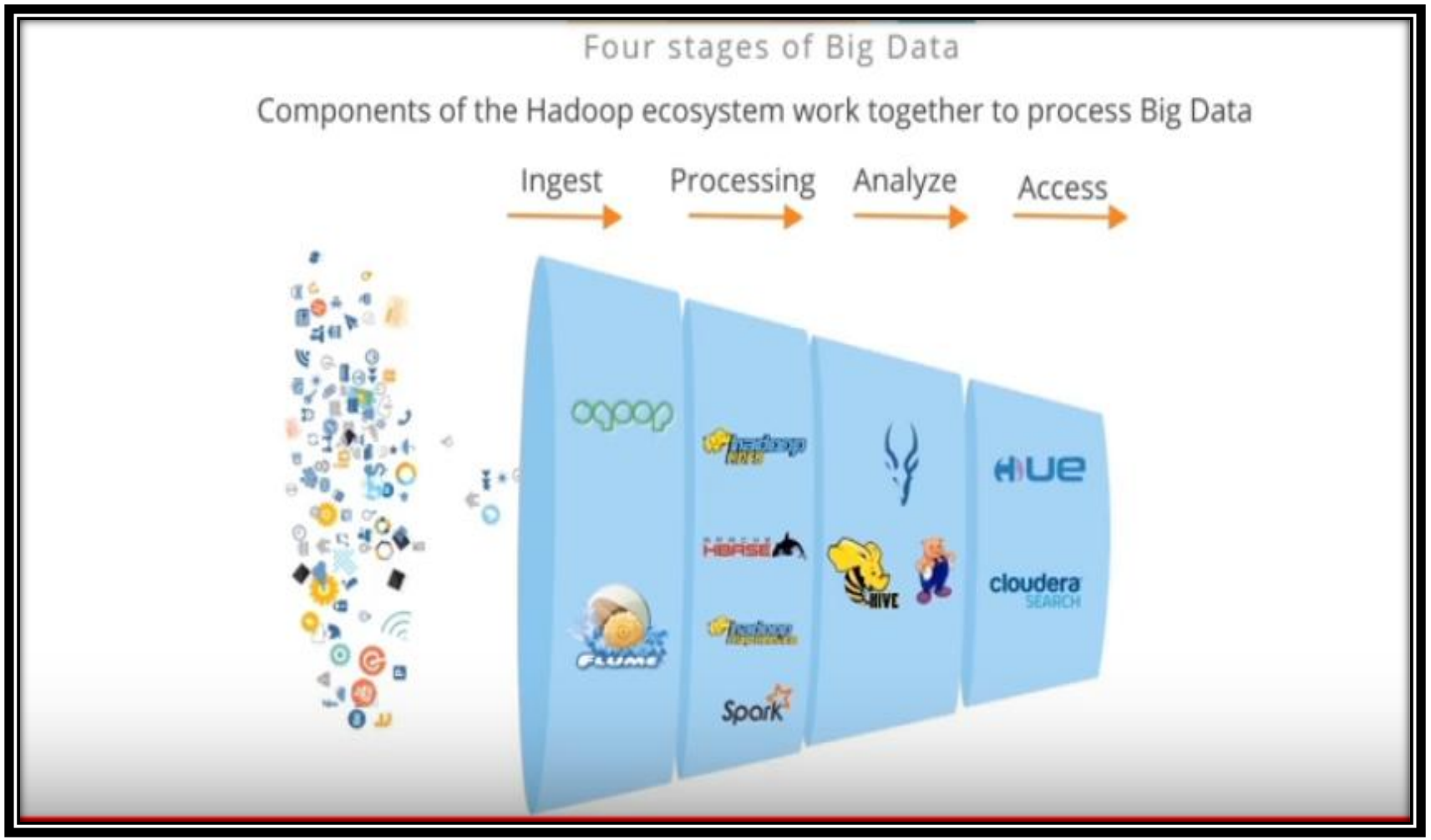
6) Work Fow System Components

6.1) Oozie

✓ Oozie is a workflow scheduler system to manage Hadoop Jobs



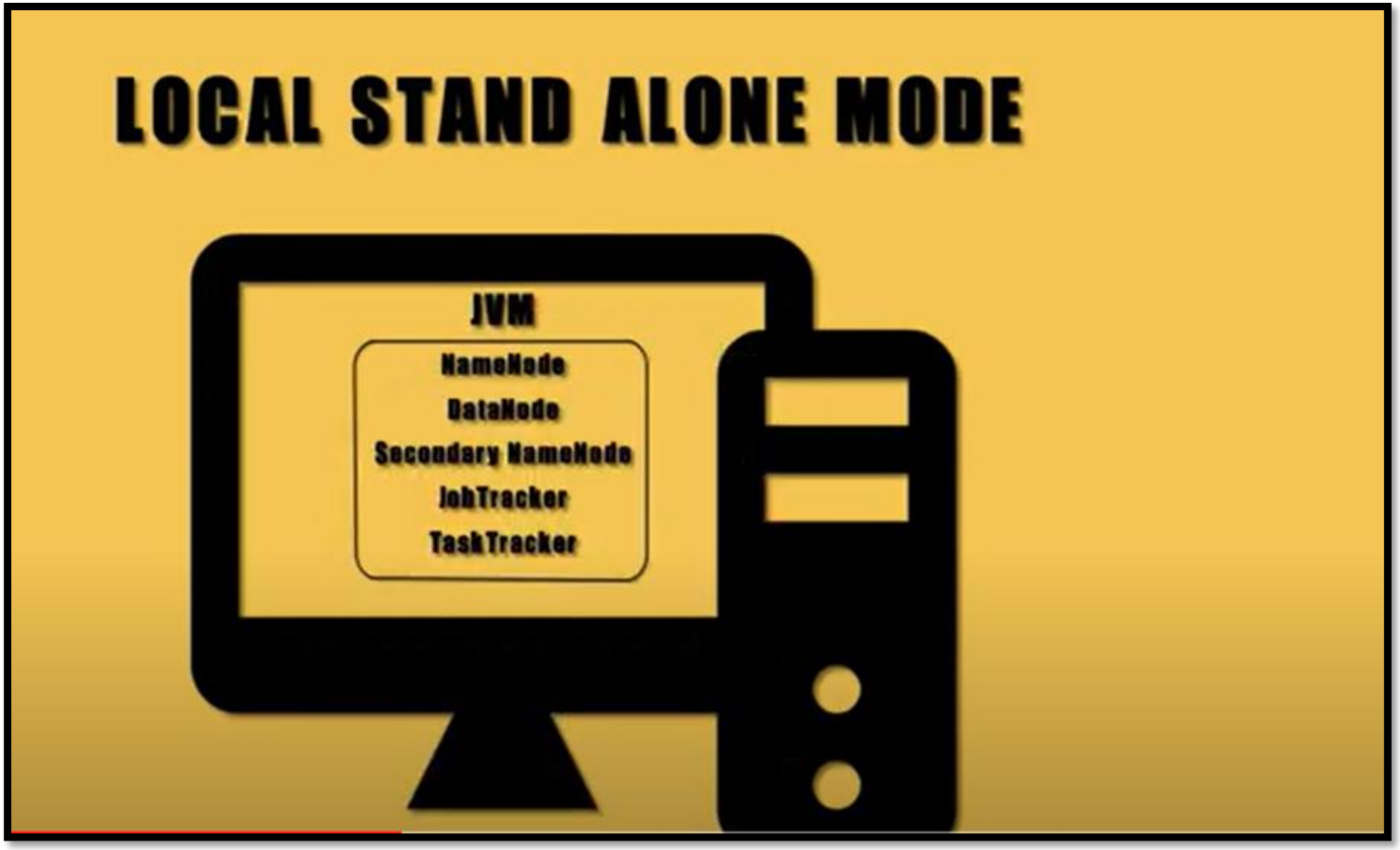
Apache Hadoop and Hadoop Ecosystem



7) Understanding different Hadoop modes

- Hadoop can run in three different modes as below-
  - ✓ Local (Standalone) Mode.
  - ✓ Pseudo-Distributed Mode
  - ✓ Fully-Distributed Mode

All the Components of HDFS like Name Node, Data Node, Job Tracker, Task Tracker in Single JVM



1. Local Mode or Standalone Mode

- Standalone mode is the default mode in which Hadoop run.
- Standalone mode is mainly used for debugging where you don't really use HDFS.
- You can use input and output both as a local file system in standalone mode.
- You also don't need to do any custom configuration in the files- **mapred-site.xml, core-site.xml, hdfs-site.xml**.
- Standalone mode is usually the fastest Hadoop modes as it uses the local file system for all the input and output.

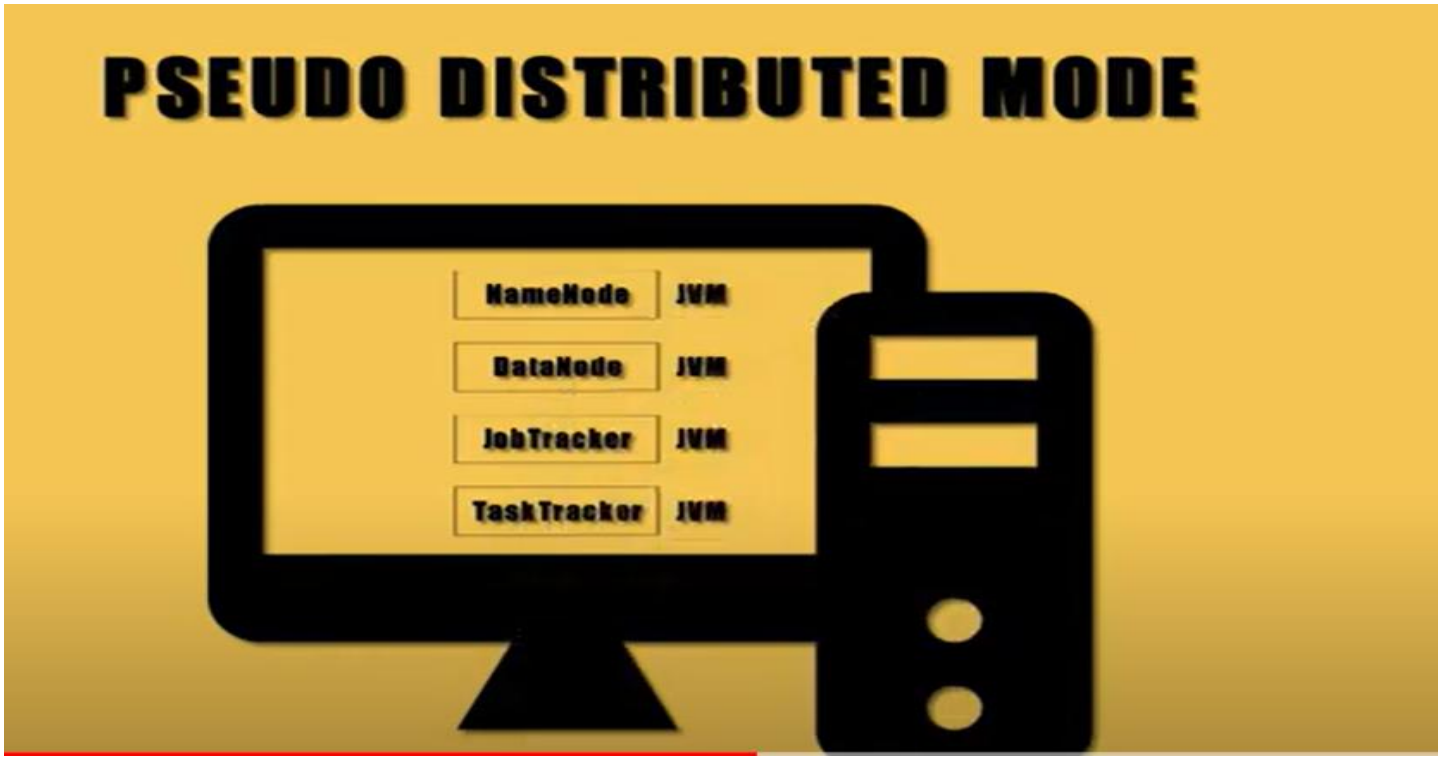
- Here is the summarized view of the standalone mode
  - ✓ Used for debugging purpose
  - ✓ HDFS is not being used
  - ✓ Uses local file system for input and output
  - ✓ No need to change any configuration files
  - ✓ Default Hadoop Modes

2) Pseudo-distributed Mode

- The **pseudo-distribute mode** is also known as a **single-node cluster** where both NameNode and DataNode will reside on the same machine.
- In pseudo-distributed mode, all the Hadoop daemons will be running on a single node.
- Such configuration is mainly used while testing when we don't need to think about the resources and other users sharing the resource.

- Here is the summarized view of pseudo distributed Mode-
  - ✓ Single Node Hadoop deployment running on Hadoop is considered as pseudo distributed mode
    All the master & slave daemons will be running on the same node.
  - ✓ Mainly used for testing purpose.
  - ✓ Replication Factor will be ONE for blocks.
  - ✓ Changes in configuration files will be required for all the three files- **mapred-site.**xml**, core-site.**xml**,** hdfs**-site.xml**

The Java Virtual Machine is called JVM, is an **abstract computing machine** or **virtual machine interface** that drives the java code.



Each Component of HDFS like Name Node, Data Node, Job Tracker and Task Tracker will be running in separate a JVM of the single Machine

3) Fully-Distributed Mode (Multi-Node Cluster)

- This is the **production mode of Hadoop** where multiple nodes will be running. Here data will be distributed across several nodes and processing will be done on each node.
- Master and Slave services will be running on the separate nodes in fully-distributed Hadoop Mode.
  - ✓ Production phase of Hadoop
  - ✓ Separate nodes for master and slave daemons
  - ✓ Data are used and distributed across multiple nodes

FULLY DISTRIBUTED MODE

Conclusion

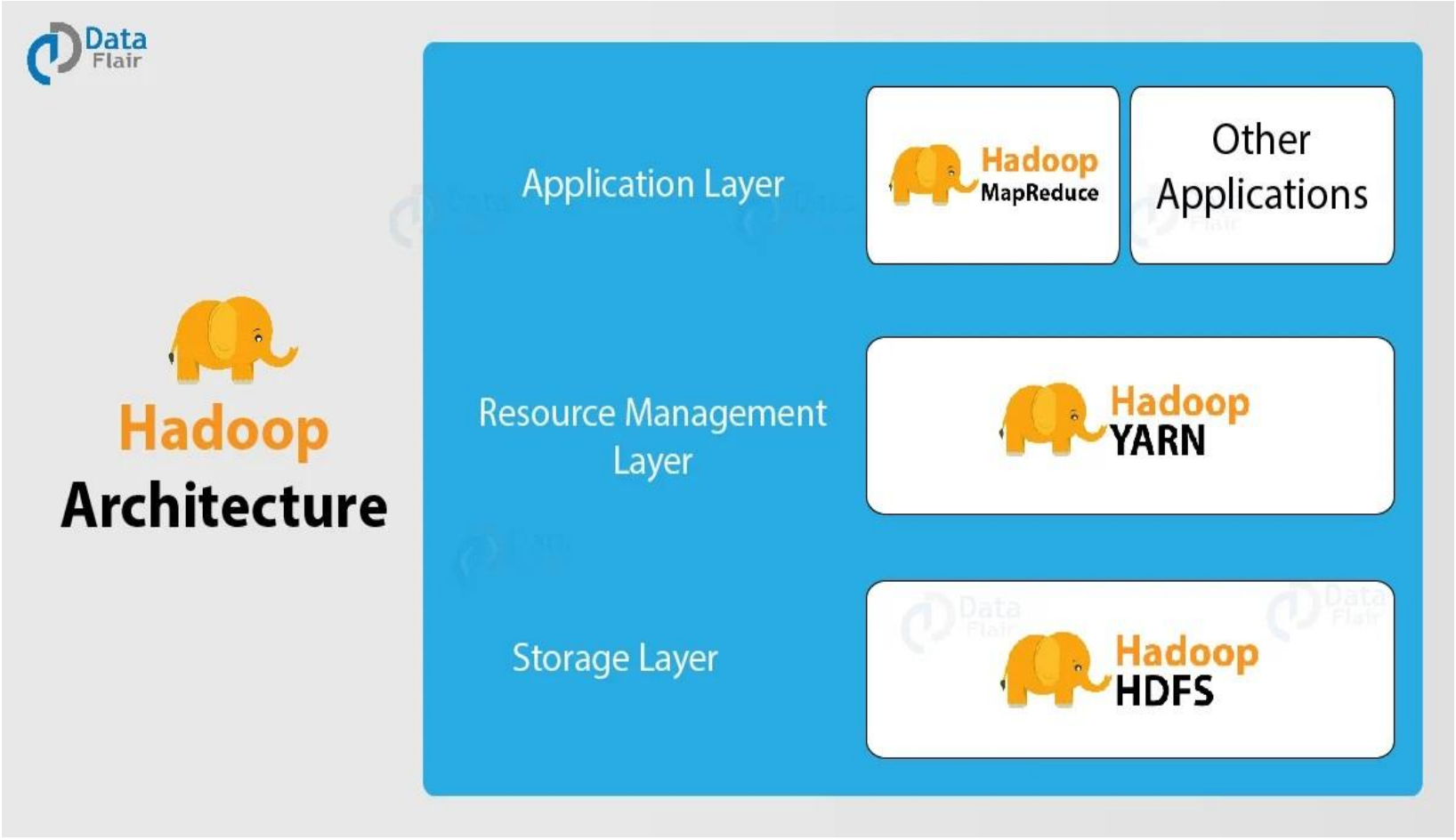
- Single mode runs a single process on one system and is not distributed. Pseudo-mode also runs on one system but it creates a cluster simulation.
- Single mode does not use hdfs, it used the local filesystem instead. But in pseudo-mode, hdfs is used. This how to storage and file system differs between these two modes.
- Pseudo mode runs virtual nodes on the same system. In single mode, only one node is run and this mode is mainly used for debugging process.

Hadoop Architecture

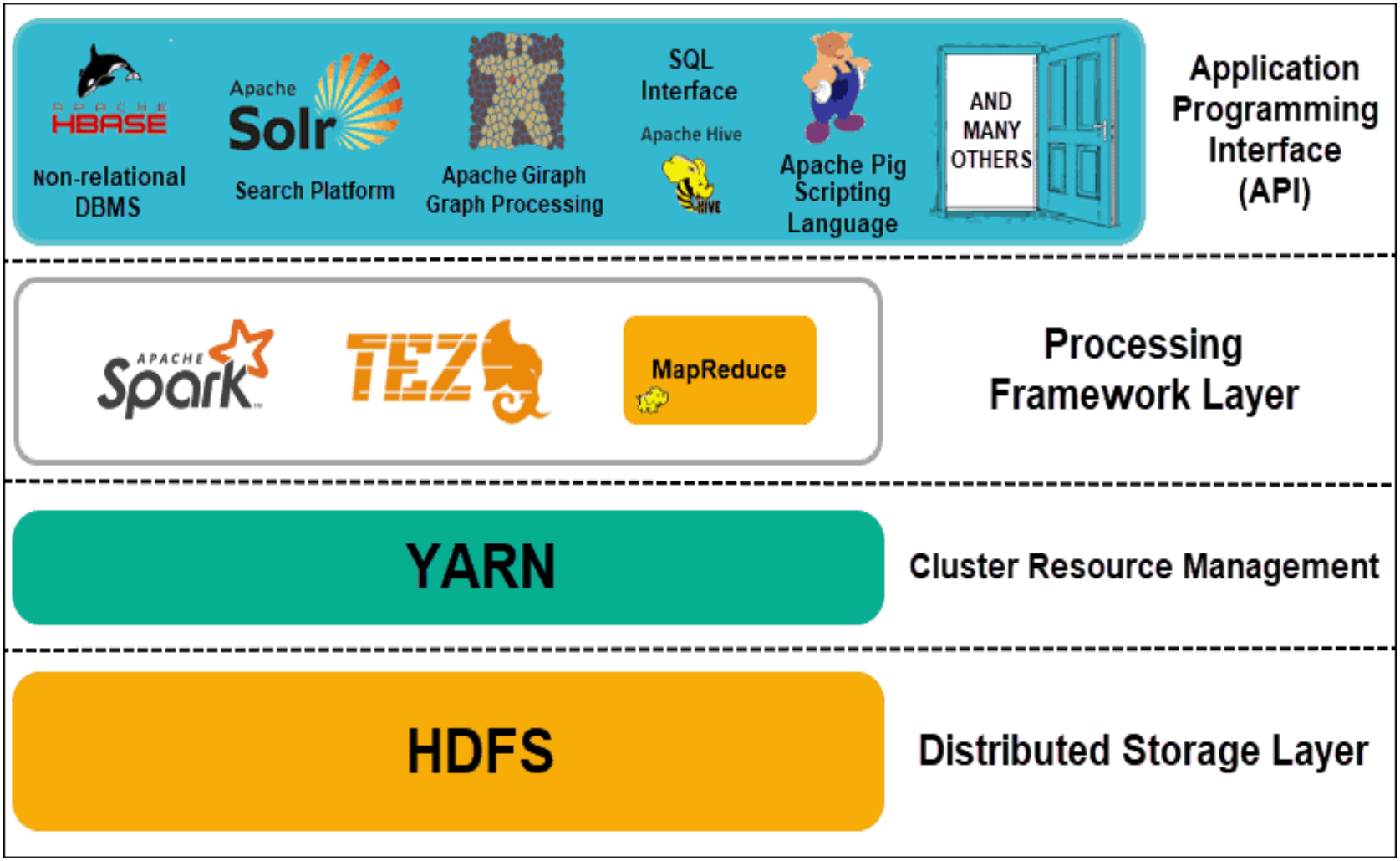
- Hadoop is a software framework developed by the Apache Software Foundation for distributed storage and processing of huge amounts of datasets.

- ***"Hadoop is a technology to store massive datasets on a cluster of cheap machines in a distributed manner".*** It was originated by Doug Cutting and Mike Cafarella.

Hadoop Architecture



Hadoop Architecture



Hadoop Components

Hadoop Consists Of 3 Core Components :

1. HDFS (Hadoop Distributed File System)
2. MapReduce
3. Yarn (Yet Another Resource Negotiator)

1. HDFS (Hadoop Distributed File System)

- It is the **storage layer of Hadoop**.
- Files in HDFS are broken into block-sized chunks.
- HDFS consists of two types of nodes that is,
  - i. NameNode
  - ii. DataNodes.
- NameNode stores metadata about blocks location.
- DataNodes stores the block and sends block reports to namenode in a definite time interval.

2. MapReduce

- MapReduce is a parallel programming model for writing distributed applications at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters.

3. YARN

- It is the Resource Management Layer.
- YARN is responsible for Resource Allocation and Job Scheduling.
- To study in detail Hadoop and its component, go through the **Hadoop Architecture** article.

Hadoop Distributed File System

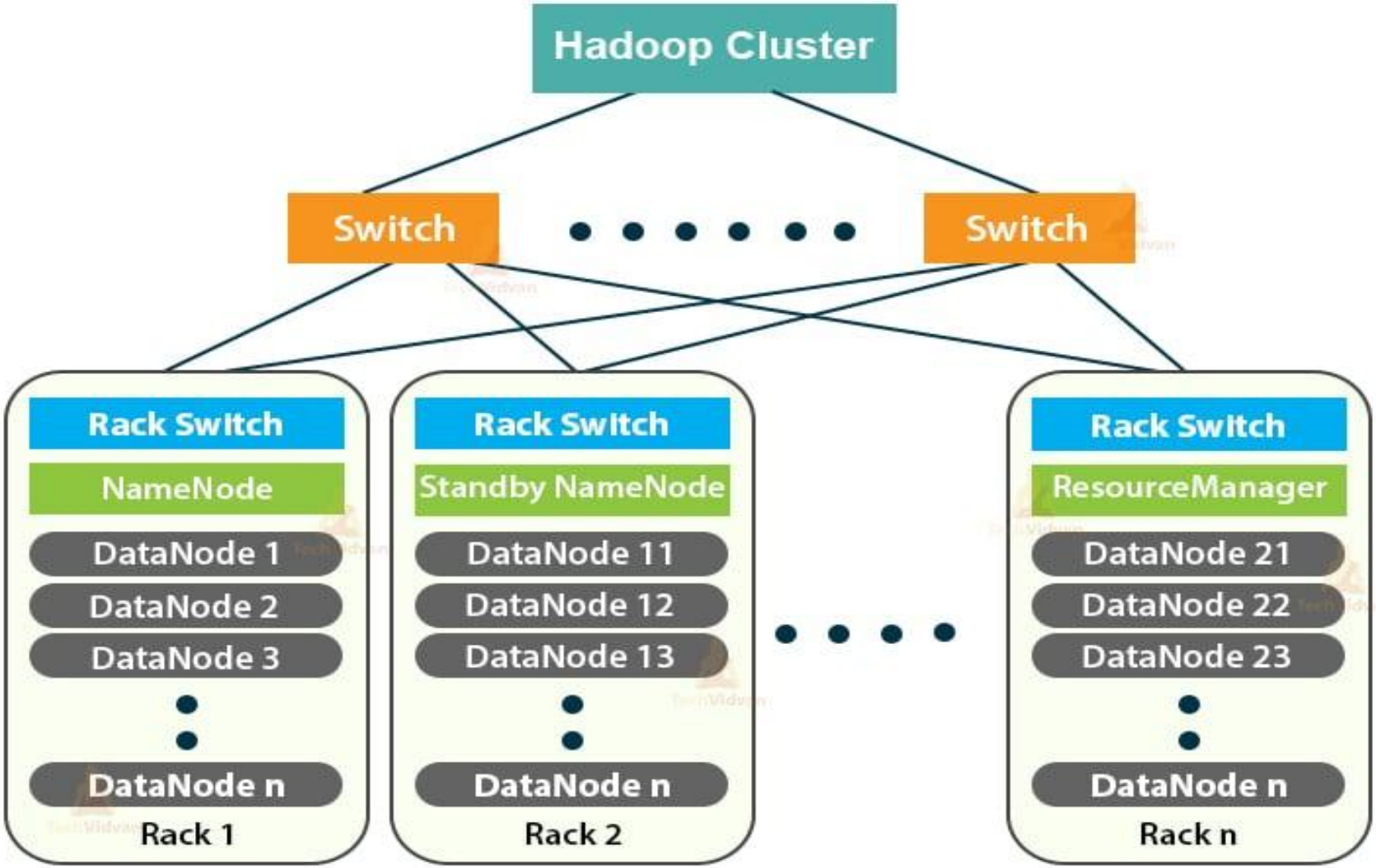
- HDFS is a distributed file system designed to run on commodity hardware.
- HDFS is like Master worker Architecture. The master is a NameNode and Slave is DataNode.

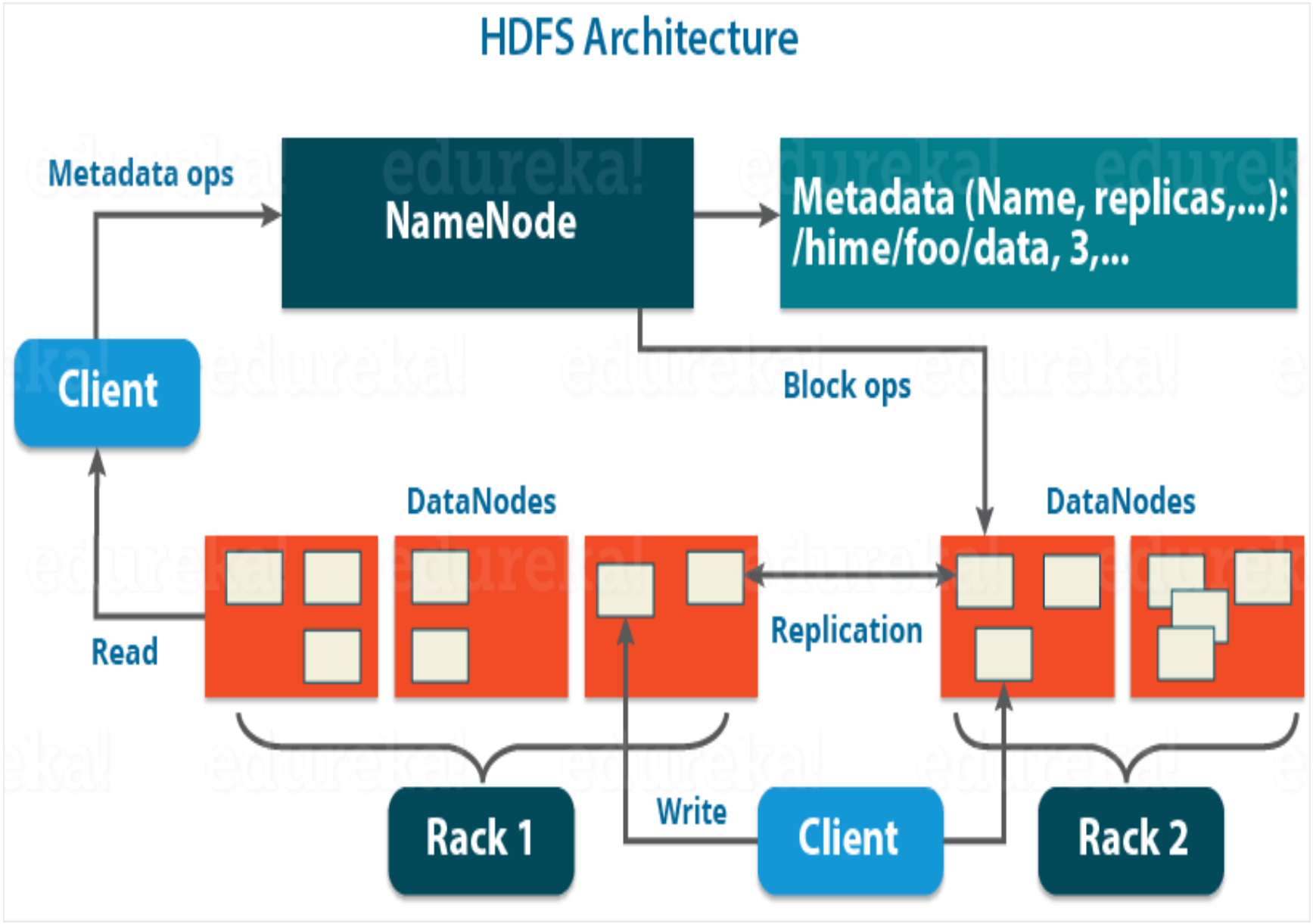
- HDFS is highly Fault-Tolerant and is designed to be deployed on Low-Cost Hardware.

Hadoop Distributed File System

- HDFS provides high Throughput access to application data and is suitable for applications that have large data sets.
- HDFS is Suitable for applications with large data sets.
- HDFS was originally built as infrastructure for the Apache Nutch web search engine project.

HDFS Architecture




HDFS Architecture
Metadata ops
NameNode
Metadata (Name, replicas,...):
/hime/foo/data, 3,...
Client
Block ops
DataNodes
DataNodes
Read
Replication
Rack 1
Write
Client
Rack 2

HDFS Services

1. Name Node
2. Secondary Name Node
3. Job tracker
4. Data Node
5. Task Tracker

NameNode

- Metadata in Memory
  - The entire metadata is in main memory
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DataNode

- DataNodes are the main storages of data. Hadoop uses Low-Cost hardware to Store data.
- DataNodes are responsible for Storing, Replication Creating, Deleting these type of jobs according to the instruction of NameNode.
- These DataNodes send the health report to the NameNode periodically.
- The defalult time is 3second. So after every second these send the report to the NameNode.

Secondary NameNode

- SNN is another specially dedicated node, which is used to take the checkpoints of the File-System.

- The SNN is not substitute of the primary NameNode.

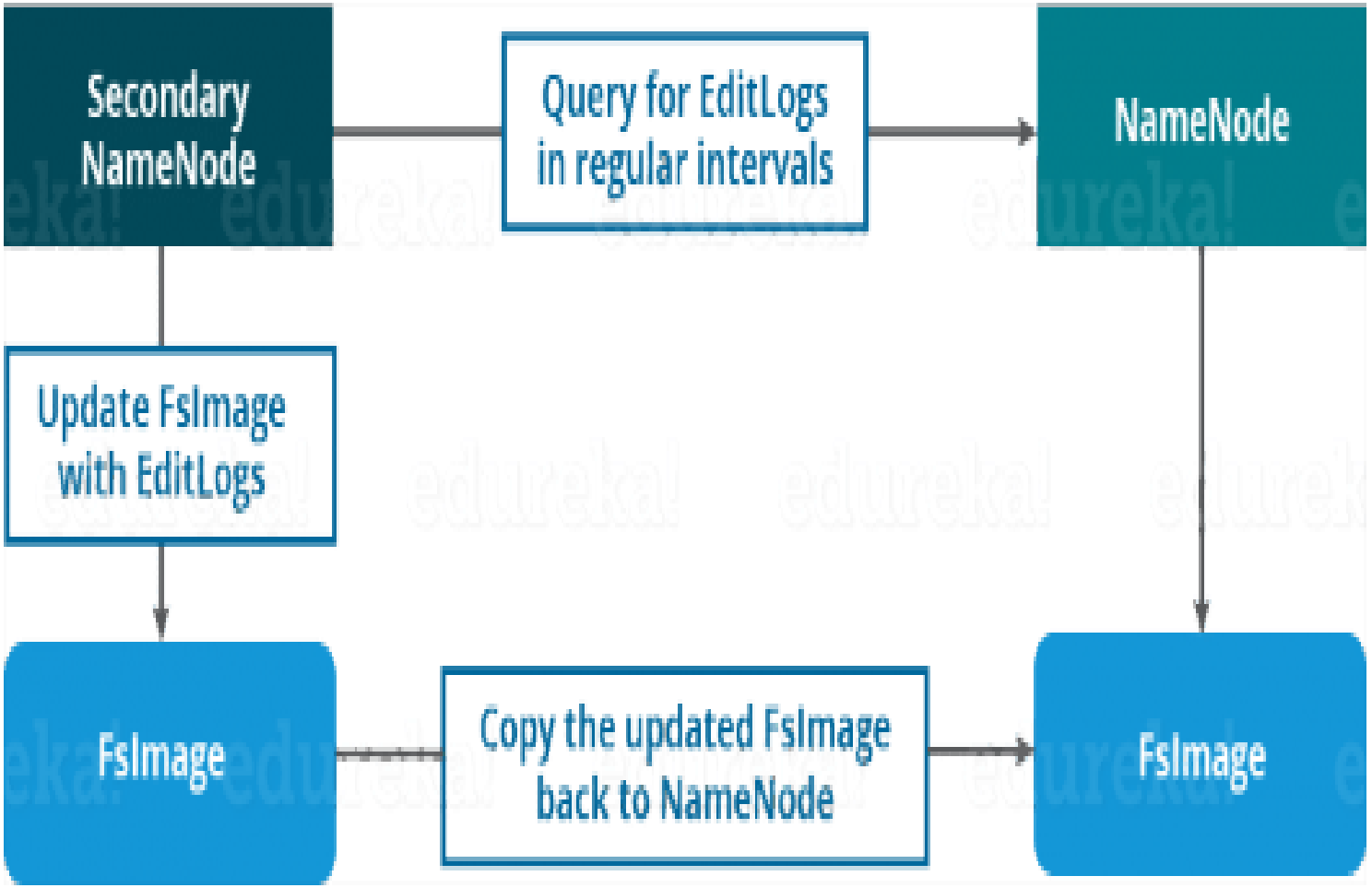
  It helps the NameNode but not replace of NameNode.

- Copies FsImage and Transaction Log from Namenode to a temporary directory

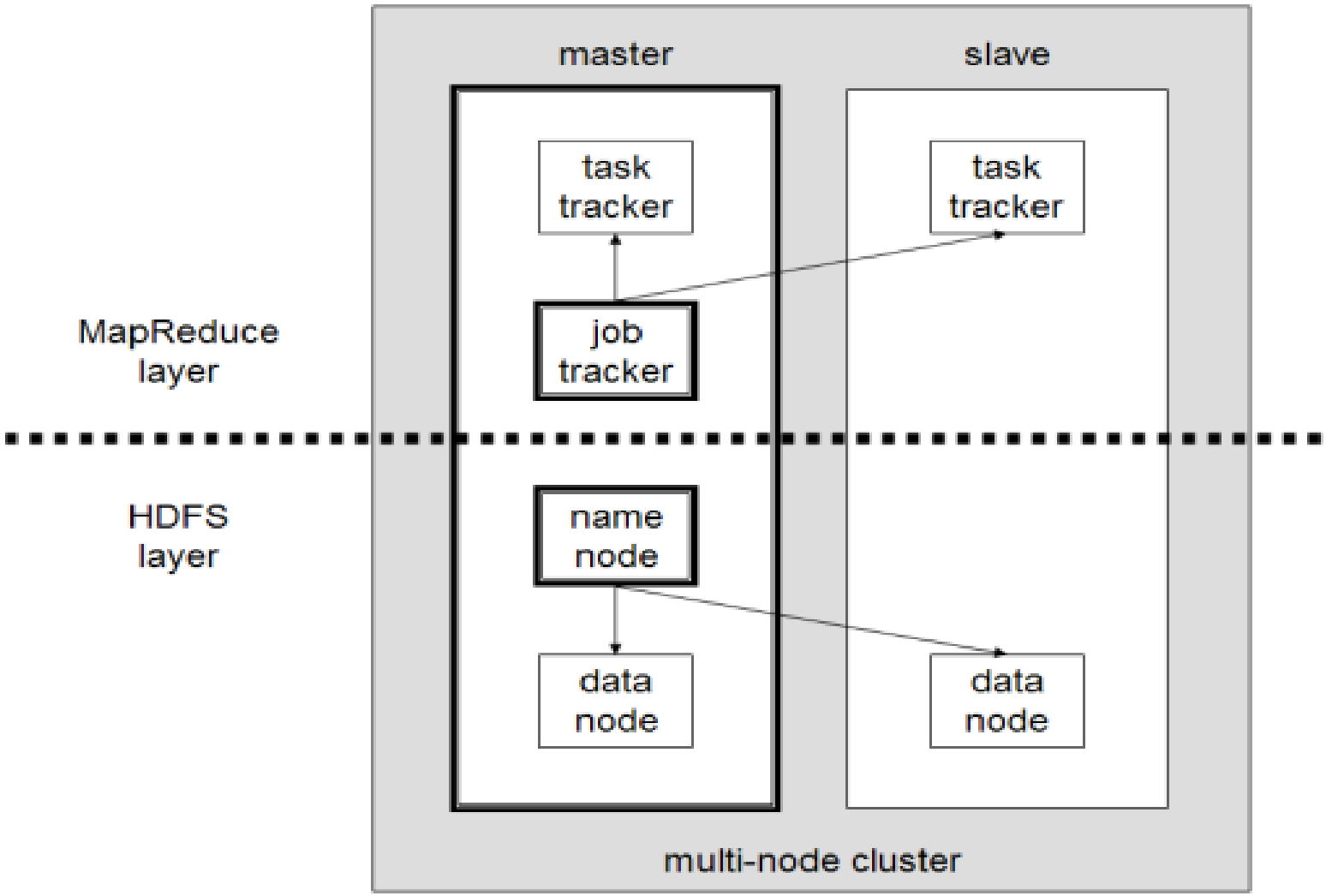
- Merges FSImage and Transaction Log into a new FSImage in temporary directory

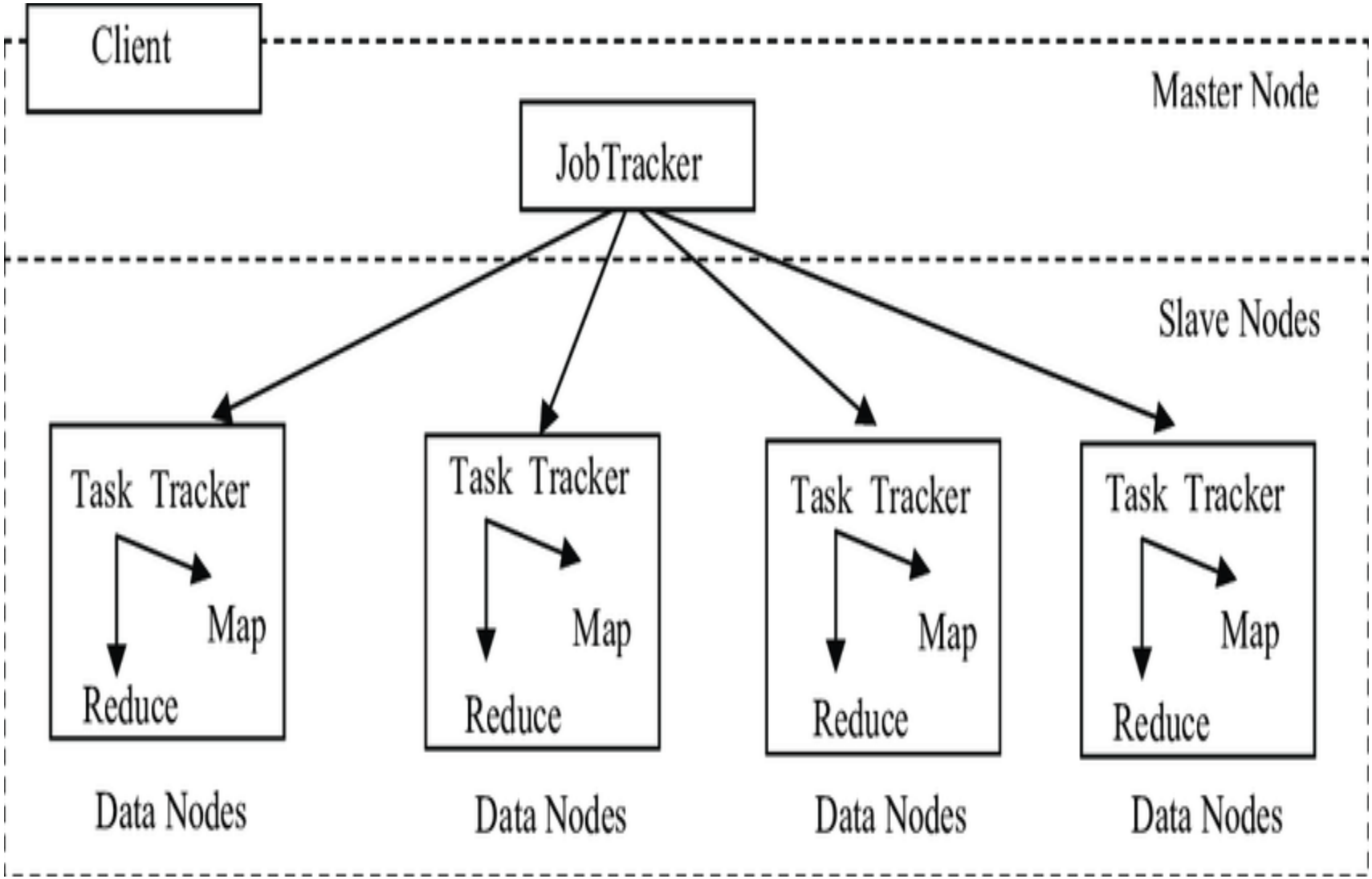
- Uploads new FSImage to the NameNode
  - Fsimage is contains Dircetory Structure (NameSpace)

Secondary NameNode



Job Tracker and Task Tracker




Client
Master Node
JobTracker
Slave Nodes
Task Tracker
Map
Reduce
Data Nodes
Task Tracker
Map
Reduce
Data Nodes
Task Tracker
Map
Reduce
Data Nodes
Task Tracker
Map
Reduce
Data Nodes

Job Tracker:

- Basically Job Tracker will be useful in the Processing the data.
- Job Tracker receives the requests from the client for MapReduce execution.
- Job Tracker talks with Name node to know about the location of the data like Job Tracker will request the Name Node for the processing the data.
- Name node in response gives the Metadata to job tracker.

Task Tracker

- It is the Slave Node for the Job Tracker and it will take the task from the Job Tracker.

- Task Tracker receives code from the Job Tracker and apply the code on the file for the process is known as Mapper.

Design Criteria of HDFS

Very Large Files
- Petabyte-Scale data

Lots of Small Files
- Growing of FileSystem Metadata

Streaming Data Access
- Write-once, read-many-times
- High Throughput

Random Data Access
- Multiple Writers, Arbitrary file update
- Low Latency

Commodity Hardware
- Node Failure handling

The Design of HDFS

- HDFS is designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.

1. Very Large Files

- "Very large" means files that are hundreds of Megabytes, Gigabytes, or Terabytes in size.

- There are Hadoop clusters running that store Petabytes of data.

2. Streaming Data Access

- HDFS is built an efficient Data Processing Pattern is Write-Once, Read-Many-Times Pattern.

- A Dataset is typically generated or copied from source, then various analyses are performed on that dataset over time.

3. Commodity Hardware

- Hadoop doesn't require Expensive, Highly Reliable Hardware to run on.
- It's designed to run on clusters of commodity hardware (commonly hardware available from multiple vendors).
- HDFS is designed to carry on working without interruption to the user in the face of such failure.

4. Low-latency data access

- Applications require low-latency access to data, in the tens of milliseconds range, but, will not work well with HDFS.

- Remember, HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency.

- Hbase is currently a better choice for low-latency access.

5. Lots of Small Files

- Here, each file, directory or block will takes 150 bytes of sizes.
- for example, if you had one million files, each taking one block size would be need at least 300 MB of memory.

6. Multiple Writers, Arbitrary File Modifications

- Files in HDFS may be written to by a single writer.
- Writes are always made at the end of the file.
- There is no support for multiple writers, or for modifications at arbitrary offsets in the file.

HDFS Concept

- Very Large Distributed File System
  - 10K nodes, 100 million files.
  - Very large means –files are in the size of hundreds of MB, GB or TB.
  - Hadoop running today stores PB of data.
- Streaming data access
  - Built around the idea of most efficient data processing pattern.
  - Write-once, read-many pattern
  - Datasets are typically generated or copied from source, and then various analysis are performed on dataset.
  - Time to read the whole dataset in important than time to read the first record.

HDFS Concept

- Assumes Commodity Hardware
  - Hadoop doesn't requires expensive hardware,
  - Designed to run on clusters of commodity hardware.
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them

- Is also worth examining the applications for which *HDFS does not works well*.
  - Low-latency data access.
    - Applications in terms of 10 of milli-seconds
    - Optimized to provide high through-put at the expense of latency
  - Lost of small files.

HDFS Concept

- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files

- Files are broken up into blocks
  - Typically 64MB block size in version1 and 128MB block size in version2
  - Each block replicated on multiple DataNodes.

Case Study: Hadoop at Yahoo!

Title

Case Study: Using Apache Hadoop at Yahoo! for Large-Scale Data Processing

Problem Statement

Yahoo! faced several challenges:

- Massive growth in user-generated data.
- Slow processing using traditional database systems.
- Difficulty storing petabytes of data.
- High infrastructure costs.
- Need for reliable and scalable storage.
- Requirement for parallel processing of huge datasets.

Textbook

Tom White, *Hadoop: The Definitive Guide*, 3rd Edition, O'Reilly Media, 2012.

To overcome these challenges, Yahoo! became one of the earliest adopters and major contributors to the development of **Apache Hadoop**.

Hadoop provided Yahoo! with a distributed computing framework capable of storing and processing petabytes of data across thousands of low-cost commodity servers.

Hadoop Components Used

1. Hadoop Distributed File System (HDFS)

•Stores massive datasets across multiple machines.

•Automatically replicates data for fault tolerance.

•Provides scalable and reliable storage.

2. MapReduce

•Processes large datasets in parallel.

•Divides tasks among multiple nodes.

•Combines results to produce final output.

3. YARN (Yet Another Resource Negotiator)

- Manages cluster resources.
- Schedules jobs efficiently.
- Improves utilization of computing resources.

4. Hadoop Common

- Provides libraries and utilities required by Hadoop modules.

Types of Data Collected

Yahoo! collected various types of structured, semi-structured, and unstructured data.

Data Type	Examples
Search Data	Search queries
Clickstream Data	User clicks
Email Data	Yahoo Mail
Advertisement Data	Ad impressions, clicks
News Data	News articles
Social Data	User interactions
Web Logs	Server logs
Images & Videos	Multimedia content

Applications of Hadoop at Yahoo!

1. Search Engine Optimization

- Processes billions of search queries.
- Improves search rankings.
- Delivers faster search results.

2. Advertisement Analytics

- Analyzes advertisement clicks.
- Measures campaign performance.
- Improves targeted advertising.

3. Clickstream Analysis

- Tracks user browsing behavior.
- Understands customer interests.
- Improves website navigation.

4. Fraud Detection

- Detects suspicious user activities.
- Identifies click fraud.
- Improves platform security.

5. Recommendation Systems

- Suggests personalized news.
- Recommends advertisements.
- Enhances user engagement.

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)
MCA DEPARTMENT

LECTURE NOTES

Subject Name: BIG DATA ANALYTICS

Year / Branch: IV B.TECH – VII SEMESTER

Regulation: R23

Prepared By: Mr. G YUVARAJU,

Assistant Professor

III - WORKING WITH PIG AND HIVE

Unit Overview

This unit introduces Apache Hive and Apache Pig, two important data warehousing and data processing tools in the Hadoop ecosystem. It begins with the fundamentals of Hive, including its architecture, data types, file formats, and Hive Query Language (HiveQL) for defining and manipulating data. The unit covers table creation, internal and external tables, and various data manipulation operations using HiveQL. It also explains advanced concepts such as logical joins, window functions (RANK, DENSE_RANK, ROW_NUMBER), table partitioning (static and dynamic), and bucketing for efficient data organization and query optimization.

The unit further introduces Apache Pig, a high-level platform for analyzing large datasets using the Pig Latin scripting language. It covers Pig architecture, statements, expressions, data types, and various data processing operators used for loading, transforming, filtering, grouping, joining, and storing data. Together, Hive and Pig simplify Big Data processing by providing user-friendly interfaces for querying and analyzing large datasets stored in Hadoop.

Learning Outcomes

After completing this unit, students will be able to:

1. Explain the architecture and working principles of Apache Hive and Apache Pig.
2. Describe Hive data types, file formats, and the features of Hive Query Language (HiveQL).
3. Create and manage internal and external Hive tables using HiveQL.
4. Perform data definition and data manipulation operations using HiveQL.
5. Apply logical joins, window functions, partitioning, and bucketing techniques to optimize query performance.
6. Explain the Pig architecture and write Pig Latin scripts using statements, expressions, and data processing operators.
7. Analyze and process large datasets efficiently using Hive and Pig within the Hadoop ecosystem.

Importance of Studying this Unit

As the volume of data continues to grow, organizations require efficient tools to store, query, and analyze massive datasets. Apache Hive enables users to perform SQL-like queries on distributed data stored in Hadoop without requiring complex programming, making it suitable for data warehousing and business intelligence applications. Apache Pig simplifies large-scale data processing through its high-level scripting language, reducing the complexity of writing MapReduce programs.

Studying this unit helps students understand how Big Data can be efficiently queried, transformed, and analyzed using Hive and Pig. The concepts learned form the foundation for advanced Big Data technologies such as Apache Spark, HBase, NoSQL databases, Data Warehousing, Data Analytics, Machine Learning, Artificial Intelligence, and Cloud Computing. These skills are highly valuable for careers in Data Engineering, Data Analytics, Business Intelligence, Big Data Development, and Cloud-based Data Processing.

Key Concepts

Apache Hive, Hive Architecture, HiveQL, Hive Data Types, File Formats, Data Definition Language (DDL), Data Manipulation Language (DML), Internal Table, External Table, Logical Joins, Window Functions, Rank, Dense Rank, Row Number, Table Partitioning, Static Partitioning, Dynamic Partitioning, Bucketing, Apache Pig, Pig Latin, Pig Architecture, Statements, Expressions, Data Types, Data Processing Operators, Loading Data, Storing Data, Hadoop Ecosystem, Data Warehousing, Big Data Query Processing.

PIG

- Pig is a high-level platform or tool which is used to process the large datasets.
- It provides a high-level of abstraction for processing over the MapReduce.
- It provides a high-level scripting language, known as Pig Latin which is used to develop the data analysis codes.
- First, to process the data which is stored in the HDFS, the programmers will write the scripts using the Pig Latin Language.
- Internally Pig Engine(a component of Apache Pig) converted all these scripts into a specific map and reduce task.
- But these are not visible to the programmers in order to provide a high-level of abstraction.
- Pig Latin and Pig Engine are the two main components of the Apache Pig tool.
- The result of Pig always stored in the HDFS.

Need of Pig:

- One limitation of MapReduce is that the development cycle is very long.
- Writing the reducer and mapper, compiling packaging the code, submitting the job and retrieving the output is a time-consuming task.
- Apache Pig reduces the time of development using the multi-query approach.
- Also, Pig is beneficial for programmers who are not from Java background.
- 200 lines of Java code can be written in only 10 lines using the Pig Latin language.
- Programmers who have SQL knowledge needed less effort to learn Pig Latin.

Pig vs MapReduce

Apache Pig	MapReduce
It is a scripting language.	It is a compiled programming language.
Abstraction is at a higher level.	Abstraction is at a lower level.
It requires fewer lines of code compared to MapReduce.	It requires more lines of code.
Less effort is needed for Apache Pig development.	More development effort is required for MapReduce.

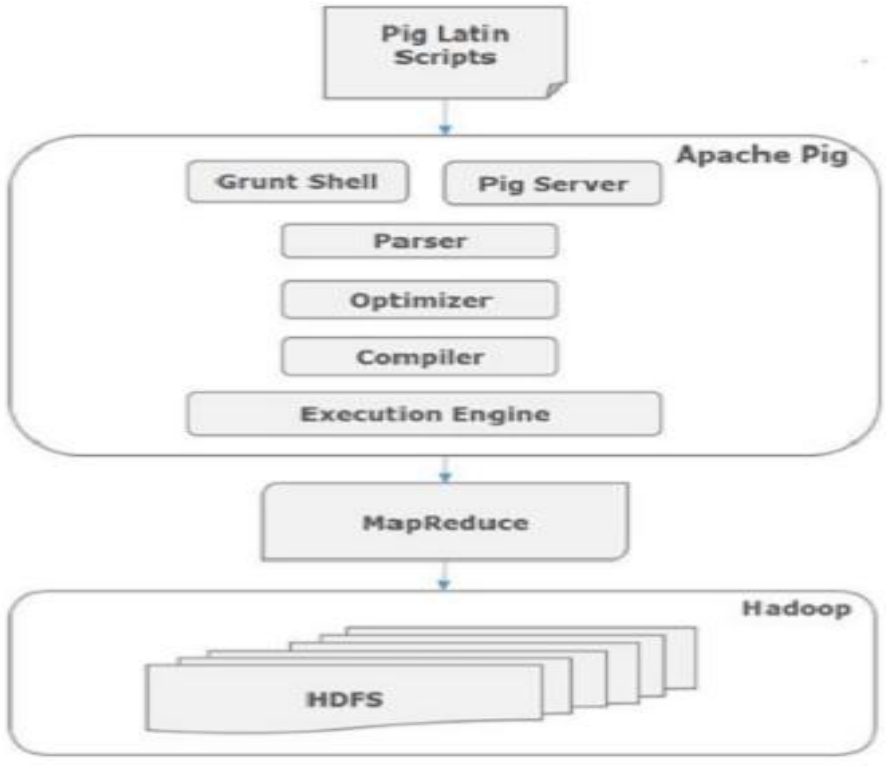
Apache Pig	MapReduce
Code efficiency is less compared to MapReduce.	Code efficiency is higher compared to Pig.
Pig provides built-in functions for ordering, sorting, and union operations.	Data operations are harder to perform compared to Pig.
It allows nested data types like map, tuple, and bag.	It does not allow nested data types.

Apache Pig - Architecture

The language used to analyze data in Hadoop using Pig is known as Pig Latin. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmers job easy. The architecture of Apache Pig is shown below.



Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown**.**

Compiler

The compiler compiles the optimized logical plan into a series of MapReduce jobs.

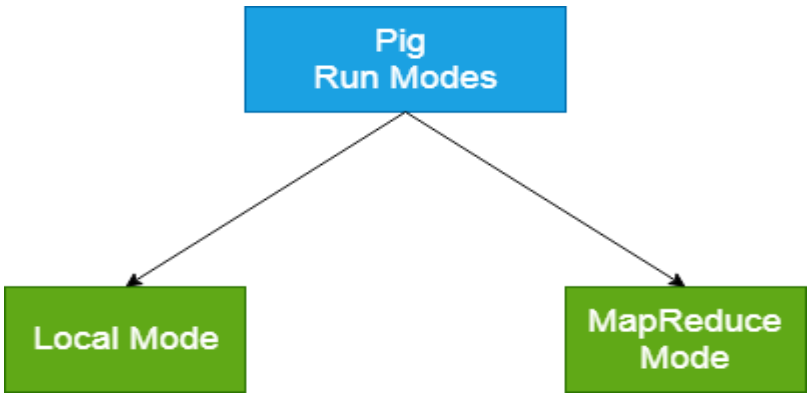
Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

Pig Execution Types

Apache Pig executes in two modes:

1. **Local Mode and**
2. **MapReduce Mode.**



Local Mode

- It executes in a single JVM and is used for development experimenting and prototyping.
- Here, files are installed and run using localhost.
- The local mode works on a local file system. The input and output data stored in the local file system.
- The command for local mode grunt shell:

$ pig-x local

Map Reduce Mode

- The MapReduce mode is also known as Hadoop Mode.
- It is the default mode.
- In this Pig renders Pig Latin into MapReduce jobs and executes them on the cluster.
- It can be executed against semi-distributed or fully distributed Hadoop installation.
- Here, the input and output data are present on HDFS.
- The command for Map reduce mode:

$ pig -x mapreduce

Pig Example

```
A = Load 'rp/WeatherDataset.txt' as (line:chararray);

B = foreach A generate
SUBSTRING(line,15,19),SUBSTRING(line,88,92),SUBSTRING(line,92,93);

store B into 'rp/revised5';

E = load 'rp/revised5/part-m-00000' using PigStorage('\t') as (year:int,temp:int,quality:int);

filtered_records = FILTER E by temp!=9999 AND quality IN (0,1,4,5,9);

grouped_records = GROUP filtered_records by year;

max_temp = FOREACH grouped_records GENERATE group,MAX(filtered_records.temp);

dump max_temp;
```

Pig Latin

- Pig Latin is the Language used in Pig
- Pig Latin Structure includes

Comments

Statements

Comments

- Proper use of commenting can make code maintenance much easier, as well as helps finding bugs faster.
- Pig Latin has two types of comment operators:

SQL-style single-line comments (--) and

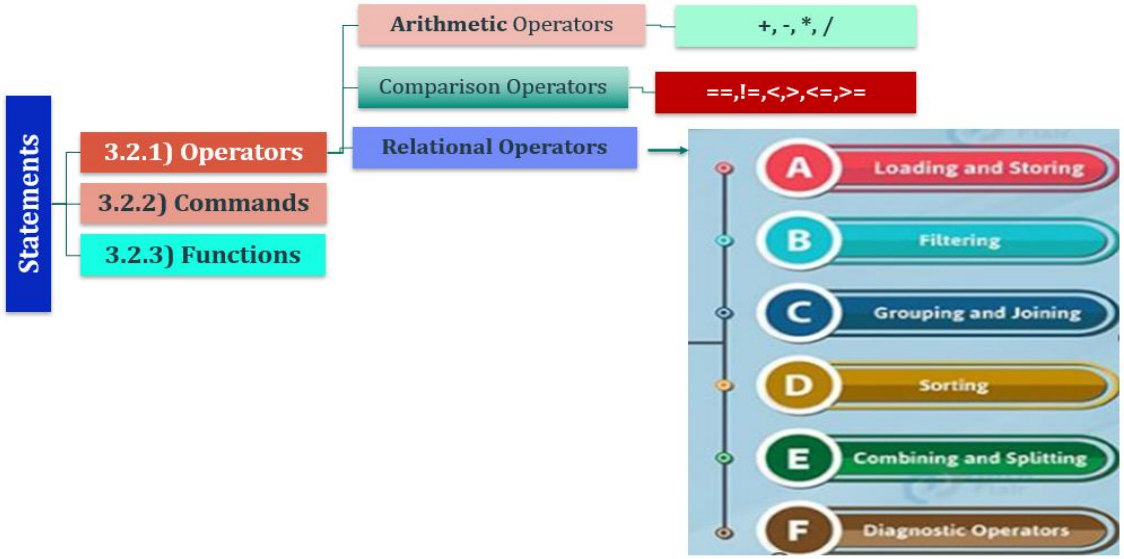
Java-style multiline comments (/* */).

Statements

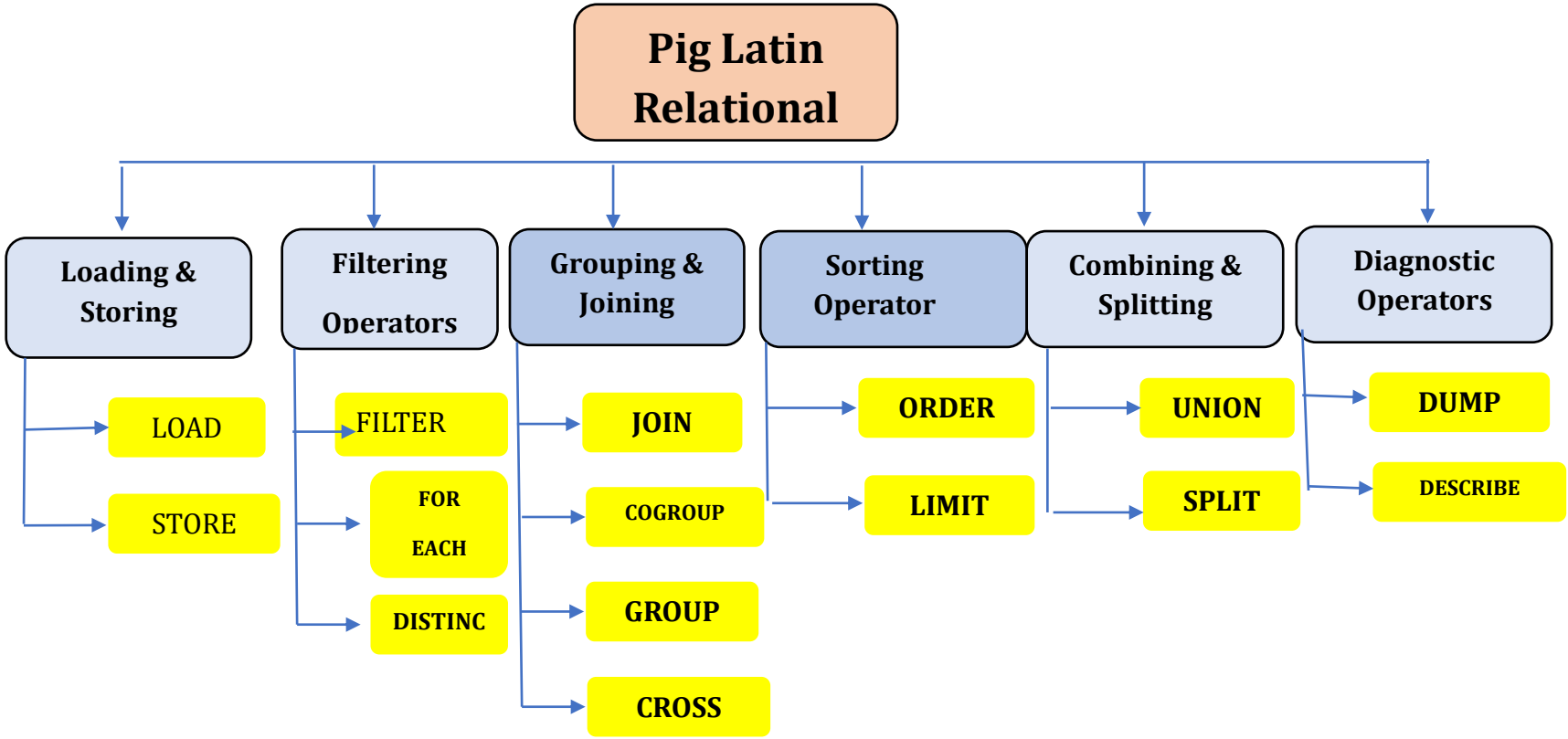
The statements are the basic constructs while processing data using Pig Latin.

- The statements can work with relations including expressions and schemas.
- However, every statement terminate with a semicolon (;).
- We will perform different operations using Pig Latin operators.
- Pig Latin statements inputs a relation and produces some other relation as output.
- As a Pig Latin Program is executed ,each statement is parsed in turn If there are syntax errors or other problems, such as undefined aliases, the interpreter will halt and display an error message.
- The Interpreter builds a logical plan for every relational operation, which forms the core of a pig Latin program.
- The Logical Plan for the statement is added to the logical plan for the program to far, and then the interpreter moves on to the next statement
- Its important to note that no data processing takes place while the logical plan of the program is being constructed
- When the Pig Latin interpreter sees the first line containing the load statement. It conforms that it is syntactically and semantically correct and adds it to the logical plan, but it does not load the data from the file .Indeed, where would it load it? Into memory?even if it did fit into memory, what would it do with data
- Similarly , pig validates GROUP and FOREACH statements and adds them to the logical plan without executing them

- The Trigger for Pig to start execution is the Dump statement
- At that point, the logical plan is compiled into a physical plan and executed.
- Pig Latin Statements Includes
    - **Operators**
    - **Commands**
    - **Functions**



Statements & operators



Loading and Storing Operator

Load Operator

Is used to load data into Apache Pig from HDFS

Syntax

Rel_Name = LOAD ‘input file path’ using PigStorage(‘\t’) as schema

where

Rel_Name – Name of the Relation where to store the data

Input file path – path of the HDFS file

Scheme – Resultant Relation schema

Eg:

A= LOAD '/data/sample.txt' as PigStorage('\t') as (sno:int,sname:chararray,branch:chararray)

- The Load statement consists of 2 parts by the "=" operator
- On the left hand side, need to mention the relation name where we want to store the data and
- On the right hand side, we have to define how to store the data (schema)

$ cat > sample.txt

100 smith MCA

101 Jones M.Tech

102 King MCA

103 Ivan MCA

104 Korth M.Tech

Ctrl +Z

$ hadoop fs –mkdir /data → Create a Directory called data in HDFS

$ hadoop fs –put sample.txt /data → Push sample.txt from local file system to HDFS data directory

$ hadoop fs -ls /data → check sample.txt exists in data directory

$ pig → Move to Pig

grunt> student = LOAD '/data/sample.txt' using PigStorage('\t') as

(sno:int, sname:chararray,branch:chararray);

grunt>dump student

(100 smith MCA)

(101 Jones M.Tech)

(102 King MCA)

(103 Ivan MCA)

(104 Korth M.Tech)

Load Operator

Purpose	Syntax	Example	Output
To Load Data from HDFS to Pig latin	Rel_Name = LOAD 'input file path' using PigStorage('\t') as schema where ➢ Rel_Name – Name of the Relation where to store the data ➢ Input file path – path of the HDFS file ➢ Scheme – Resultant Relation schema	grunt> student = LOAD '/data/sample.txt' using PigStorage('\t') as (sno:int, sname:chararray, branch :chararray); grunt>dump student	(100 smith MCA) (101 Jones M.Tech) (102 King MCA) (103 Ivan MCA) (104 Korth M.Tech)

Store Operator

Purpose	Syntax	Example	Output

Is used to store data from Apache Pig to HDFS	STORE Rel_Name INTO 'HDFS Directory';
	where
	Rel_Name – Name of the Relation which needs to be stored from pig to HDFSc
	HDFS Directory – HDFS Path where to store
		$ hadoop fs –mkdir /Results
		$ pig
		grunt>STORE student into '/Results';
			$ hadoop fs –cat /Results/part-m-00000
			100 smith MCA
			101 Jones M.Tech
			102 King MCA
			103 Ivan MCA
			104 Korth M.Tech

Filter Operator

Purpose	Syntax	Example	Output
Is used FILTER Rows from the Relation Is used to get specific rows of the Relation It is equivalent to WHERE clause in SQL	Res_Relation = FILTER Rel_Name BY Condition where Rel_Name – Name of the Relation on which filter need to be performed Condition– Filter condition	Grunt>MCA_Stu = FILTER student BY branch == 'MCA'; Grunt>M.Tech_Stu = FILTER student BY branch == 'MCA';	Grunt>dump MCA_Stu; (100 smith MCA) (102 King MCA) (103 Ivan MCA) Grunt>dump M.Tech_Stu ; 101 Jones M.Tech 104 Korth M.Tech

ForEach .. Generate Operator

Purpose	Syntax	Example	Output
Is used to get specific columns It is equivalent to SELECT clause in SQL	Res_Relation = FOREACH Rel_Name GENERATE COL1,COL2,…COLN Where Rel_Name – Name of the Relation from which specific columns to be retreived COL1,COL2,..COLN are the specific columns to be rerived	Grunt> no_branch = FOREACH student GENERATE sno,branch; If schema is not defined then Grunt>no_branch = FOREACH student GENERATE $1,S3;	Grunt>dump no_banch; (100 MCA) (101 M.Tech) (102 MCA) (103 MCA) (104 M.Tech)

GROUPING & JOINING Operators

JOIN OPERATOR

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

Purpose	Syntax	Example	Output
Performs an inner join of two or more relations based on common field values.	Res_Relation = JOIN Rel1 by Key, Rel2 by key where Rel1 is the First Relation Rel2 is the Second Relation Key is the common field	grunt> emp_dept = JOIN emp BY deptno ,dept BY deptno	Grunt>dump emp_dept; eno ename deptno deptno dname 100 Smith 10 10 HR 101 Jones 20 20 SYSTEM 102 King 10 10 HR 103 Ivan 20 20 SYSTEM 104 Korth 30 30 MARKETING

COGROUP OPERATOR

- Join always give a flat structure
- The COGROUP statement is similar to join but instead creates a nested set of output tuples.
- COGROUP generates a tuple for each unique grouping key
  - ✓ The first field of each tuple is the key
  - ✓ And the remaining fields are bags of tuples from the relations with a matching key
  - ✓ The first bag contains the matching tuples from relation A with the same key.
  - ✓ The second bag contains the matching tuples from relation B with the same key

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

Purpose	Syntax	Example	Output
Same as Join instead JOIN returns flat structure whereas COGROUP returns nested tuples	Res_Relation = COGROUP Rel1 by Key, Rel2 by key where Rel1 is the First Relation Rel2 is the Second Relation Key is the common field	grunt> emp_dept= COGROUP emp BY deptno ,dept BY deptno	Grunt>dump emp_dept; (10, {(100,Smith,10),(102,King,10)},{(10,HR)}) (20, {(101,Jones,20),(103,Ivan,20)},{(20,SYSTEM)}) (30, {(104,Korth,30)},{(30,MARKETING)})

GROUP OPERATOR

Purpose	Syntax	Example	Output
Join and Cogroup works on one or more relations whereas GROUP operator is used to group data in a single relation	Res_Relation = GROUP Rel by Field where Rel is the Relation Field is column or field on which to group the data	grunt> deptwise = GROUP emp by deptno	Grunt>dump deptwise; (10, {(100,Smith,10),(102,King,10)}) (20, {(101,Jones,20),(103,Ivan,20)}) (30, {(104,Korth,30)})

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

CROSS OPERATOR

Purpose	Syntax	Example	Output
Returns Cross Product of 2 or more Relations	Res_Relation = CROSS Rel1,Rel2 where Rel1 and Rel2 are Relations on which cross product need to be applied	grunt> cp = CROSS emp,dept;	**Grunt>dump cp;** 100 Smith 10 10 HR 10 100 Smith 10 20 SYSTEM 20 101 Jones 20 10 HR 10 101 Jones 20 20 SYSTEM 20 102 King 10 10 HR 10 102 King 10 20 SYSTEM 20

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM

Sorting and Limiting Data

ORDER OPERATOR

Purpose	Syntax	Example	Output
Returns data in Sorted order	Res_Relation = ORDER Rel1 by FieldName where Rel1 is the Relation FieldName is the column on which data to be sorted	grunt> sorted_names = ORDER emp BY ename;	Grunt>dump sorted_names; (103 Ivan 20) (101 Jones 20) (102 King 10) (104 Korth 30) (100 Smith 10)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

LIMIT OPERATOR

Purpose	Syntax	Example	Output
is used to get a limited number of tuples from a relation.	Res_Relation = LIMIT Rel1 Number where Rel1 is the Relation Number is Number of tuples to be displayed	grunt>Limited_tuples = LIMIT emp 2;	Grunt>dump sorted_names; (100 Smith 10) (101 Jones 20)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

Combining and Splitting Data

UNION OPERATOR

Purpose	Syntax	Example	Output
is used to combine 2 or more relations into one	Res_Relation = UNION Rel1,Rel2 where Rel1 is the First Relation Rel2 is the Second Relation	grunt>combined_rels = UNION emp,dept;	Grunt>dump combined_rels; (100 Smith 10) (101 Jones 20) (102 King 10) (103 Ivan 20) (104 Korth 30) (10 HR) (20 SYSTEM) (30 MARKETING)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

SPLIT OPERATOR

Purpose	Syntax	Example	Output
is used to Split single Relation into 2 or more Relations	Split Rel1 into Rel2 if condition1 , Rel3 if condition2 where Rel1, Rel2, Rel3 are Relations. Condition1 and condition 2 are conditions on which the Rel1 to be split	grunt>split emp into deptno10 if deptno ==10, deptno20 if deptno ==20;	grunt>dump deptno10; (100 Smith 10) (102 King 10) Grunt>dump deptno20; (101 Jones 20) (103 Ivan 20)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

Diagnostic Operators

DUMP OPERATOR

Purpose	Syntax	Example	Output
is used to Display Results on the Screen	DUMP Relation_Name; where Relation_Name is the name of the Relation whose content to be printed	grunt>DUMP emp;	grunt>dump emp; (100 Smith 10) (101 Jones 20) (102 King 10) (103 Ivan 20) (104 Korth 30)

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

DESCRIBE OPERATOR

Purpose	Syntax	Example	Output
describes the schema of a relation.	DESCRIBE Relation_Name; where Relation_Name is the name of the Relation	grunt>DESCRIBE emp;	grunt>DESCRIBE emp; emp: { eno:int,ename:chararray,deptno:int}

grunt>dump emp

eno	ename	deptno
100	Smith	10
101	Jones	20
102	King	10
103	Ivan	20
104	Korth	30

grunt>dump dept

deptno	dname
10	HR
20	SYSTEM
30	MARKETING

PIG DATA TYPES

S.N.	Data Type	Description & Example
1	int	Represents a signed 32-bit integer. **Example** : 8
2	long	Represents a signed 64-bit integer. **Example** : 5L
3	float	Represents a signed 32-bit floating point. **Example** : 5.5F
4	double	Represents a 64-bit floating point. **Example** : 10.5
5	chararray	Represents a character array (string) in Unicode UTF-8 format. **Example** : 'tutorials point'
6	Bytearray	Represents a Byte array (blob).
7	Boolean	Represents a Boolean value. **Example** : true/ false.
8	Datetime	Represents a date-time. **Example** : 1970-01-01T00:00:00.000+00:00
9	Biginteger	Represents a Java BigInteger. **Example** : 60708090709
10	Bigdecimal	Represents a Java BigDecimal .**Example** : 185.98376256272893883
Complex Types		
11	Tuple	A tuple is an ordered set of fields. **Example** : (raja, 30)
12	Bag	A bag is a collection of tuples. **Example** : {(raju,30),(Mohhammad,45)}
13	Map	A Map is a set of key-value pairs. **Example** : ['name'#'Raju', 'age'#30]

SCHEMAS

- Schemas enable to assign names and types to fields .
- Schemas are optional but we encourage you to use them whenever possible; type declarations result in better parse-time error checking and more efficient code execution.
- Schema can be defined for both names and types of the fields

- Schema can be defined only for field names, in this case, the field type defaults to bytearray. Here fields can be referred through names
- Schema may not be defined ; in this case, the field is un-named and the field type defaults to bytearray. Here the fields can be referred through positional notation like $1,$2,etc..

Unknown Schema Handling

- When you JOIN/COGROUP/CROSS multiple relations, if any relation has an unknown schema (or no defined schema, also referred to as a null schema), the schema for the resulting relation is null.
- If you UNION two relations with incompatible schema, the schema for resulting relation is null.

Schema Merging

- In Pig, no need to declare the schema for every new relation in the data flow
- pig can figure out the resulting schemas for the output of a relational operation by considering the schema of the input relation
- Filter , limit are the operators who resultant relation schema is same as input schema

HIVE

Introduction to Hive

Apache Hive is an open-source data warehouse software built on top of the Hadoop ecosystem. It provides a SQL-like query language called Hive Query Language (HiveQL) that enables users to store, manage, query, and analyze large datasets stored in the Hadoop Distributed File System (HDFS). Hive converts HiveQL queries into MapReduce, Apache Tez, or Apache Spark jobs for distributed processing.

Hive was originally developed by Facebook to simplify the processing of massive datasets and later became an Apache Software Foundation project.

Need for Hive

Writing MapReduce programs in Java is complex and time-consuming. Hive provides an SQL-like interface that allows users familiar with SQL to perform data analysis without writing Java code.

Hive is mainly used for:

- Data warehousing
- Business intelligence
- Data summarization
- Batch processing
- Report generation
- Large-scale data analysis

File Formats in Hive

Introduction

Apache Hive supports multiple **file formats** for storing data in the Hadoop Distributed File System (HDFS). The choice of file format affects **storage efficiency, compression, query performance, and processing speed**. Hive supports both **row-oriented** and **column-oriented** file formats, allowing users to choose the most suitable format based on their application requirements.

Types of File Formats in Hive

1. Text File

A **Text File** is the default file format in Hive. Data is stored as plain text, where each record occupies one line and fields are separated by delimiters such as commas, tabs, or spaces.

Features

- Default file format in Hive.
- Human-readable.
- Stores one record per line.

- Suitable for simple data storage.

Applications

- Small datasets
- Data import and export
- Testing and development

2. Sequence File

A **Sequence File** is a binary file format provided by Hadoop that stores data as **key-value pairs**. It is mainly used for storing intermediate data generated during MapReduce processing.

Features

- Binary file format.
- Stores data as key-value pairs.
- Supports compression.
- Faster than text files.

Applications

- Intermediate MapReduce output
- Hadoop data storage

3. RCFile (Record Columnar File)

RCFile (Record Columnar File) is a column-oriented storage format that combines row-based and column-based storage. It stores records in rows while organizing data by columns.

Features

- Column-based storage.
- Supports compression.
- Improves query performance.
- Reduces disk I/O operations.

Applications

- Data warehousing
- Analytical processing

4. ORC File (Optimized Row Columnar)

ORC (Optimized Row Columnar) is a highly optimized columnar storage format developed specifically for Hive. It provides excellent compression, indexing, and fast query execution.

Features

- Columnar storage.
- High compression ratio.
- Built-in indexing.
- Fast query execution.
- Optimized for Hive.

Applications

- Data warehouses
- Business intelligence
- Big Data analytics

5. Parquet File

Parquet is a column-oriented storage format designed for efficient storage and retrieval of large datasets. It is widely used with Hive, Spark, and other Big Data frameworks.

Features

- Column-based storage.
- High compression.
- Efficient encoding techniques.
- Cross-platform compatibility.
- Suitable for analytical queries.

Applications

- Big Data analytics
- Machine Learning
- Cloud data lakes

6. Avro File

Avro is a row-oriented data serialization format that stores both **data and schema** together. It is widely used for data exchange between different systems.

Features

- Row-based storage.

- Stores schema with data.
- Supports schema evolution.
- Compact binary format.
- Platform-independent.

Applications

- Data exchange
- Streaming applications
- Kafka integration

Comparison of Hive File Formats

File Format	Storage Type	Compression	Query Performance	Common Use
Text File	Row-based	Low	Slow	Small datasets and testing
Sequence File	Binary	Medium	Good	MapReduce intermediate data
RCFile	Column-based	High	Better	Data warehousing
ORC	Column-based	Very High	Excellent	Hive data warehouses
Parquet	Column-based	Very High	Excellent	Hive, Spark, cloud analytics
Avro	Row-based	Medium	Good	Data exchange and streaming

HiveQL Data Definition Language (DDL)

Introduction

HiveQL Data Definition Language (DDL) is a set of commands used to define and manage the structure of databases and tables in Apache Hive. DDL commands are used to create, modify, describe, and delete databases, tables, views, partitions, and other schema objects. These commands define how data is organized and stored in the Hadoop Distributed File System (HDFS).

Table Creation in Hive

Hive supports two types of tables:

1. Internal Table (Managed Table)
2. External Table

Internal Table

An Internal Table is managed completely by Hive. When the table is dropped, both the table schema (metadata) and the data stored in HDFS are deleted.

```
create table emp (
   eid int,
   ename string,
   esal int,
   eloc string
)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;
```

Loading Data:

```
1. load data local inpath 'sample1.txt' into table emp;
2. load data inpath '/yuvaraj/sample2.txt' into table emp;
3. insert overwrite table emp1 select * from emp;
```

External Table

Only metadata is deleted when the table is dropped; data remains in HDFS.

```
create external table emp_ex (
   eid int,
   ename string,
   esal int,
   eloc string
)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;

insert overwrite table emp_ex select * from emp1;
```

Partitioning

```
create table emp_part (
eid int,
ename string,
esal int
)
partitioned by (eloc string)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;
```

Static Partition:

```
insert overwrite table emp_part partition(eloc='hyd') select eid,ename,esal from emp1 where eloc='hyd';
insert overwrite table emp_part partition(eloc='bng') select eid,ename,esal from emp1 where eloc='bng';
```

dynamic partition:

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table emp_part partition(eloc) select eid,ename,esal,eloc from emp1;
```

Bucketing

```
create table emp_cluster(
eid int,
ename string,
esal int,
eloc string
)
clustered by (eid) into 4 buckets
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile;

insert overwrite table emp_cluster select * from emp1;
```

List Buckets:

```
hdfs dfs -ls /user/hive/warehouse/emp_cluster
```

alter table:

```
alter table emp add columns(dept string);
```

```
alter table emp replace columns(
eid int,
ename string,
esal int,
eloc string,
dept string
);
```

```
alter table emp rename to employee;
```

Joins in Hive

Introduction

A **Join** in Hive is used to combine rows from two or more tables based on a common column. Joins help retrieve related information stored in different tables. Hive supports several types of joins, including **Inner Join, Left Outer Join, Right Outer Join, Full Outer Join, Left Semi Join,** and **Cross Join**.

Employee Table (emp)

eid	ename	deptid	salary
101	Rahul	1	50000
102	Priya	2	60000
103	Amit	3	55000
104	Neha	4	45000

Department Table (dept)

deptid	dname
1	HR
2	Finance
3	IT
5	Sales

1. Inner Join

An **Inner Join** returns only the rows that have matching values in both tables.

Syntax

```
select emp.eid, emp.ename, dept.dname
from emp
join dept
on emp.deptid = dept.deptid;
```

Output

eid	ename	dname
101	Rahul	HR
102	Priya	Finance
103	Amit	IT

Explanation:

- Employee **Neha (deptid=4)** is not displayed because there is no matching department.
- Department **Sales (deptid=5)** is also not displayed because no employee belongs to it.

2. Left Outer Join

Definition

A **Left Outer Join** returns **all rows from the left table** and only the matching rows from the right table. If there is no match, NULL values are returned.

Syntax

```
select emp.eid, emp.ename, dept.dname
from emp
left outer join dept
on emp.deptid = dept.deptid;
```

Output

eid	ename	dname
101	Rahul	HR
102	Priya	Finance
103	Amit	IT
104	Neha	NULL

Explanation:

- All employees are displayed.
- Neha has no matching department, so **NULL** is shown.

3. Right Outer Join

Definition

A **Right Outer Join** returns **all rows from the right table** and the matching rows from the left table.

Syntax

```
select emp.eid, emp.ename, dept.dname
from emp
right outer join dept
on emp.deptid = dept.deptid;
```

Output

eid	ename	dname
101	Rahul	HR
102	Priya	Finance
103	Amit	IT
NULL	NULL	Sales

Explanation:

- All departments are displayed.
- Sales has no employee, so employee details are NULL.

4. Full Outer Join

A **Full Outer Join** returns **all rows from both tables**. If no matching record exists, NULL values are displayed.

Syntax

```
select emp.eid, emp.ename, dept.dname
from emp
full outer join dept
on emp.deptid = dept.deptid;
```

Output

eid	ename	dname
101	Rahul	HR
102	Priya	Finance
103	Amit	IT
104	Neha	NULL
NULL	NULL	Sales

Explanation:

- Displays every employee and every department.
- Unmatched values are filled with NULL.

5. Left Semi Join

Definition

A **Left Semi Join** returns only the rows from the left table that have matching values in the right table. It does **not** return columns from the right table.

Syntax

```
select * from emp left semi join dept
on emp.deptid = dept.deptid;
```

Output

eid	ename	deptid	salary
101	Rahul	1	50000
102	Priya	2	60000
103	Amit	3	55000

Explanation:

- Returns only employees whose department exists.
- Department columns are not included.

6. Cross Join

Definition

A **Cross Join** returns the **Cartesian product** of two tables. Every row of the first table is combined with every row of the second table.

Syntax

select *from emp cross join dept;

Output (Partial)

eid	ename	deptid	dname
101	Rahul	1	HR
101	Rahul	2	Finance
101	Rahul	3	IT
101	Rahul	5	Sales
102	Priya	1	HR
...	...	...	...

Explanation:

- If emp has **4 rows** and dept has **4 rows**, the result contains **16 rows (4 × 4)**.
-

Window Functions in Hive (RANK, DENSE_RANK, ROW_NUMBER)

Window functions in **Hive** are used to perform calculations across a set of rows related to the current row without grouping the data. They are commonly used for **ranking, numbering, and analytical operations**.

Syntax

```
function_name() over (
   [partition by column_name]
   [order by column_name]
)
```

- **PARTITION BY**: Divides the data into groups.
- **ORDER BY**: Defines the order within each partition.

Sample Employee Table

EmpID	Name	Department	Salary
101	Ravi	HR	30000
102	Sita	HR	40000
103	Ram	HR	40000
104	John	IT	50000

EmpID	Name	Department	Salary
105	Mary	IT	60000
106	David	IT	60000

Create Table:

```
create table employee(
empid int,
ename string,
dept string,
salary int
)
row format delimited
fields terminated by ',';
```

Load Data:

```
load data local inpath '/home/hadoop/employee.csv'
into table employee;
```

1. ROW_NUMBER()

Definition

ROW_NUMBER() assigns a **unique sequential number** to each row.

Even if two rows have the same salary, each row gets a different number.

Syntax

```
SELECT empid,
      ename,
      dept,
      salary,
      ROW_NUMBER() OVER(PARTITION BY dept ORDER BY salary DESC) AS
row_num
FROM employee;
```

Output

Department	Name	Salary	ROW_NUMBER
HR	Sita	40000	1
HR	Ram	40000	2
HR	Ravi	30000	3

Department	Name	Salary	ROW_NUMBER
IT	Mary	60000	1
IT	David	60000	2
IT	John	50000	3

Characteristics

- Always produces unique numbers.
- No duplicate ranks.
- Best for selecting the top N records.

2. RANK()

Definition

RANK() assigns the same rank to equal values.

If there is a tie, the next rank is skipped.

Syntax

```
SELECT empid,
    ename,
    dept,
    salary,
    RANK() OVER(PARTITION BY dept ORDER BY salary DESC) AS rank_no
FROM employee;
```

Output

Department	Name	Salary	RANK
HR	Sita	40000	1
HR	Ram	40000	1
HR	Ravi	30000	3
IT	Mary	60000	1
IT	David	60000	1
IT	John	50000	3

Characteristics

- Equal values receive the same rank.

- Ranking contains gaps.
- Useful in competition ranking.

3. DENSE_RANK()

definition

dense_rank() also gives the same rank to duplicate values.

unlike rank(), it does not skip any ranks.

syntax

```
select empid,
    ename,
    dept,
    salary,
    dense_rank() over(partition by dept order by salary desc) as dense_rank_no
from employee;
```

Output

Department	Name	Salary	DENSE_RANK
HR	Sita	40000	1
HR	Ram	40000	1
HR	Ravi	30000	2
IT	Mary	60000	1
IT	David	60000	1
IT	John	50000	2

Characteristics

- Same values receive the same rank.
- No gaps in ranking.
- Useful when continuous ranking is required.

Difference Between ROW_NUMBER(), RANK(), and DENSE_RANK()

Feature	ROW_NUMBER()	RANK()	DENSE_RANK()
Duplicate values	Different numbers	Same rank	Same rank

Feature	ROW_NUMBER()	RANK()	DENSE_RANK()
Rank gaps	No	Yes	No
Unique numbering	Yes	No	No
Best use	Top N rows	Competition ranking	Continuous ranking

Example Without PARTITION BY

```
select empid,
    ename,
    salary,
    row_number() over(order by salary desc) as row_num,
    rank() over(order by salary desc) as rank_num,
    dense_rank() over(order by salary desc) as dense_rank_num
from employee;
```

Output

Name	Salary	ROW_NUMBER	RANK	DENSE_RANK
Mary	60000	1	1	1
David	60000	2	1	1
John	50000	3	3	2
Sita	40000	4	4	3
Ram	40000	5	4	3
Ravi	30000	6	6	4

Case Study: Employee Salary Analysis Using Hive Window Functions

Title: Analyzing Employee Performance and Salary Rankings Using Apache Hive

Problem Statement

ABC Technologies has more than **10,000 employees** working across different departments such as **HR, IT, Sales, and Finance**. The organization stores millions of employee records in a Hadoop ecosystem.

The HR department wants to perform salary-based analysis to improve decision-making and reward employee performance.

The main requirements are:

- Identify the **highest-paid employees** in each department.
- Assign **salary rankings** to employees within their departments.
- Handle employees having the **same salary values** correctly.
- Select the **top three employees from each department** for annual bonus allocation.
- Analyze employee salary patterns efficiently using SQL-like queries.

Since traditional databases cannot efficiently process large-scale employee data, ABC Technologies uses **Apache Hive** on Hadoop to perform distributed data analysis using **Hive Window Functions**.

Dataset: Employee Salary Details

Employee_ID	Employee_Name	Department	Salary
101	Ravi	IT	85000
102	Priya	HR	75000
103	Arun	IT	90000
104	Sneha	Finance	95000
105	Kiran	Sales	80000
106	Anjali	IT	90000
107	Rahul	HR	72000

Employee_ID	Employee_Name	Department	Salary
108	Meena	Finance	88000
109	Vijay	Sales	78000
110	Divya	HR	85000

Hive Table Creation

```
CREATE TABLE employee_salary (
   employee_id INT,
   employee_name STRING,
   department STRING,
   salary INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Applying Hive Window Functions

1. Ranking Employees Based on Salary

Query:

```
SELECT employee_id, employee_name, department, salary, RANK() OVER(
PARTITION BY department ORDER BY salary DESC) AS salary_rank FROM
employee_salary;
```

Output:

Employee_Name	Department	Salary	Salary Rank
Arun	IT	90000	1
Anjali	IT	90000	1
Ravi	IT	85000	3
Sneha	Finance	95000	1
Meena	Finance	88000	2
Divya	HR	85000	1
Priya	HR	75000	2

Employee_Name	Department	Salary	Salary Rank
Rahul	HR	72000	3

2. Finding Highest Paid Employee in Each Department

Query:

SELECT * FROM (SELECT employee_name, department, salary, ROW_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) AS rank FROM employee_salary) as WHERE rank = 1;

Output:

Employee_Name	Department	Salary
Arun	IT	90000
Sneha	Finance	95000
Divya	HR	85000
Kiran	Sales	80000

Explanation:
ROW_NUMBER() assigns a unique sequence number to employees in each department.

3. Selecting Top Three Employees from Each Department

Query:

SELECT * FROM (SELECT employee_name, department, salary, RANK() OVER(

PARTITION BY department ORDER BY salary DESC) AS rank FROM employee_salary) as WHERE rank <= 3;

Output:

Employee_Name	Department	Salary	Rank
Arun	IT	90000	1
Anjali	IT	90000	1
Ravi	IT	85000	3
Sneha	Finance	95000	1
Meena	Finance	88000	2
Divya	HR	85000	1
Priya	HR	75000	2

Employee_Name	Department	Salary	Rank
Rahul	HR	72000	3

Hive Window Functions Used

Function	Purpose
ROW_NUMBER()	Assigns a unique number to each row.
RANK()	Assigns ranking and handles duplicate salaries with the same rank.
DENSE_RANK()	Provides ranking without skipping numbers after duplicates.
PARTITION BY	Divides data into groups based on department.
ORDER BY	Sorts employees based on salary.

TextBooks

1. **dward Capriolo, Dean Wampler, and Jason Rutherglen**, *Programming Hive*, O'Reilly Media, First Edition, 2012.
2. Covers Hive architecture, HiveQL, tables, joins, partitions, and advanced querying.

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)
MCA DEPARTMENT

LECTURE NOTES

Subject Name: BIG DATA ANALYTICS

Year / Branch: IV B.TECH – VII SEMESTER

Regulation: R23

Prepared By: Mr. G YUVARAJU,

Assistant Professor

HBASE- ZOOKEEPER – SQOOP

Unit Overview

This unit introduces Apache HBase, Apache ZooKeeper, and Apache Sqoop, three essential technologies in the Hadoop ecosystem that support NoSQL data storage, distributed coordination, and data transfer between relational databases and Hadoop. It begins with the fundamentals of Apache HBase, including its concepts, architecture, data model, and basic operations. The unit explains the differences between HBase and traditional Relational Database Management Systems (RDBMS), highlighting HBase's ability to handle massive volumes of structured and semi-structured data with high scalability and real-time access.

The unit further introduces Apache ZooKeeper, a centralized coordination service used in distributed systems. It explains ZooKeeper architecture, services such as configuration management, synchronization, naming, leader election, and distributed locking, along with the process of building distributed applications using ZooKeeper.

Finally, the unit covers Apache Sqoop, a data transfer tool designed to efficiently import and export data between relational databases and Hadoop. Topics include importing databases into HDFS and Hive, working with imported data, importing large objects (LOBs), and exporting processed data back to relational databases. Together, HBase, ZooKeeper, and Sqoop provide a complete framework for storing, coordinating, and transferring Big Data efficiently in enterprise environments.

Learning Outcomes

After completing this unit, students will be able to:

Explain the concepts, architecture, and working principles of Apache HBase.

Differentiate between HBase and traditional Relational Database Management Systems (RDBMS).

Describe the HBase data model and perform basic HBase operations.

Explain the architecture and services provided by Apache ZooKeeper in distributed systems.

Demonstrate how ZooKeeper supports distributed application development through coordination and synchronization services.

Describe the architecture and working of Apache Sqoop for data migration between Hadoop and relational databases.

Perform database imports, manage imported data, import large objects (LOBs), and export Hadoop data back to relational databases using Sqoop**.**

Importance of Studying this Unit

Modern organizations generate enormous amounts of structured and semi-structured data that require scalable storage, efficient coordination, and seamless integration with existing enterprise databases. Apache HBase provides a highly scalable NoSQL database capable of handling billions of records with

real-time read and write capabilities, making it ideal for applications such as social media, online banking, IoT, and recommendation systems.

Apache ZooKeeper plays a critical role in managing distributed applications by providing reliable coordination, synchronization, leader election, and configuration management services, ensuring high availability and fault tolerance.

Apache Sqoop bridges the gap between traditional relational databases and Hadoop by enabling fast and efficient bulk data transfer for analytics and reporting. Understanding these technologies equips students with the knowledge required to build scalable Big Data solutions and prepares them for advanced topics such as Apache Spark, Kafka, Cassandra, Cloud Computing, Data Engineering, Data Warehousing, Machine Learning, and Distributed Systems. These skills are highly valuable for careers in Big Data Engineering, Data Analytics, Database Administration, Cloud Platforms, and Enterprise Data Integration.

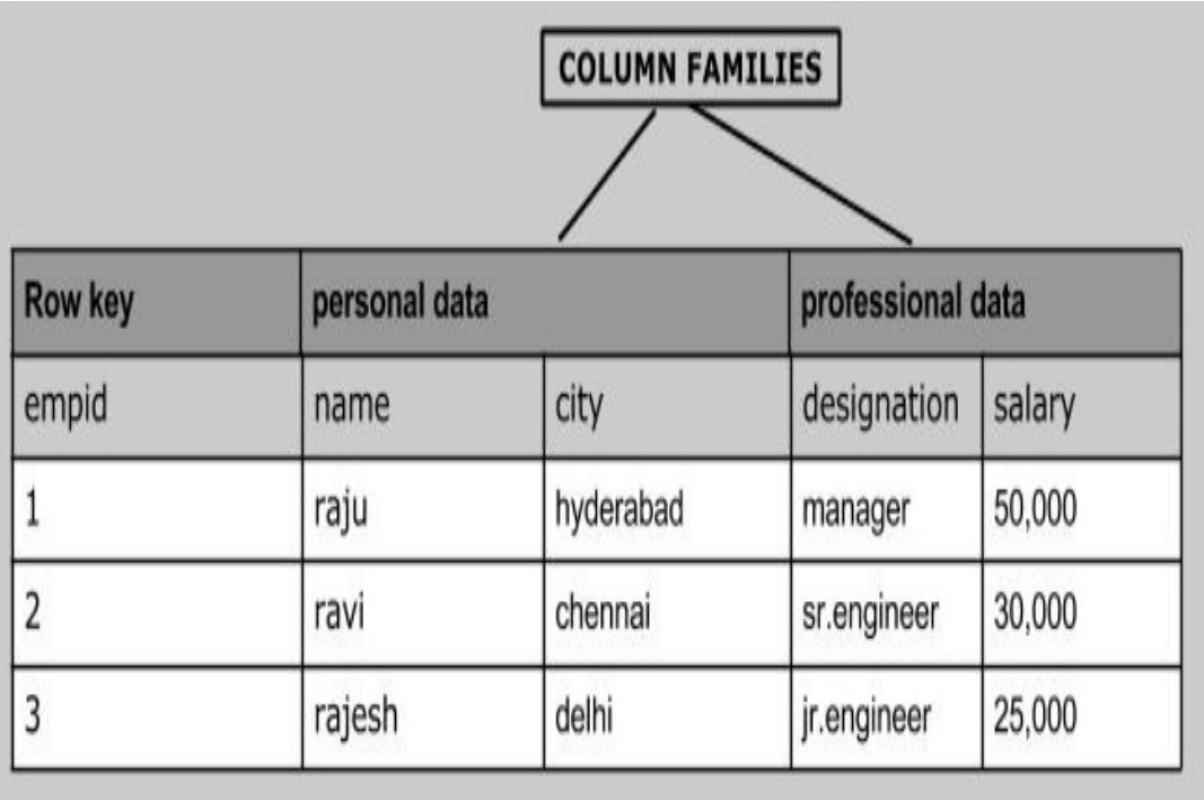
Introduction to Hbase

- HBase is distributed column-oriented database built on top of the Hadoop file system.
- HBase an Open-Source Non-Relational Distributed DB modeled, after Google's Bigtable and it is written in java.
- Its developed as part of Apache Software Foundation.
- HBase is data model that provide quick random access to huge amounts of structured data**.**
- Its part of the Hadoop ecosystem that provides random real-time read/write access to data in the HDFS either data can store directly in HDFS or through HBase.
- HBase sits on top of the Hadoop File System and provides read and write access.
- Data consumer reads/accesses the data in HDFS randomly using HBase.

Column Oriented vs. Row Oriented

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

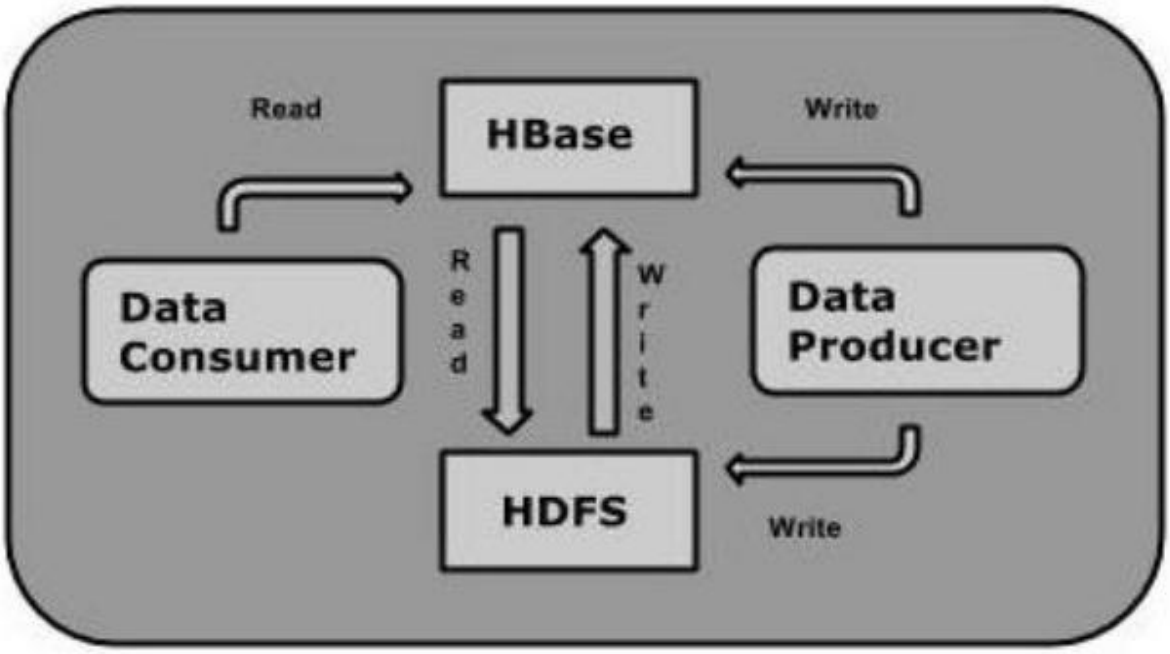
Example:



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

Example1:

Rowid	Column Family			Column Family			Column Family			Column Family		
	clo1	col 2	col 3	col 1	col 2	col 3	col1	col2	col3	col1	col2	col3
1												
2												
3												



Components of the Diagram

1. Data Producer

A **Data Producer** is any application or user that generates data and wants to store it in Hadoop.

Examples:

- Banking application
- E-commerce website
- IoT sensors
- Social media application

Function:

- Sends data to HBase.
- HBase stores the data in HDFS.

Data Flow:

Data Producer → HBase → HDFS

2. HBase

HBase is a **distributed, column-oriented NoSQL database** built on top of HDFS.

Functions of HBase:

- Accepts write requests from data producers.
- Retrieves data for data consumers.

- Provides random, real-time read and write access.
- Stores data physically in HDFS.

HBase acts as a **middleware** between applications and HDFS.

3. HDFS (Hadoop Distributed File System)

HDFS is the storage layer of Hadoop.

Functions:

- Stores large amounts of distributed data.
- Replicates data across multiple machines.
- Provides fault tolerance and reliability.

HBase uses HDFS to permanently store all data.

4. Data Consumer

A **Data Consumer** is an application or user that retrieves data.

Examples:

- Business Intelligence tools
- Data Analytics applications
- Web applications
- Reporting systems

Function:

- Sends read requests to HBase.
- HBase fetches the required data from HDFS and returns it.

Data Flow:

HDFS → HBase → Data Consumer

Working of the Diagram

Step 1: Writing Data

1. A **Data Producer** generates data.
2. The producer sends the data to **HBase**.
3. HBase processes the request.
4. HBase writes the data into **HDFS**.
5. The data is safely stored in distributed storage.

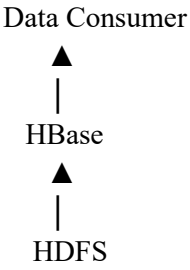
Flow:

Data Producer

Step 2: Reading Data

1. A **Data Consumer** requests data.
2. The request goes to **HBase**.
3. HBase locates the required data in **HDFS**.
4. HBase retrieves the data.
5. The data is returned to the consumer.

Flow:



Why HBase is Used?

HDFS alone is designed for **high-throughput, sequential access** to large files. It is **not suitable for fast random reads and writes**.

HBase adds the following capabilities on top of HDFS:

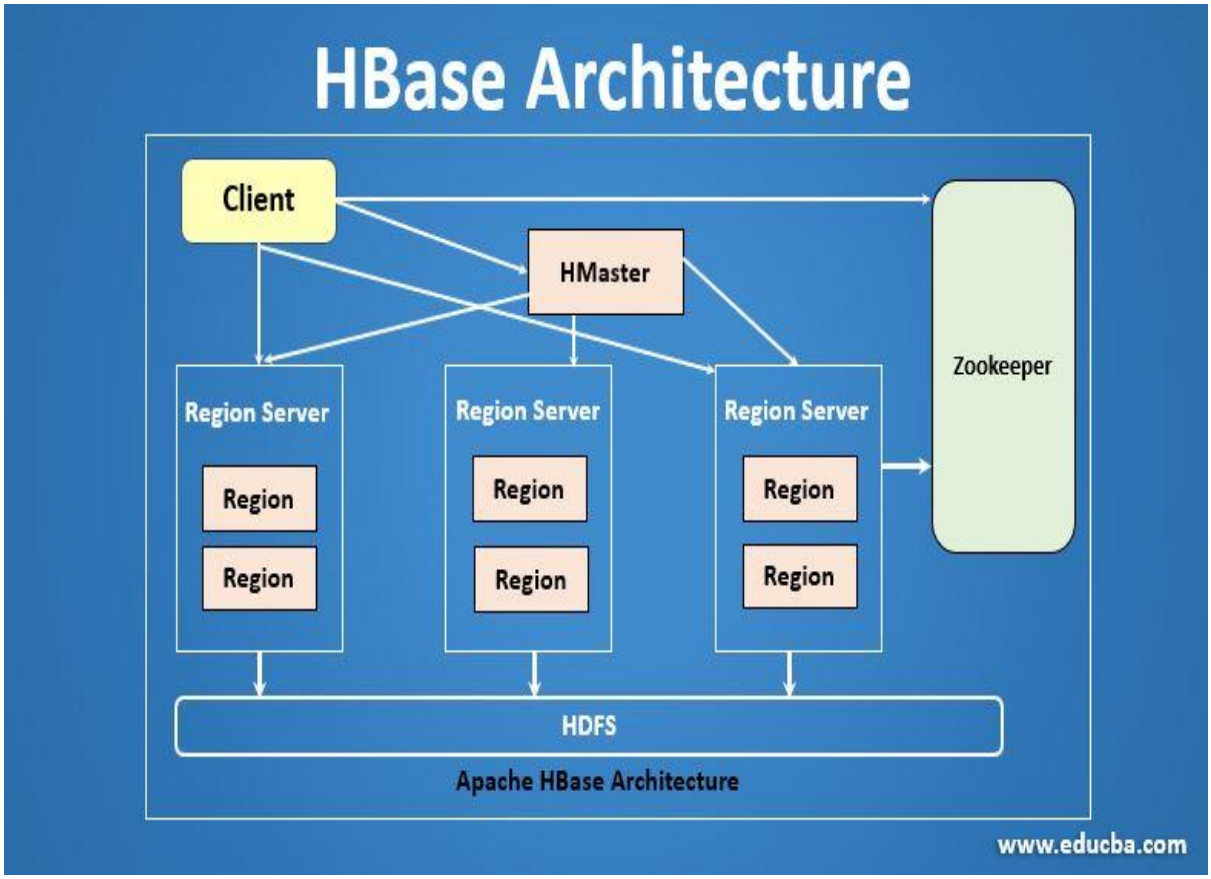
- Random access to data
- Real-time read and write operations
- Efficient storage of large tables
- Fast retrieval using row keys
- Automatic data partitioning and scalability

Thus, HBase enables applications to access data efficiently while leveraging HDFS for reliable storage.

HBase vs RDMS

Sr. No.	Key	RDBMS	HBase
1	Definition	RDBMS stands for Relational DataBase Management System.	HBase has no full form.
2	SQL	RDBMS requires SQL, Structured Query Language.	HBase does not need SQL.
3	Schema	RDBMS has a fixed schema.	HBase has no fixed schema.
4	Orientation	RDBMS is row oriented.	HBase is column oriented.
5	Scalability	RDBMS faces problems in scalability.	HBase is highly scalable.
6	Nature	DBMS is static in nature.	HBase is dynamic in nature.
7	Data Retrieval	RDBMS data retrieval is slow.	HBase data retrieval is fast.
8	RULE	RDBMS follows ACID(Atomicity, Consistency, Isolation and Durability) Rule.	HBase follows CAP(Consistency, Availability, and Partition-tolerance) Rule.
9	Data structure	RDBMS handles structural data.	HBase handles structural, non-structural and semi-structural data.
10	Sparse Data Handling	Sparse data handling is not present.	Sparse data handling is present.

HBase Architecture



Master Server

HBase has mainly three different parts are HMaster, RegionServer and the Zookeeper.

HMaster: HMaster in HBase is a process which helps to **assign the regions to region server**. It balances the loads by assigning the regions.

- HMaster helps to create, modify, and delete tables in the database.
- It also cares about different tasks **when the client wants to change the schema or metadata.**

  Region Server are the main working with nodes. It handles the Read, Write, and Modify requests from the clients.
- The region server runs on every node in the Hadoop clusters.
- Its read cache called **Block Cache,** read data are stored in the **read cache,** and when the cache is full, recently used data is removed. Another cache is present here called **MemStore.**
- Region Server are the main working with nodes. It handles the Read, Write, and Modify requests from the clients.
- The region server runs on every node in the Hadoop clusters.
- Its read cache called **Block Cache,** read data are stored in the **read cache,** and when the cache is full, recently used data is removed. Another cache is present here called **MemStore.**

HBase Data Model

The HBase data model is based on **Google Bigtable**.

Unlike relational databases, HBase stores data in **column families** instead of fixed columns.

Components of HBase Data Model

1. Table

A collection of rows

Example:

Student

2. Row Key

Each row has a unique identifier called the **Row Key**.

Example:

101
102
103

3. Column Family

A group of related columns.

Example:

Personal
Academic

4. Column Qualifier

A specific column inside a column family.

Example

Personal
Name
Age

Academic
CGPA

5. Cell

A cell is the intersection of:

- Row Key
- Column Family

- Column Qualifier

Each cell stores one value.

Example

Row Key =101

Personal:Name = Rahul

6. Timestamp

Every cell automatically stores a timestamp.

It is used for:

- Version control
- Data recovery
- Maintaining multiple versions of data

HBase Data Model Example

Row Key	Personal:Name	Personal:Age	Academic:CGPA
101	Rahul	20	8.9
102	Anita	21	9.2
103	Kiran	20	8.5

Here:

- Row Key → 101
- Column Family → Personal
- Column Qualifier → Name
- Cell Value → Rahul

HBase Operations

HBase provides basic **CRUD (Create, Read, Update, Delete)** operations using the HBase Shell.

1. Create Table

Creates a new table.

Syntax

create 'student','personal','academic'

2. List Tables

Displays all tables.

list

3. Describe Table

Displays the structure of a table.

describe 'student'

4. Insert Data (Put)

Adds data to a table.

Syntax

Put‘student','101','personal:name','Rahul'
put 'student','101','academic:cgpa','8.9'

5. Read Data (Get)

Retrieves a single row.

get 'student','101'

6. Scan Table

Displays all records in the table.

scan 'student'

7. Update Data

Updating is performed using another **put** command.

put 'student','101','academic:cgpa','9.1'

The old value is replaced with the new value (while older versions may still be retained based on version settings).

8. Delete a Column

Deletes a specific column value.

delete 'student','101','personal:name'

9. Delete All Columns in a Row

deleteall 'student','101'

10. Disable Table

A table must be disabled before it can be dropped.

disable 'student'

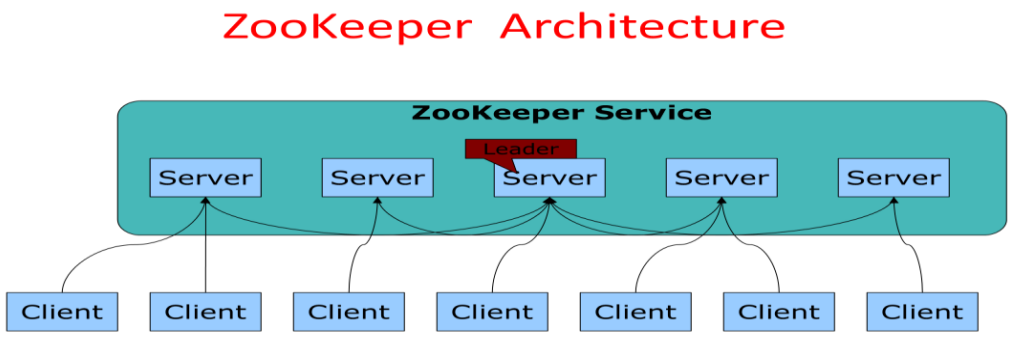
11. Drop Table

Deletes the table permanently.

drop 'student'

Introduction to ZooKeeper

1. ZooKeeper is an open source Apache project that provides a centralized service for providing configuration information, naming, synchronization and group services over large clusters in distributed systems.
2. The goal is to make these systems easier to manage with improved, more reliable.
3. Zookeeper is used in distributed systems to coordinate distributed processes and services.
4. Zookeeper is designed to be highly reliable and fault-tolerant, and it can handle high levels of read and write throughput.
5. Zookeeper is used in distributed systems to coordinate distributed processes and services.
6. Zookeeper is designed to be highly reliable and fault-tolerant, and it can handle high levels of read and write throughput.



- All servers store a copy of the data (in memory)
- A leader is elected at startup.
- All updates go through leader
- Update responses are sent when a majority of servers have persisted the change.

Client: Client is one of the nodes in the distributed application cluster.

- It helps you to accesses information from the server.
- Every client sends a message to the server at **regular intervals** that helps the server to know that the client is alive.

Server: The server sends an acknowledge(response) when any client connects.

- **In the case,** when there is no response from the connected server, the client automatically redirects the message to another server.

Leader: It gives all the information to the clients as well as an acknowledgment that the server is alive.

- It would performs automatic recovery if any of the connected nodes failed.

Follower:

- Server node which follows leader instruction is called a follower (who are all follows leader's instruction).
- The Client read requests are handled by the correspondingly connected **Zookeeper server.**
- The client writes requests are handled by the **Zookeeper leader.**

Ensemble/Cluster:

Group of Zookeeper servers which is called ensemble **or** a Cluster.

- You can use ZooKeeper infrastructure in the cluster mode to have the system at the optimal value when you are running the Apache.

ZooKeeper WebUI:

- If you want to work with ZooKeeper resource management, then you need to use WebUI.
- It allows work with ZooKeeper using the web user interface, instead of using the command line.
- It offers fast and effective communication with the ZooKeeper application.

Building Applications with Zookeeper

1. **Configuration Management**:

ZooKeeper can be used to store and manage configuration information for applications. Rather than hardcoding configuration values, applications can retrieve them from ZooKeeper, allowing for dynamic updates without the need for application restarts. ZooKeeper's watches can be used to receive notifications when configuration values change.

2. **Distributed Locking:**

ZooKeeper provides a distributed locking mechanism that can be used to coordinate access to shared resources among multiple processes or nodes. Applications can use ZooKeeper's lock implementation to ensure that only one process holds the lock at a time, preventing conflicts and ensuring data consistency.

3.Leader Election:

ZooKeeper can be used for leader election in distributed systems. Applications can utilize ZooKeeper's consensus algorithm to elect a leader among a group of nodes. The leader can then take on specific responsibilities or tasks while the other nodes act as followers.

4. Service Discovery:

ZooKeeper can serve as a service registry and discovery mechanism. Applications can register themselves as znodes in ZooKeeper, providing information about their availability and endpoints. Other applications can then discover and interact with these services by querying ZooKeeper.

6. **Distributed Queues**:

ZooKeeper can be used to implement distributed queues, where multiple processes or nodes can enqueue and dequeue elements. This is useful for scenarios where you need to distribute work among multiple workers or ensure ordered processing of tasks across multiple nodes.

7. Name Services:

ZooKeeper can act as a central name service, maintaining consistent naming information for distributed systems. Applications can use ZooKeeper to store and retrieve naming information such as node addresses or endpoints, enabling dynamic discovery and interaction with resources.

8. **Fault-tolerant Systems:**

ZooKeeper's fault-tolerant and highly available nature makes it well-suited for building fault-tolerant systems. Applications can rely on ZooKeeper to provide consistency and resilience, allowing them to recover from failures and continue functioning smoothly.

Apache Sqoop

Introduction to Apache Sqoop

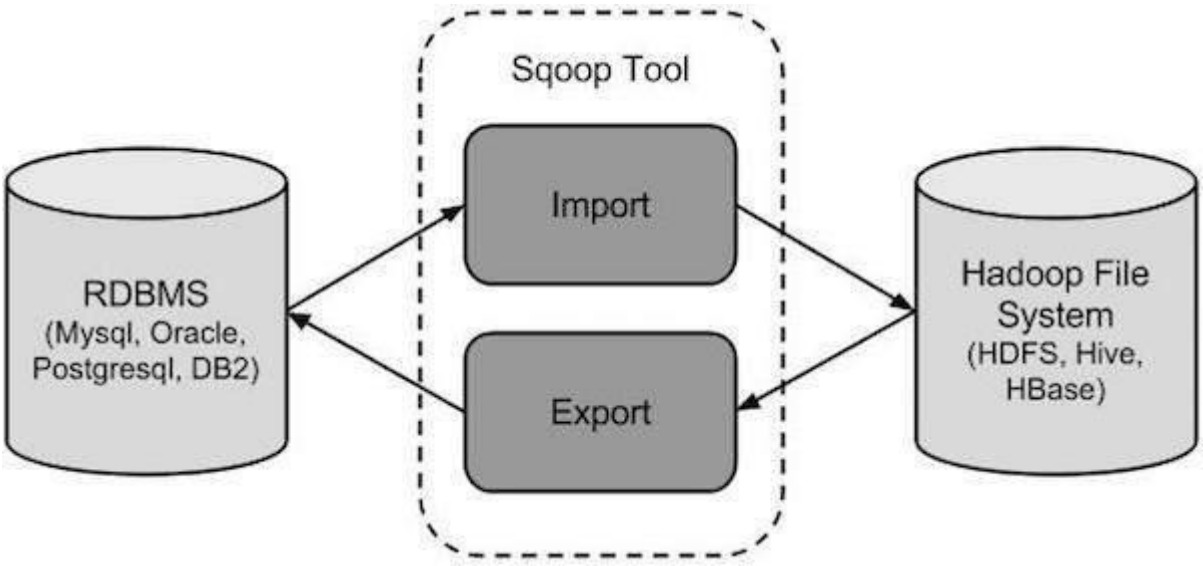
Apache Sqoop (SQL-to-Hadoop) is an open-source tool in the Hadoop ecosystem used to efficiently transfer bulk data between Relational Database Management Systems (RDBMS) and the Hadoop ecosystem. It automates the process of importing data from databases such as MySQL, Oracle, SQL Server, PostgreSQL, and DB2 into Hadoop and exporting processed data back to these databases.

Sqoop uses JDBC (Java Database Connectivity) to connect to relational databases and employs MapReduce for parallel data transfer, making it fast and efficient for handling large datasets.

Definition

Apache Sqoop is a command-line tool that transfers structured data between relational databases and Hadoop components such as HDFS, Hive, and HBase.

Sqoop = SQL + Hadoop



Theory of Import and Export

Import

Import is the process of copying data from a Relational Database (RDBMS) into the Hadoop ecosystem (HDFS, Hive, or HBase).

Flow:

MySQL → Sqoop → HDFS

Export

Export is the process of copying processed data from Hadoop back into a relational database.

Flow:

HDFS → Sqoop → MySQL

Practical Implementation

Step 1: Login to MySQL

Theory

Before importing data, a connection must be established with the MySQL server.

Command

mysql -u root -p

Enter the password:

Password: root

Step 2: Create Database and Table

Theory

A database stores related tables. Here, we create a database named yuvaraj and an employee table containing employee details.

Commands

CREATE DATABASE yuvaraj;

USE yuvaraj;

```
CREATE TABLE emp(
eid INT,
ename VARCHAR(20),
esal INT,
eloc VARCHAR(20)
);
```

Insert sample data.

```
INSERT INTO emp VALUES
(100,'yuva',10000,'rct'),
(101,'ram',15000,'chennai'),
(102,'kiran',20000,'hyderabad'),
(103,'anitha',25000,'bangalore');
```

Step 3: Display the Table

The SELECT statement is used to verify whether the data has been inserted successfully.

Command

SELECT * FROM emp;

Step 4: Open Another Terminal

Keep the MySQL session active. Open another terminal to execute Sqoop commands.

Importing Data

1. Import Entire Table

The import command copies all records from the MySQL table into HDFS.

```
sqoop import \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--table emp \
-m 1 \
--target-dir /yuvarajdir/sqoop1
```

Explanation

- --connect : Database connection URL
- --username : Database username
- --password : Database password
- --table : Table to import
- -m 1 : Uses one mapper
- --target-dir : Destination directory in HDFS

2. Import Using WHERE Clause

Only selected rows satisfying the condition are imported.

Command

```
sqoop import \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--table emp \
-m 1 \
```

```
--where "esal > 10000" \
--target-dir /yuvarajdir/sqoop1
```

Only employees with salary greater than 10000 are imported.

3. Import Using Tab Delimiter

By default, fields are comma-separated. The delimiter can be changed.

```
sqoop import \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--table emp \
-m 1 \
--where "esal > 10000" \
--fields-terminated-by '\t' \
--target-dir /yuvarajdir/sqoop1
```

4. Import Using SQL Query

Instead of importing the entire table, custom SQL queries can be executed.

```
sqoop import \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--query "SELECT * FROM emp WHERE esal > 10000 AND \$CONDITIONS" \
-m 1 \
--target-dir /yuvarajdir/sqoop1
```

Note: The variable $CONDITIONS is mandatory in all Sqoop query imports.

Import All Tables

Imports every table from the selected database.

Command

```
sqoop import-all-tables \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--warehouse-dir /yuvarajdir/sqoop1
```

Excluding Tables

Sometimes certain tables are not required during import. Sqoop allows excluding specific tables.

Command

```
sqoop import-all-tables \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--warehouse-dir /yuvarajdir/sqoop1 \
--exclude-tables dummy
```

Incremental Import

Incremental import copies only newly inserted records instead of importing the entire table repeatedly.

Command

```
sqoop import \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--table emp \
--incremental append \
--check-column eid \
--last-value 790 \
-m 1 \
--target-dir /yuvarajdir/sqoop2
```

Explanation

- append : Imports newly appended records
- check-column : Column used to identify new records
- last-value : Last imported value

List Databases

Displays all databases available in the MySQL server.

Command

```
sqoop list-databases \
--connect jdbc:mysql://localhost/ \
--username root \
--password root
```

List Tables

Displays all tables present in the selected database.

```
sqoop list-tables \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root
```

Export Data

Theory

After processing data in Hadoop, it can be exported back into MySQL.

Command

```
sqoop export \
--connect jdbc:mysql://localhost/yuvaraj \
--username root \
--password root \
--table emp \
--export-dir /yuvarajdir/sqoop1
```

TEXT BOOKS:

1. Big Data, Big Analytics: Emerging, Michael Minnelli, Michelle Chambers, and AmbigaDhiraj

REFERENCE BOOKS:

1. "HBase: The Definitive Guide", O'Reilley, Lars George,2011

Case Study 1: HBase – Real-Time Data Storage and Processing at Facebook

Title :

Case Study: Using Apache HBase for Real-Time Messaging Data Management at Facebook

Problem Statement

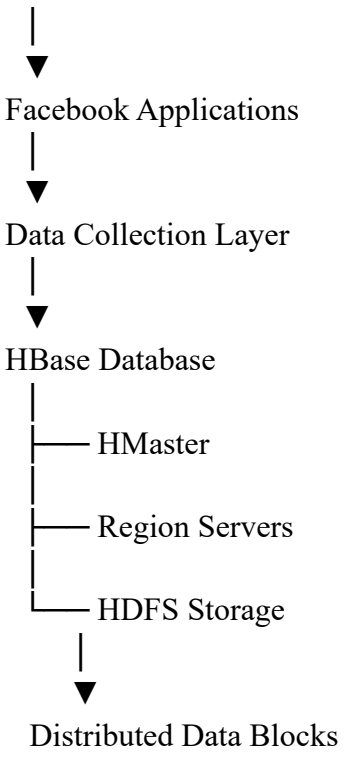
Facebook faced several challenges:

- Billions of users generating huge volumes of data every day.
- Need for real-time access to user messages and activity data.
- Traditional relational databases could not scale horizontally.
- Requirement for fault-tolerant and distributed storage.
- Need to handle millions of read and write operations per second.

Solution Using HBase Architecture

Facebook implemented HBase to store large-scale data efficiently.

Data Flow Architecture



Role of HBase Components

1. HMaster

- Manages HBase tables.
- Assigns regions to Region Servers.
- Handles load balancing.

2. Region Server

- Stores and manages data regions.
- Performs read/write operations.

3. HDFS

- Provides reliable distributed storage.
- Stores HBase data files.

4. ZooKeeper

- Provides coordination between HBase components.
- Maintains server status information.

Implementation

Facebook stored:

- User profiles.
- Messages.
- Activity logs.
- Timeline information.
- Real-time analytics data.
- Example HBase Table:

Row Key	User	Activity	Time
101	John	Posted Photo	10:30 AM
102	Mary	Sent Message	10:35 AM
103	David	Liked Post	10:40 AM

Benefits Achieved

- Fast data retrieval.
- Horizontal scalability.
- Fault tolerance.
- Real-time processing capability.
- Handles petabytes of data.

Case Study 2: ZooKeeper – Distributed Coordination at Yahoo!

Title

Case Study: Using Apache ZooKeeper for Managing Distributed Hadoop Services at Yahoo!

Problem Statement

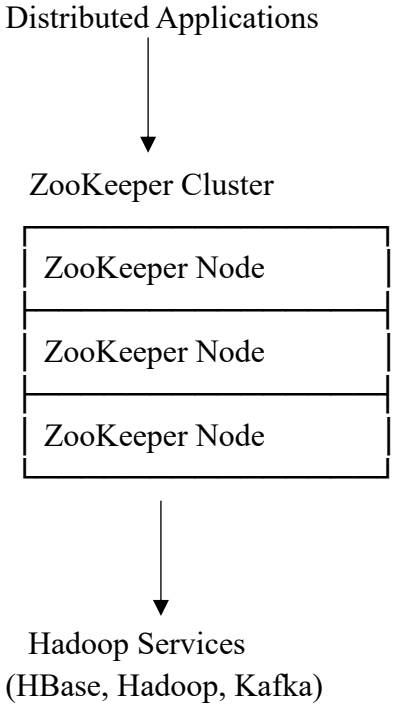
Yahoo faced challenges:

- Thousands of servers running distributed applications.
- Difficulty maintaining configuration information.
- Need for automatic failure detection.
- Requirement for coordination between multiple services.
- Need to select master nodes automatically.

Solution Using ZooKeeper

ZooKeeper was deployed as a coordination layer.

Architecture



ZooKeeper Functions

1. Leader Election

Example:

Server A
Server B
Server C

|
ZooKeeper chooses
▼
Leader Server

If the leader fails, ZooKeeper automatically selects a new leader.

2. Configuration Management

Stores:

- Cluster information.
- Server locations.
- Application settings.

3. Failure Detection

ZooKeeper monitors distributed nodes.

Example:

- Node 1 ✓
- Node 2 ✓
- Node 3 ✗

ZooKeeper detects failure

Applications of ZooKeeper

Used with:

- HBase
- Hadoop YARN
- Apache Kafka
- Apache Solr
- Distributed databases

Case Study 3: Sqoop – Data Migration at LinkedIn

Title

Case Study: Using Apache Sqoop for Data Transfer Between RDBMS and Hadoop at LinkedIn

Problem Statement

LinkedIn needed to:

- Transfer huge volumes of database records into Hadoop.
- Perform large-scale analytics.
- Combine traditional databases with big data platforms.
- Reduce manual data movement.

Solution Using Sqoop

Sqoop provided automated data transfer between:

- MySQL
- Oracle Database
- PostgreSQL
- Hadoop HDFS
- Hive
- HBase

Data Import Example

Import Employee Data from MySQL to HDFS

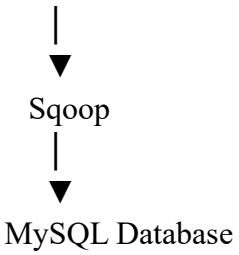
Command:

```
sqoop import \
--connect jdbc:mysql://server/company \
--username root \
--password password \
--table employee \
--target-dir /employee_data
```

Data Export Example

Moving Hadoop results back to database:

Hive Analysis Result



LinkedIn Data Processing Flow

- Users
- LinkedIn Applications
- Operational Databases
- Sqoop Import
- Hadoop HDFS
- Hive/Spark Analytics
- Recommendations and Insights

Comparison Summary

Technology	Organization Example	Main Purpose
HBase	Facebook	Real-time NoSQL storage
ZooKeeper	Yahoo	Distributed coordination
Sqoop	LinkedIn	Data transfer between RDBMS and Hadoop

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)
MCA DEPARTMENT

LECTURE NOTES

Subject Name: BIG DATA ANALYTICS

Year / Branch: IV B.TECH – VII SEMESTER

Regulation: R23

Prepared By: Mr. G YUVARAJU,

Assistant Professor

APACHE SPARK

Unit Overview

This unit introduces Apache Spark, a powerful open-source distributed computing framework designed for fast and efficient processing of large-scale data. Unlike traditional Hadoop MapReduce, Spark performs computations in memory, significantly improving processing speed for batch processing, real-time analytics, machine learning, graph processing, and interactive applications.

The unit begins with the fundamentals of Apache Spark, including its architecture, advantages over Hadoop MapReduce, lazy evaluation, in-memory processing, and Directed Acyclic Graph (DAG)-based execution model. Students will understand how Spark optimizes data processing through Spark Context and Spark Session, which serve as entry points for developing Spark applications.

The unit further explores Resilient Distributed Datasets (RDDs), Spark's fundamental distributed data abstraction, along with transformations, actions, and the differences between narrow and wide transformations. It then introduces Spark DataFrames, schema-based distributed datasets that enable efficient SQL-like operations. Students will learn to convert RDDs into DataFrames, apply DataFrame transformations, and understand the role of the Catalyst Optimizer in improving query performance.

Finally, the unit covers advanced DataFrame operations, including working with dates and timestamps, handling null values, managing complex data types and JSON data, grouping and aggregation, window functions, joins, multiple data sources, broadcast variables, and accumulators. These concepts provide students with practical knowledge for developing scalable, high-performance data processing applications in modern Big Data environments.

Learning Outcomes

After completing this unit, students will be able to:

1. Explain the architecture, components, and working principles of Apache Spark.
2. Compare Apache Spark with Hadoop MapReduce and justify Spark's advantages for Big Data processing.
3. Describe lazy evaluation, in-memory processing, and Directed Acyclic Graph (DAG) execution in Spark.
4. Create and manipulate Resilient Distributed Datasets (RDDs) using transformations and actions.
5. Differentiate between narrow and wide transformations and analyze their impact on performance.
6. Create and manipulate Spark DataFrames and convert RDDs into DataFrames.
7. Explain the role of the Catalyst Optimizer in query optimization.

8. Apply DataFrame transformations for filtering, sorting, grouping, aggregation, and joining datasets.
9. Process dates, timestamps, null values, complex data types, and JSON data using Spark SQL functions.
10. Implement window functions, broadcast variables, accumulators, and data source operations for efficient distributed data processing.

Importance of Studying this Unit

Modern organizations process enormous volumes of structured, semi-structured, and unstructured data that require fast, scalable, and fault-tolerant processing frameworks. Apache Spark addresses these requirements by providing an in-memory distributed computing engine capable of processing data significantly faster than traditional Hadoop MapReduce.

Spark has become one of the most widely used technologies for Big Data analytics, data engineering, machine learning, real-time streaming, cloud computing, and artificial intelligence applications. Its rich APIs and optimized execution engine simplify the development of complex data processing pipelines while improving overall system performance.

Understanding Apache Spark enables students to design scalable data analytics solutions, optimize distributed computations, and efficiently process data from diverse sources such as HDFS, Hive, relational databases, cloud storage, and JSON files. These skills form the foundation for advanced technologies including Apache Kafka, Apache Flink, Databricks, Delta Lake, Apache Iceberg, Machine Learning with Spark MLlib, Cloud Data Engineering, Data Warehousing, and Artificial Intelligence.

The concepts learned in this unit are highly valuable for careers in Big Data Engineering, Data Engineering, Data Analytics, Machine Learning Engineering, Cloud Computing, Software Development, Business Intelligence, and Enterprise Data Processing.

Apache Spark

Introduction

Apache Spark is an open-source distributed computing framework developed for processing large volumes of data quickly and efficiently. Unlike Hadoop MapReduce, which stores intermediate results on disk, Spark performs most computations in memory (RAM). This significantly improves performance and makes Spark suitable for real-time analytics, machine learning, and interactive applications.

Although Spark can run independently, it is often integrated with the Hadoop ecosystem to leverage Hadoop Distributed File System (HDFS) for storage while replacing MapReduce for processing.

Advantages of Apache Spark over Hadoop MapReduce

1. In-Memory Processing

Spark stores intermediate data in memory (RAM) instead of writing it to disk after every operation.

Hadoop MapReduce: Uses disk-based processing, which increases execution time.

Spark: Uses in-memory processing, resulting in much faster computation.

Advantage:

- Faster execution
- Reduced disk I/O
- Better performance for iterative applications

2. Faster Performance

Spark is significantly faster than Hadoop MapReduce because it minimizes disk access.

- Up to 100 times faster for in-memory processing.
- Around 10 times faster than Hadoop when processing data stored on disk.

Applications:

- Real-time analytics
- Machine learning
- Interactive queries

3. Supports Multiple Workloads

Hadoop MapReduce mainly supports batch processing.

Spark supports multiple workloads using a single framework.

These include:

- Batch Processing
- Real-Time Stream Processing
- Machine Learning
- Graph Processing
- SQL Queries

This eliminates the need to use multiple tools.

4. Easy Programming

Spark provides high-level APIs in multiple programming languages.

Supported languages include:

- Python (PySpark)
- Java
- Scala
- R

Developers can write fewer lines of code compared to Hadoop MapReduce.

5. Lazy Evaluation

Spark does not execute transformations immediately.

Instead:

1. Records all transformations.
2. Creates an execution plan (DAG).
3. Executes only when an action is called.

This optimization improves execution efficiency.

6. DAG Execution Engine

Spark uses a Directed Acyclic Graph (DAG) scheduler.

Advantages include:

- Better task optimization
- Reduced execution time
- Improved fault recovery
- Efficient resource utilization

Hadoop MapReduce executes jobs in fixed Map and Reduce stages, whereas Spark supports more flexible execution plans**.**

7. Supports Interactive Processing

Spark enables users to perform interactive queries quickly because data is processed in memory.

Suitable for:

- Business Intelligence (BI)
- Dashboards
- Data Exploration
- Ad-hoc Queries

Hadoop MapReduce has high latency and is not suitable for interactive analysis.

8. Better Machine Learning Support

Spark provides MLlib, a built-in Machine Learning library.

It includes algorithms for:

- Classification
- Regression
- Clustering
- Recommendation
- Dimensionality Reduction

Hadoop requires additional frameworks like Mahout.

9. Real-Time Stream Processing

Spark supports real-time data processing using Spark Streaming and Structured Streaming.

Applications include:

- Online Banking
- Fraud Detection
- Social Media Analytics
- IoT Sensor Data
- Stock Market Analysis

Hadoop MapReduce is designed only for offline batch processing.

10. Graph Processing

Spark includes GraphX, which simplifies graph analytics.

Applications:

- Social Networks
- Road Networks
- Recommendation Systems
- Network Analysis

Hadoop requires external graph-processing tools**.**

11. Fault Tolerance

Spark uses Resilient Distributed Datasets (RDDs) to recover lost data automatically.

Advantages:

- Reliable processing
- Automatic recovery
- High availability

12. Unified Framework

Spark combines several data processing capabilities into a single platform.

Modules include:

- Spark Core
- Spark SQL
- Spark Streaming
- MLlib

- GraphX

Hadoop requires separate components such as MapReduce, Hive, Pig, Mahout, and Giraph.

13. Better Resource Utilization

Spark efficiently utilizes:

- CPU
- Memory
- Network resources

This improves cluster performance and reduces overall execution time.

14. Compatibility with Hadoop

Spark can work seamlessly with the Hadoop ecosystem.

It supports:

- HDFS
- Hive
- HBase
- YARN
- Apache Kafka
- Cassandra

Organizations can upgrade their processing engine without changing the storage layer.

Lazy Evaluation in Apache Spark

Definition

Lazy Evaluation is an optimization technique in Apache Spark where transformations are not executed immediately. Instead, Spark records all the transformations and creates an execution plan. The actual computation begins only when an Action (such as collect(), count(), or save()) is invoked.

In simple terms, Spark delays execution until the result is required, allowing it to optimize the entire workflow before processing the data.

How Lazy Evaluation Works

1. A user creates an RDD or DataFrame.
2. Transformations such as map(), filter(), and flatMap() are applied.

3. Spark stores these transformations in a Directed Acyclic Graph (DAG).
4. No computation occurs until an Action is called.
5. When an Action is executed, Spark optimizes the DAG and performs all transformations efficiently.

DAG (Directed Acyclic Graph) in Apache Spark

Definition

A **Directed Acyclic Graph (DAG)** is the execution plan used by Apache Spark to process data efficiently. It is a graph consisting of **vertices (operations)** and **edges (dependencies)**, where data flows in one direction and no cycles (loops) are allowed.

Spark creates a DAG from all the transformations applied to an RDD or DataFrame. The DAG is then optimized before execution, resulting in faster and more efficient data processing.

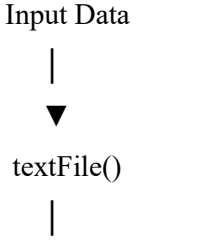
Components of a DAG

- **Vertices (Nodes):** Represent transformations or computations.
- **Edges:** Represent dependencies between operations.
- **Stages:** Groups of tasks separated by shuffle operations.
- **Tasks:** Individual units of work executed on worker nodes.

Working of DAG

1. User creates an RDD or DataFrame.
2. Transformations such as map() and filter() are applied.
3. Spark records all transformations without executing them.
4. Spark creates a DAG showing the dependencies.
5. When an action such as collect() or count() is called, the DAG Scheduler optimizes the execution plan.
6. Tasks are executed in parallel across the cluster.

DAG Execution Flow



▼

map()

|

▼

filter()

|

▼

reduceByKey()

|

▼

collect()

RDD (Resilient Distributed Dataset)

Definition

RDD (Resilient Distributed Dataset) is the fundamental data structure in Apache Spark. It is a **distributed**, **immutable**, and **fault-tolerant** collection of objects that can be processed in parallel across multiple nodes in a cluster.

RDD is the core abstraction of Spark and forms the foundation for distributed data processing.

Developed by: Apache Spark

Full Form: Resilient Distributed Dataset

Characteristics of RDD

- **Resilient:** Automatically recovers lost data using lineage information.
- **Distributed:** Data is partitioned and stored across multiple nodes.
- **Immutable:** Once created, an RDD cannot be modified. Any transformation creates a new RDD.
- **Fault Tolerant:** Lost partitions can be recomputed automatically.
- **Parallel Processing:** Data is processed simultaneously on multiple machines.
- **Lazy Evaluation:** Transformations are executed only when an action is called.

Creating an RDD

1. From a Collection

```
from pyspark import SparkContext
```

```
sc = SparkContext("local", "RDDExample")

data = [10, 20, 30, 40, 50]

rdd = sc.parallelize(data)

print(rdd.collect())
```

Output

```
[10, 20, 30, 40, 50]
```

2. From a Text File

```
rdd = sc.textFile("employee.txt")

print(rdd.collect())
```

RDD Operations

RDD operations are classified into two categories:

1. **Transformations**
2. **Actions**

Transformations

Definition

Transformations are operations that create a **new RDD** from an existing RDD. They are **lazy**, meaning Spark records them but does not execute them until an action is called.

Common Transformations

- map()
- filter()
- flatMap()
- distinct()
- union()
- groupByKey()
- reduceByKey()
- sortByKey()

- join()

Types of Transformations

Transformations are classified into:

1. Narrow Transformations
2. Wide Transformations

Narrow Transformations

Definition

A **Narrow Transformation** is a transformation in which each child partition depends on **only one parent partition**. No data is transferred (shuffled) between partitions.

Because there is **no shuffle**, narrow transformations execute quickly.

Characteristics

- One-to-one partition dependency.
- No data movement across partitions.
- Faster execution.
- Lower network communication.
- Better performance.

Diagram

Parent Partitions Child Partitions

Partition 1 ——————► Partition 1

Partition 2 ——————► Partition 2

Partition 3 ——————► Partition 3

Examples

map()

```
rdd = sc.parallelize([1,2,3,4])

result = rdd.map(lambda x: x*2)
```

```
print(result.collect())
```

Output

```
[2, 4, 6, 8]
```

filter()

```
rdd = sc.parallelize([10,20,30,40])

result = rdd.filter(lambda x:x>20)

print(result.collect())
```

Output

```
[30, 40]
```

Other Narrow Transformations

- map()
- filter()
- flatMap()
- mapPartitions()
- union()

Advantages

- Faster execution.
- No shuffle operation.
- Less network overhead.
- Efficient resource utilization.

Wide Transformations

Definition

A **Wide Transformation** is a transformation in which a child partition depends on **multiple parent partitions**. Data must be **shuffled** across the cluster before processing.

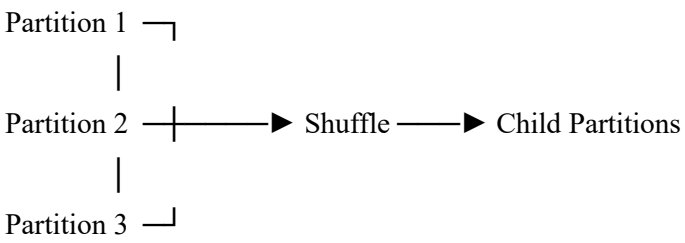
Shuffle is the process of redistributing data between partitions.

Characteristics

- Many-to-many partition dependency.
- Data is transferred between executors.
- Requires shuffle.
- Slower than narrow transformations.
- Higher network communication.

Diagram

Parent Partitions



Examples

reduceByKey()

```
rdd = sc.parallelize([
("A",10),
("A",20),
("B",30)
])

result = rdd.reduceByKey(lambda x,y:x+y)

print(result.collect())
```

Output

```
[('A', 30), ('B', 30)]
```

groupByKey()

rdd.groupByKey()

join()

rdd1.join(rdd2)

Other Wide Transformations

- groupByKey()
- reduceByKey()
- sortByKey()
- distinct()
- join()
- cogroup()

1. DataFrames

Definition

A **DataFrame** is a distributed collection of data organized into **rows and named columns**, similar to a table in a relational database or an Excel spreadsheet. It is one of the most widely used data structures in Apache Spark because it provides better performance and supports SQL-like operations.

A DataFrame is built on top of RDDs and uses Spark SQL's optimization engine (Catalyst Optimizer) for efficient execution.

Features of DataFrames

- Distributed collection of data
- Organized into rows and columns
- Schema-based structure
- Supports SQL queries
- Optimized using Catalyst Optimizer
- Fault-tolerant
- Supports multiple data sources

Advantages

- Faster than RDDs

- Easy to use
- Supports SQL
- Better memory management
- Automatic query optimization

Creating a DataFrame

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("DataFrameExample").getOrCreate()

data = [
   (101,"Ram",50000),
   (102,"Hari",60000)
]

df = spark.createDataFrame(data,["ID","Name","Salary"])

df.show()
```

2. RDD to DataFrame

Definition

Spark allows conversion of an **RDD into a DataFrame** by assigning column names (schema). DataFrames are preferred because they provide better optimization and SQL support.

Example

```
rdd = spark.sparkContext.parallelize([
(101,"Ram",50000),
(102,"Hari",60000)
])

df = rdd.toDF(["ID","Name","Salary"])

df.show()
```

Advantages

- Faster execution
- SQL support

- Schema-based processing
- Better optimization

3. Catalyst Optimizer

Definition

The **Catalyst Optimizer** is Spark SQL's built-in query optimization engine. It automatically analyzes and optimizes SQL queries and DataFrame operations before execution.

Functions

- Query analysis
- Logical optimization
- Physical optimization
- Code generation

Working

SQL Query

↓

Logical Plan

↓

Optimized Logical Plan

↓

Physical Plan

↓

Execution

Advantages

- Faster execution
- Automatic optimization
- Efficient memory usage
- Improved query performance

4. DataFrame Transformations

Definition

DataFrame transformations create a **new DataFrame** from an existing DataFrame without modifying the original one.

Common Transformations

select()

Select required columns.

df.select("Name","Salary").show()

filter()

Filter rows.

df.filter(df.Salary > 50000).show()

orderBy()

Sort data.

df.orderBy("Salary").show()

withColumn()

Add a new column.

from pyspark.sql.functions import col

df.withColumn("Bonus",col("Salary")*0.10).show()

withColumnRenamed()

Rename a column.

df.withColumnRenamed("Salary","Income").show()

drop()

Remove a column.

df.drop("Salary").show()

5. Working with Dates and Timestamps

Spark provides built-in functions for handling date and time values.

Current Date

from pyspark.sql.functions import current_date

df.select(current_date()).show()

Current Timestamp

from pyspark.sql.functions import current_timestamp

df.select(current_timestamp()).show()

Other Date Functions

- year()
- month()
- dayofmonth()
- date_add()
- datediff()

Example:

from pyspark.sql.functions import year

df.select(year("JoiningDate")).show()

6. Working with Nulls in Data

Missing values are represented as **NULL**.

Remove Null Values

df.na.drop().show()

Replace Null Values

df.na.fill("Unknown").show()

Replace Numeric Nulls

df.na.fill(0).show()

Example

Before

ID Name Salary

101 Ram 50000

102 NULL 60000

103 Hari NULL

After fill()

ID Name Salary

101 Ram 50000

102 Unknown 60000

103 Hari 0

9. Grouping

Grouping combines records having the same values.

Spark uses **groupBy()**.

Example

```
df.groupBy("Department")\
.sum("Salary")\
.show()
```

Aggregate Functions

- sum()
- avg()

- count()
- max()
- min()

Output

Department Total Salary

HR 120000

IT 150000

10. Window Functions

Definition

Window functions perform calculations across a group of rows while keeping each row in the result.

Unlike groupBy(), window functions do **not** reduce the number of rows.

row_number()

```
from pyspark.sql.window import Window

from pyspark.sql.functions import row_number

window = Window.partitionBy("Department").orderBy("Salary")

df.withColumn(
"RowNo",
row_number().over(window)
).show()
```

rank()

```
from pyspark.sql.functions import rank

df.withColumn(
"Rank",
rank().over(window)
).show()
```

dense_rank()

```
from pyspark.sql.functions import dense_rank


df.withColumn(
"DenseRank",
dense_rank().over(window)
).show()
```

11. Joins

Joins combine data from two or more DataFrames.

Types of Joins

Inner Join

Returns matching records.

```
emp.join(dept,"DeptID").show()
```

Left Join

Returns all records from the left table.

```
emp.join(dept,"DeptID","left").show()
```

Right Join

Returns all records from the right table.

```
emp.join(dept,"DeptID","right").show()
```

Full Outer Join

Returns all records from both tables.

```
emp.join(dept,"DeptID","outer").show()
```

12. Data Sources

Spark supports multiple data sources.

Supported Sources

- CSV
- JSON
- Parquet

- ORC
- Text Files
- Hive
- HDFS
- JDBC
- MySQL
- PostgreSQL
- Amazon S3

Read CSV

```
df = spark.read.csv(
"employee.csv",
header=True,
inferSchema=True
)
```

Write CSV

```
df.write.csv("output")
```

13. Broadcast Variables

Definition

Broadcast Variables distribute a **read-only variable** to all worker nodes efficiently.

Instead of sending a copy with every task, Spark sends it only once.

Example

```
lookup = {
"A":"HR",
"B":"IT"
}

broadcastVar = spark.sparkContext.broadcast(lookup)

print(broadcastVar.value)
```

Advantages

- Reduces network communication
- Improves join performance
- Saves memory

14. Accumulators

Definition

Accumulators are shared variables used to **collect information from worker nodes** back to the driver program. Workers can **add** values to an accumulator, but only the driver can read its final value.

They are commonly used for counters and statistics.

Example

```
acc = spark.sparkContext.accumulator(0)


def add(x):
    global acc
    acc += x

rdd.foreach(add)


print(acc.value)
```

TextBook

1. **SPARK: The Definitive Guide, Bill Chambers &MateiZaharia, O'Reilley, 2018Edition 3. Business Intelligence and Analytic Trends for Today's Businesses", Wiley,2013**