PERIYAR UNIVERSITY

(NAAC 'A++' Grade with CGPA 3.61 (Cycle - 3)

State University - NIRF Rank 56 - State Public University Rank 25

SALEM - 636 011

CENTRE FOR DISTANCE AND ONLINE EDUCATION

(CDOE)

MASTER OF COMPUTER APPLICATIONS

SEMESTER - II

CORE COURSE: BIG DATA ANALYTICS

(Candidates admitted from 2024 onwards)

PERIYAR UNIVERSITY

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

M. C. A. 2024 admission onwards

22UPCSC1C09 Core VIII : BIG DATA ANALYTICS

Prepared by:

Centre for Distance and Online Education (CDOE)
Periyar University
Salem - 636011

Syllabus

UNIT No	CONTENTS
1	Big Data and Analytics: Classification of Digital Data: Structured Data- Semi Structured Data and Unstructured Data. Introduction to Big Data: Characteristics - Evolution - Definition - Challenges with Big Data - Other Characteristics of Data - Big Data - Traditional Business Intelligence versus Big Data - Data Warehouse and Hadoop. Environment Big Data Analytics: Classification of Analytics - Challenges - Big Data Analytics important - Data Science - Data Scientist - Terminologies used in Big Data Environments - Basically Available Soft State Eventual Consistency - Top Analytics Tools
2	Technology Landscape: NoSQL, Comparison of SQL and NoSQL, Hadoop -RDBMS Versus Hadoop - Distributed Computing Challenges - Hadoop Overview - Hadoop Distributed File System - Processing Data with Hadoop - Managing Resources and Applications with Hadoop YARN - Interacting with Hadoop Ecosystem
3	Mongodb and Mapreduce Programming: MongoDB: Mongo DB - Terms used in RDBMS and Mongo DB - Data Types - MongoDB Query Language. MapReduce: Mapper - Reducer - Combiner -Partitioner- Searching - Sorting - Compression
4	Hive: Introduction - Architecture - Data Types - File Formats - Hive Query Language Statements - Partitions - Bucketing - Views - Sub- Query - Joins - Aggregations - Group by and Having - RCFile - Implementation - Hive User Defined Function - Serialization and Deserialization.
5	Pig: Introduction - Anatomy - Features - Philosophy - Use Case for Pig - Pig Latin Overview - Pig Primitive Data Types - Running Pig - Execution Modes of Pig - HDFS Commands - Relational Operators - Eval Function - Complex Data Types - Piggy Bank - User-Defined Functions - Parameter Substitution - Diagnostic Operator - Word Count Example using Pig - Pig at Yahoo! - Pig Versus Hive

Table of Contents

UNIT-I: BIG DATA AND ANALYTICS

UNIT I OBJECTIVES

Unit 1 provides an introduction to the concepts, technologies, and methods related to Big Data and Analytics. It focuses on understanding the characteristics of Big Data, the importance of analytics, and the various tools and techniques used in the field.

Unit Summary

1. **Introduction to Digital Data**
    - Classification of Digital Data: Structured, Semi-Structured, and Unstructured Data
    - Characteristics and Evolution of Big Data
2. **Introduction to Big Data**
    - Definition and Challenges of Big Data
    - Comparison between Traditional Business Intelligence and Big Data
    - Data Warehousing and Hadoop
3. **Big Data Analytics**
    - Classification of Analytics
    - Importance and Challenges of Big Data Analytics
    - The Role of Data Science and Data Scientists
    - Terminologies in Big Data Environments
    - Overview of Basically Available Soft State Eventual Consistency (BASE)
    - Top Analytics Tools

1.1 Types of Digital Data

What is Data ?

Data is present internal to the enterprise and also exists outside the four walls and firewalls of the enterprise. Data is present in homogeneous sources as well as in heterogeneous sources. The need of the hour is to understand, manage, process, and take the data for analysis to draw valuable insights.

Data → Information

Information → Insights

1.2 Classification of Digital Data

Digital data can be broadly classified into three types such as structured, semi-structured, and unstructured data. It is graphically shown in figure 1.1

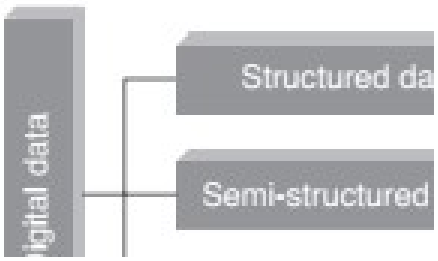


Figure 1.1 Classification of digital data

Unstructured Data: This type of data does not adhere to a predefined data model and is not easily processed by computer programs. It constitutes about 80–90% of an organization's data and includes formats like memos, chat logs, PowerPoint presentations, images, videos, letters, research papers, white papers, and email bodies.

Semi-Structured Data: This data does not fit neatly into a fixed data model but contains some organizational elements. Although it is not readily usable by computer programs in its raw form, it includes elements such as emails, XML files, and HTML markup. It often includes metadata, but this metadata may not be comprehensive enough for detailed processing.

Structured Data: This data is highly organized and formatted, typically into rows and columns, making it easily processable by computer programs. It involves clear relationships between data entities, such as classes and their associated objects.

Data stored in databases is an example of structured data. Ever since the 1980s most of the enterprise data has been stored in relational databases complete with rows/records/tuples, columns/attributes/fields, primary keys, foreign keys, etc. Over a period of time Relational Database Management System (RDBMS) matured and the RDBMS, as they are available today, have become more robust, cost-effective, and efficient. We have grown comfortable working with RDBMS – the storage, retrieval, and management of data has been immensely simplified. The data held in RDBMS is typically structured data. However, with the Internet connecting the world, data that

existed beyond one's enterprise started to become an integral part of daily transactions. This data grew by leaps and bounds so much so that it became difficult for the enterprises to ignore it. All of this data was not structured. A lot of it was unstructured. In fact, Gartner estimates that almost 80% of data generated in any enterprise today is unstructured data. Roughly around 10% of data is in the structured and semi-structured category.

1.2.1 Structured Data

Let us begin with a very basic question — When do we say that the data is structured? The simple answer is when data conforms to a pre-defined schema/structure we say it is structured data.

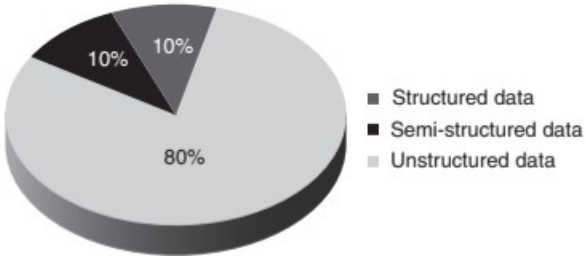


Figure 1.2 Approximate percentage distribution of digital data.

Think structured data, and think data model — a model of the types of business data that we intend to store, process, and access. Let us discuss this in the context of an RDBMS. Most of the structured data is held in RDBMS. An RDBMS conforms to the relational data model wherein the data is stored in rows/columns. Refer Table 1.1.

The number of rows/records/tuples in a relation is called the *cardinality of a relation* and the number of columns is referred to as the *degree of a relation.*

Table 1.2 is an example of a good structured table (complete with table name, meaningful column names with data types, data length, and the relevant constraints) with absolute adherence to relational data model.

Table 1.1 A relation/table with rows and columns

	Column 1	*Column 2*	*Column 3*	*Column 4*
Row 1				

Table 1.2 Schema of an "Employee" table in a RDBMS such as Oracle

Column Name	*Data Type*	*Constraints*
EmpNo	Varchar(10)	PRIMARY KEY
EmpName	Varchar(50)	
Designation	Varchar(25)	NOTNULL
DeptNo	Varchar(5)	
ContactNo	Varchar(10)	NOTNULL

1.2.1.1 Source of Structured Data

Structured data, characterized by its organized format and clear schema, is often generated by day-to-day business activities and is commonly associated with online transaction processing (OLTP) systems.

To effectively manage and store structured data, organizations can leverage various RDBMS solutions, including those offered by Oracle Corp., IBM (DB2), Microsoft (SQL Server), EMC (Greenplum), Teradata, MySQL (an open-source RDBMS), and PostgreSQL (an advanced open-source RDBMS).

These RDBMS solutions provide robust capabilities for storing, querying, and managing structured data efficiently. They are well-suited for handling transactional data, which is typically structured and generated in large volumes as a result of business operations. Figure 1.3 likely provides a visual representation or comparison of different RDBMS solutions, showcasing their features, strengths, and potential use cases.

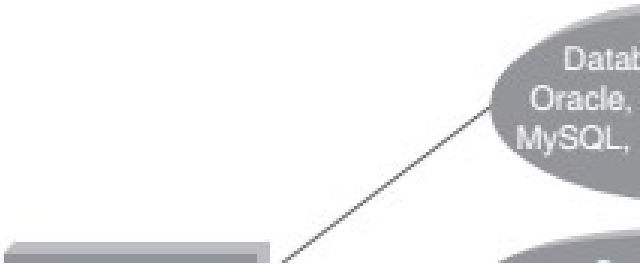


Figure 1.3 Sources of Structured data.

1.2.1.2. Ease of Working with Structured Data

Structured data provides the ease of working with it. Refer Figure 1.4. The ease is with respect to the following:

1. **Insert/update/delete:** The Data Manipulation Language (DML) operations provide the required ease with data input, storage, access, process, analysis, etc.
2. **Security:** How does one ensure the security of information? There are available staunch encryption and tokenization solutions to warrant the security of information throughout its lifecycle. Organizations are able to retain control and maintain compliance adherence by ensuring that only authorized individuals are able to decrypt and view sensitive information.
3. **Indexing:** An index is a data structure that speeds up the data retrieval operations (primarily the SELECT DML statement) at the cost of additional writes and storage space, but the benefits that ensue in search operation are worth the additional writes and storage space.
4. **Scalability:** The storage and processing capabilities of the traditional RDBMS can be easily scaled up by increasing the horsepower of the database server (increasing the primary and secondary or peripheral storage capacity, processing capacity of the processor, etc.).

Figure 1.4 Ease of Working with Sturctured Data

5. **Transaction processing:** RDBMS has support for Atomicity, Consistency, Isolation, and Durability (ACID) properties of transaction. Given next is a quick explanation of the ACID properties:

- ***Atomicity:*** A transaction is atomic, means that either it happens in its entirety or none of it at all.
- ***Consistency: The*** database moves from one consistent state to another consistent state. In other words, if the same piece of information is stored at two or more places, they are in complete agreement.
- ***Isolation:*** The resource allocation to the transaction happens such that the transaction gets the impression that it is the only transaction happening in isolation.
- ***Durability:*** All changes made to the database during a transaction are permanent and that accounts for the durability of the transaction.

1.2.2 Semi-Structured Data

Semi-structured data is also referred to as self-describing structure Refer Fig. 1.5.

1. It does not conform to the data models that one typically associates with relational databases or any other form of data tables.

2. It uses tags to segregate semantic elements.

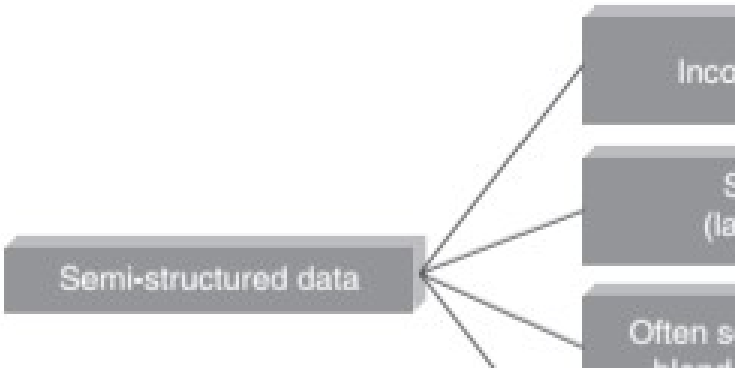


Figure 1.5 Characterstics of Semi-Structured Data

3. Tags are also used to enforce hierarchies of records and fields within data.
4. There is no separation between the data and the schema. The amount of structure used is dictated by the purpose at hand.
5. In semi-structured data, entities belonging to the same class and also grouped together need not necessarily have the same set of attributes. And if at all, they have the same set of attributes, the order of attributes may not be similar and for all practical purposes it is not important as well.

1.2.2.1 Sources of Semi-Structured Data

The popular sources for semi-structured data are “XML” and “JSON” as depicted in Figure 1.6.

1. **XML:** eXtensible Markup Language (XML) is hugely popularized by web services developed utilizing the Simple Object Access Protocol (SOAP) principles.
2. **JSON:** Java Script Object Notation (JSON) is used to transmit data between a server and a web application. JSON is popularized by web services developed utilizing the Representational State Transfer (REST) – an architecture style for creating scalable web services. MongoDB (open-source, distributed, NoSQL, documented-oriented database) and Couchbase (originally known as Membase)

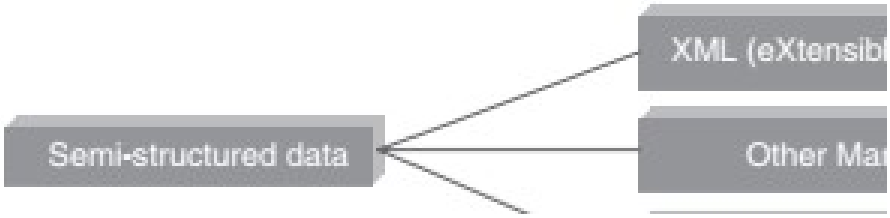


Figure 1.6 Sources of Semi-Structured Data

Sample JSON document:

```
 {
 _id:9,
 Book_Title: “Fundamentals of  Analytics”,
Author_Name: “Seema”,
 Publishers: “Wiley India”,
 Year :“2021”
}
```

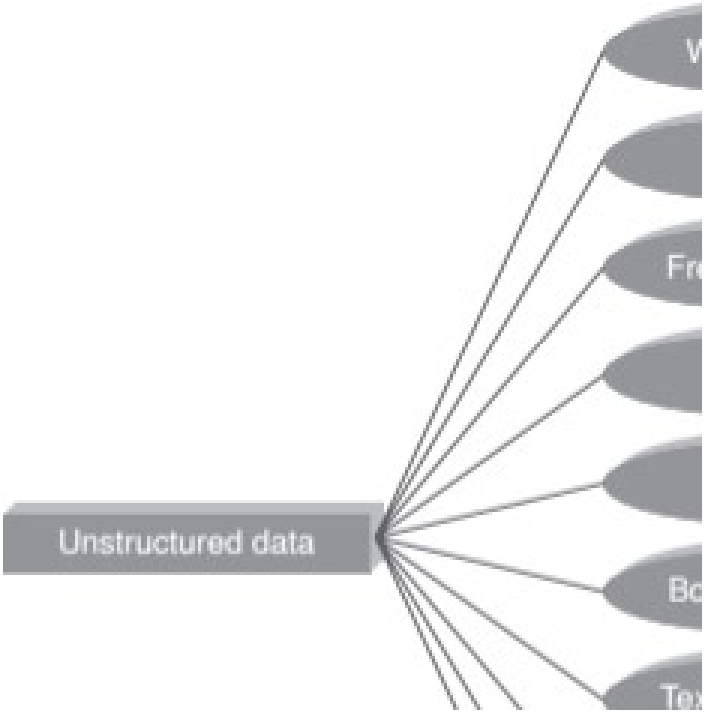


Figure 1.7 Sources of unstructured data

1.2.3 Unstructured Data

Unstructured data does not conform to any pre-defined data model. Figure 1.7 depicts the sources of unstructured data.

1.2.3.1 Issues with "Unstructured" Data

Although unstructured data is known NOT to conform to a pre-defined data model or be organized in a pre-defined manner, there are incidents wherein the structure of the data (placed in the unstructured category) can still be implied. Figure 1.8 illustrate the issues in the it.

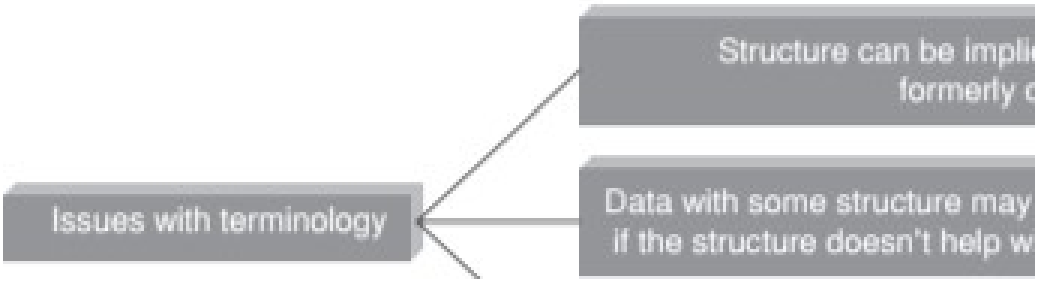


Figure 1.8 Issues with terminology of unstructured data

The following techniques are used to find patterns in or interpret unstructured data:

1. **Data mining:** First, we deal with large data sets. Second, we use methods at the intersection of artificial intelligence, machine learning, statistics, and database systems to unearth consistent patterns in large data sets and/or systematic relationships between variables. It is the analysis step of the "knowledge discovery in databases" process. Few popular data mining algorithms are as follows:

    a. Association rule mining: It is also called "market basket analysis" or "affinity analysis". It is used to determine "What goes with what?" It is about when you buy a product, what is the other product that you are

likely to purchase with it. For example, if you pick up bread from the grocery, are you likely to pick eggs or cheese to go with it. •

b. Regression analysis: It helps to predict the relationship between two variables. The variable whose value needs to be predicted is called the dependent variable and the variables which are used to predict the value are referred to as the independent variables.

Lets sum up

Big Data and Analytics introduce fundamental concepts essential for understanding the landscape of digital data. It covers the classification of digital data into structured, semi-structured, and unstructured forms, highlighting their respective characteristics and usage scenarios. Structured data, organized into predefined schemas, exemplifies data found in traditional relational databases, crucial for transactional processes due to its consistency and ease of management.

Semi-structured data, lacking a rigid schema but employing tags for hierarchy, is prevalent in formats like XML and JSON, commonly used in web services. Meanwhile, unstructured data, which lacks any predefined format or model, constitutes a significant portion of enterprise data, including documents, emails, and multimedia content. The unit also discusses the evolution of Big Data, emphasizing its challenges and distinctiveness from traditional Business Intelligence, along with an overview of technologies like Hadoop and data warehousing. Additionally, it explores the importance of analytics in harnessing valuable insights from Big Data, the role of data science and scientists, and introduces key terminologies such as BASE (Basically Available Soft State Eventual Consistency). This foundational understanding sets the stage for deeper exploration into tools and techniques employed in Big Data analytics.

1.3 Introduction to Big Data

The rapid growth of the "Internet of Things" has led to an overwhelming expansion of big data. Businesses today are inundated with vast amounts of data, creating both challenges and opportunities. This raises several key questions:

Why is big data now indispensable?

How has it become so crucial for modern business operations?

How does it compare with traditional Business Intelligence (BI) systems?

Will big data replace traditional relational databases and data warehouses, or will it enhance them?

Examples of Big Data:

Big data analytics plays a vital role across various sectors, including retail, IT infrastructure, and social media.

Retail: Big data offers significant advantages for improving sales and marketing strategies. For instance, the U.S. retailer Target analyzed customer purchasing patterns and discovered that major life events like marriage, divorce, and pregnancy drive substantial shifts in consumer behavior. Target used these insights to manage its inventory, anticipating demand for specific products based on life-event trends.

IT Infrastructure: The MapReduce model is often used in big data projects due to its ability to handle large, unconventional datasets. Hadoop, which is built on a distributed file system, allows organizations to use clusters of servers to process massive data volumes cost-effectively.

Social Media: Platforms like Twitter and Facebook generate enormous quantities of unstructured data. Hadoop and its ecosystem of tools help manage and analyze this data, enabling companies to create social graphs and manage transactions between millions of users.

Professional Networks: LinkedIn is an example of a company where data is central to its product. The platform, which caters to professionals, has grown to over 250 million users as of 2014, offering various data-driven services such as job recruitment, advertising, and social graphs of users' professional networks.

Healthcare:

Big data is revolutionizing the healthcare industry by enabling better patient outcomes, predictive analytics, and personalized medicine. Hospitals and healthcare providers use

vast amounts of patient data to identify trends, predict outbreaks, and offer more effective treatments.

Example: IBM's Watson Health uses big data to analyze medical records, clinical trials, and research papers to help doctors make more informed decisions. Predictive analytics are used to foresee potential health risks based on a patient's medical history and lifestyle.

Finance: The finance industry uses big data to detect fraud, assess credit risks, and provide personalized financial services.

Example: Credit card companies use big data analytics to monitor transactions in real-time, spotting irregularities that might indicate fraudulent activity. Machine learning algorithms help in predicting customer defaults or offering tailored loan products based on spending habits and income data.

Manufacturing: Big data is used in manufacturing for optimizing production processes, predictive maintenance, and improving product design by analyzing customer feedback.

Example: General Electric (GE) uses big data to monitor machinery in real time through sensors embedded in industrial equipment. This allows GE to predict when a machine part will fail and perform maintenance before a breakdown occurs, reducing downtime.

1.4 Characteristics of Data:

As in Figure 1.9, data can be characterized by three main attributes:

1. Composition: This refers to the structure and sources of the data, including its granularity, type, and whether it is static or real-time.
2. Condition: This aspect relates to the usability of data. It asks whether the data is ready for analysis or if it requires cleansing or enhancement.
3. Context: The context addresses the origin, purpose, and sensitivity of the data, as well as any related events.

Before the big data revolution, "small data" was about certainty, with clear, defined sources and relatively stable structures. In contrast, big data introduces new complexities in the composition, condition, and context of information.

Figure 1.9 Charasteristics of data

1.5 Evolution of Big Data:

1970s and before was the era of mainframes. The data was essentially primitive and structured. Relational databases evolved in 1980s and 1990s. The era was of data intensive applications. The World Wide Web (WWW) and the Internet of Things (IOT) have led to an onslaught of structured, unstructured, and multimedia data. Table 1.3 clearly showed the evolution of big data.

Table 1.3 The evolution of Big Data

	Data Generation and Storage	Data Utilization
Complex and Unstructured		
Complex and Relational		Relational database Data-intensive applications

1.6 Definition of Big Data:

Big data refers to vast, fast, and diverse datasets that require advanced and cost-effective methods for processing to gain better insights and improve decision-making. These datasets often exceed the capacity of traditional database tools and require innovative technologies to handle their size and complexity.

Big data is characterized by three main dimensions:

Volume: The enormous scale of data, often measured in terabytes, petabytes, or even zettabytes.

Velocity: The speed at which data is generated and processed, moving from traditional batch processing to real-time data streams.

Variety: The different types of data, ranging from structured (like databases) to semi-structured (e.g., XML files, emails), and unstructured data (such as multimedia, documents, and emails).

Key Characteristics:

Variety: Data comes in multiple formats. Structured data fits neatly into relational databases, while semi-structured data includes formats like HTML, XML, and CSV files. Unstructured data comprises diverse content like images, videos, research papers, and emails, all of which lack a predefined structure.

Velocity: This refers to the speed at which data is generated and must be processed. As systems move from basic desktop applications to real-time data streams, the need for immediate data processing has grown significantly.

Volume: The sheer amount of data being produced has grown exponentially, reaching terabytes, petabytes, and even zettabytes.

The Gartner Definition of Big Data:

According to Gartner, big data involves "high-volume, high-velocity, and/or high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making."

This definition can be broken into three main parts:

High-Volume, High-Velocity, and High-Variety Information Assets: Big data typically involves vast quantities of data, coming in various formats and requiring quick processing for storage, analysis, and interpretation.

Cost-Effective and Innovative Processing: Handling big data requires embracing new technologies and methodologies to ingest, store, and analyze the data efficiently. This could include distributed storage systems, cloud computing, or advanced analytics platforms that are capable of managing large datasets at a lower cost.

Enhanced Insight and Decision-Making: The ultimate goal of big data is to derive meaningful insights that drive smarter, faster, and more informed decision-making. Organizations that can effectively analyze big data gain a competitive edge by using these insights to create business value.

Big data has transformed industries by enabling deeper analysis of vast datasets, leading to more informed strategies and faster responses to market changes.

Data —> Information —> Actionable intelligence —> Better decisions — >Enhanced business value

1.6.1 Challenges with big data

Following are a few challenges with big data are depicted in figure 1. 10.

Figure 1.10 Challenges with bigdata

- **Data Volume:** The amount of data being generated is increasing exponentially, and this trend is expected to continue. Critical questions arise, such as:

“Is all this data valuable for analysis?”

“Should we work with the entire dataset or just a portion?”

“How can we filter useful information from irrelevant data?”

- **Storage:** Cloud computing offers a cost-efficient, scalable, and flexible solution for managing big data infrastructure. However, using cloud services for hosting big data solutions adds complexity, particularly when deciding whether to keep data systems within the organization or outsource them.

- **Data Retention:** Determining how long data should be stored is another challenge. While some data might be valuable for long-term decision-making, other data may quickly become outdated and lose its relevance.

- **Skilled Professionals:** Organizations need highly skilled data scientists and professionals to develop, manage, and run big data applications that generate meaningful insights. The demand for expertise in this field is growing rapidly.

- **Additional Challenges:** Other significant issues include the capture, storage, search, analysis, transfer, and security of big data. Ensuring that these processes are efficient and secure is crucial for handling massive datasets.

- **Visualization:** Big data refers to datasets that exceed the storage and processing capabilities of traditional database tools. Data visualization, or the representation of data using computer graphics, is emerging as a specialized field, yet there are still very few experts in this area.

What is Big Data?

Big data refers to vast and intricate datasets that are too large and complex to be efficiently processed using traditional data management tools. These datasets are often described by four key characteristics, known as the "4 Vs" of big data:

Volume: Big data encompasses huge quantities of information, often measured in terabytes, petabytes, or even exabytes. The sheer size of these datasets surpasses the capabilities of traditional database systems, requiring specialized techniques and tools for storage and processing.

Velocity: The speed at which big data is generated is extremely high, with data flowing continuously from sources like sensors, social media, mobile devices, and the Internet of Things (IoT). This rapid rate of data creation presents challenges for real-time data processing and analysis.

Variety: Big data comes in many different forms, including structured, semi-structured, and unstructured formats. Structured data is organized with a predefined schema, often stored in databases or tables, while unstructured data lacks such structure and includes elements like text, images, and videos. Semi-structured data falls in between, with some level of organization but less rigid than fully structured data.Other characteristics of data that are not necessarily specific to big data but are still crucial considerations in data analysis include the following: Here's a summary:

1. **Veracity and Validity**:
    - **Veracity**: Denotes the trustworthiness and reliability of data, taking into account biases, noise, and abnormality. It questions whether all the data being stored, mined, and analyzed is meaningful and relevant to the problem at hand.
    - **Validity**: Denotes the accuracy and correctness of the data. It emphasizes the importance of ensuring that the data selected for analysis is accurate, regardless of whether it falls under the realm of big data.
2. **Volatility**:
    - **Volatility**: Addresses the duration for which data remains valid and relevant. It raises questions about how long data should be stored and how long it retains its usefulness.
    - Some data may be required for long-term decision-making and remain valid for extended periods, while other data quickly becomes obsolete, sometimes within minutes or hours.

1.7 Why Big data?

The larger the amount of data available for analysis, the higher the accuracy of the insights derived, leading to greater confidence in the decisions based on those findings.

Improved accuracy in analysis can result in significant benefits, such as increased operational efficiency, reduced costs and time, the development of new products and services, and the optimization of current offerings. (See Figure 1.11.)

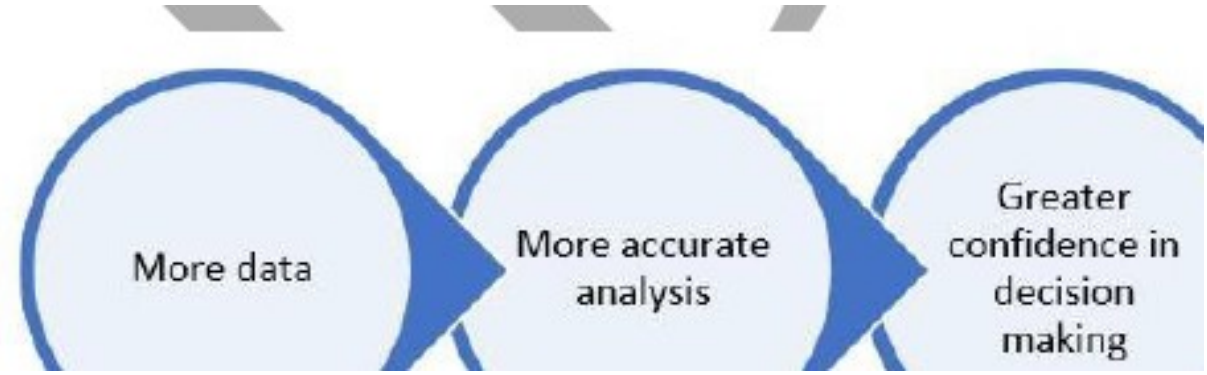


Figure 1.11 Why is Big Data?

1.7.1 Business Intelligence vs Big Data

While both Big Data and Business Intelligence (BI) are used to analyze data and support companies in decision-making, they differ in several key aspects, particularly in how they function and the types of data they process.

Traditional BI operates by gathering all business-related data into a centralized server. The data is typically analyzed in an offline mode after being stored in a Data Warehouse. This data is structured and organized using relational databases, along with additional indexes and access methods such as multidimensional cubes for efficient querying.These are the main differences between

1.7.1.1 Big Data and Business Intelligence:

In a Big Data environment, data is stored on a distributed file system rather than a centralized server, offering greater security and flexibility.

- Big Data solutions bring the processing tasks to the data itself, rather than transferring the data to the processing functions. This method enables easier handling of large data volumes in a more efficient and agile manner.

- Big Data can analyze both structured and unstructured data, with the latter growing at a faster rate than traditional structured data. However, unstructured data analysis presents unique challenges.
- Big Data addresses these challenges by enabling a comprehensive analysis of data from multiple sources, whether it is historical or real-time. This allows companies to make swift, informed decisions that impact their business operations.
- Big Data technologies utilize mass parallel processing (MPP), which enhances the speed of data analysis. By executing multiple tasks simultaneously and breaking them into smaller, parallel parts, the results are merged at the end, enabling quick analysis of large datasets.

Lets sum up

In this section, the focus is on understanding Big Data and its significance in modern business operations. Big Data refers to large and complex datasets characterized by high volume, velocity, and variety, which traditional data processing methods struggle to handle effectively. The advent of the Internet of Things (IoT) has contributed significantly to the proliferation of Big Data, generating data at unprecedented speeds and in diverse formats such as structured, semi-structured, and unstructured.

In summary, Big Data represents a paradigm shift in data analytics, offering organizations the ability to leverage diverse data sources for informed decision-making and competitive advantage. It complements traditional BI approaches by addressing the challenges posed by the volume, velocity, and variety of modern data, thereby transforming how businesses extract value from their data assets.

1.8 Hadoop Environment Big Data Analytics

Hadoop is revolutionizing the way Big Data, particularly unstructured data, is managed. The Apache Hadoop software library, which functions as a framework, plays a critical role in processing Big Data. It allows vast amounts of data to be efficiently handled in distributed processing systems across clusters of computers, utilizing simple

programming models. Hadoop is designed to scale from a single server to thousands of machines, each providing local computation and storage. Rather than relying on hardware for high availability, the framework is engineered to detect and manage failures at the application level, ensuring a highly available service even when individual machines are prone to breakdowns.

Components of the Hadoop Community Package:

File system and OS-level abstractions

A MapReduce engine (either MapReduce or YARN)

The Hadoop Distributed File System (HDFS)

Java Archive (JAR) files

Scripts to initiate Hadoop

Source code, documentation, and a contribution section

Key Activities in Big Data Management:

Storage: Big Data must be gathered into a unified repository, though it doesn't need to reside in a single physical database.

Processing: The process involves more complexity than traditional methods, including data cleansing, enrichment, calculation, transformation, and algorithm execution.

Access: For Big Data to be useful, it must be easily searchable, retrievable, and presented in a way that aligns with business objectives. Without this, the data holds no real value for business purposes.

1.8.1 What is Big Data Analytics?

"Big Data Analytics" entails the following:

1. **Technology-Enabled Analytics**:
    - Uses data analytics and visualization tools from leading vendors like IBM, Tableau, SAS, R Analytics, Statistica, and World Programming Systems (WPS) to process and analyze big data effectively.
2. **Insightful Business Steering**:

- Aims to gain meaningful insights into business operations, customer demographics, vendor relationships, etc., to make informed decisions and enhance strategies.
- Example: Personalized recommendations from online retailers demonstrate the use of data analytics to understand customer preferences and offer relevant products or services.

3. **Competitive Edge**:
   - Provides findings enabling quicker and better decision-making, granting a competitive advantage in the market.
4. **Collaboration Among IT, Business Users, and Data Scientists**:
   - Involves close collaboration among IT, business users, and data scientists to leverage big data analytics effectively.
   - Refer to Figure 3.3 for likely illustrating the interplay and collaboration among these communities.
5. **Handling Large and Varied Datasets**:
   - Deals with datasets surpassing current storage and processing capabilities, requiring advanced technologies and techniques for analysis.
6. **Efficient Code-to-Data Processing**:
   - Involves moving processing code to data rather than moving large data volumes, enhancing efficiency, especially for distributed processing.

1.9 What Big Data isn’t?

Big data isn't just about volume; it encompasses variety and velocity. It's not solely reliant on technology but also on strategic analysis. Big data isn't exclusive to large organizations; small and medium-sized businesses can benefit too. Figure 1.12 clearly depicts what is big data and figure 1.13 describes what big data isn’t.

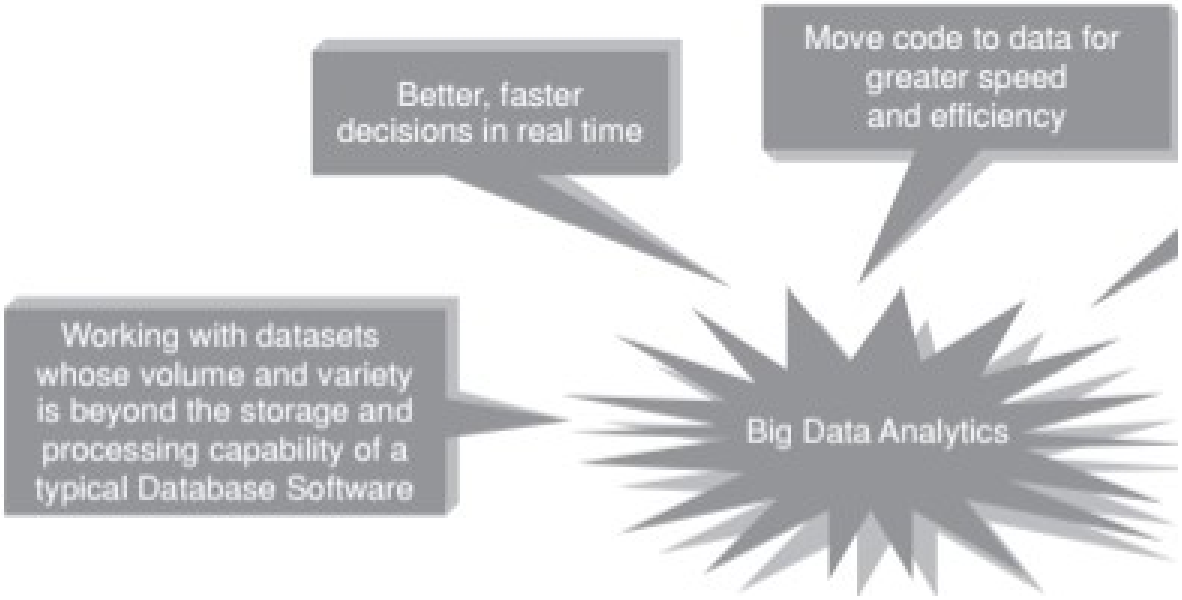


Figure 1.12 What is big Data Analytics?

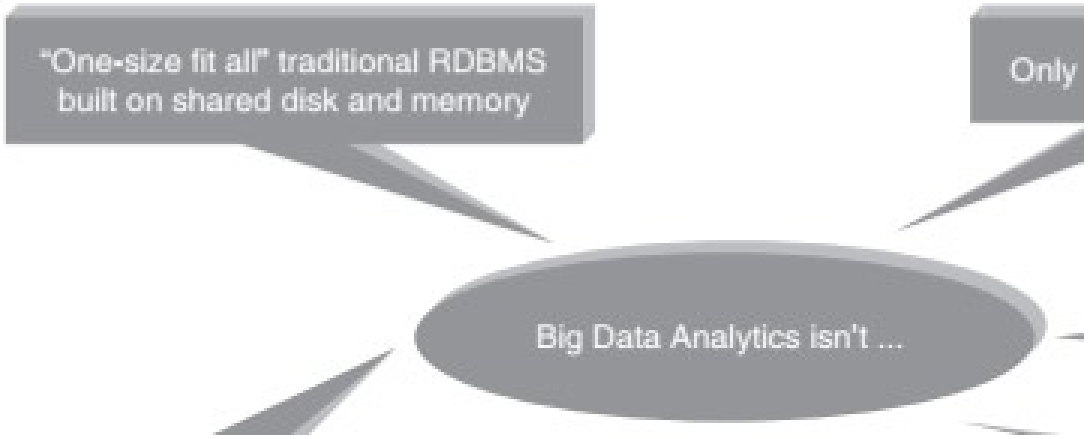


Figure 1.13 What is Big Data Analytics isn't?

1.9.1 Classification of analytics involves categorizing different types of data analysis methods based on their objectives and techniques.

1. Descriptive Analytics:

- **Definition**: Descriptive analytics involves examining historical data to uncover patterns, trends, or insights.

- **Techniques**: It primarily uses data aggregation and data mining techniques to summarize and analyze historical data.
- **Advantages**:
    - Provides quick reports on Return on Investment (ROI) by showing how performance achieved business or target goals.
    - Helps identify gaps and performance issues early, before they become significant problems.
    - Enables the identification of specific learners who may require additional support or resources.
    - Helps analyze the value and impact of course design and learning resources.

2. Predictive Analytics:

- **Definition**: Predictive analytics utilizes algorithms and machine learning to identify trends in data and forecast future behaviors or outcomes.
- **Techniques**: It involves statistical modeling and machine learning algorithms to predict future trends or events based on historical data.
- **Advantages**:
    - Personalizes training needs by identifying individual gaps, strengths, and weaknesses, offering tailored learning resources.
    - Helps retain talent by tracking employee career progression and forecasting required skills for career advancement.
    - Supports employees falling behind by offering intervention support before their performance declines significantly.
    - Simplifies reporting and visuals for better decision-making.

3. Prescriptive Analytics:

- **Definition**: Prescriptive analytics generates recommendations and decisions based on computational findings from algorithmic models.

- **Techniques**: It involves developing specific algorithmic models to generate automated recommendations or decisions based on identified problems or desired outcomes.
- **Example**: If predictive analytics reveals that learners lacking a specific skill may not complete a course, prescriptive analytics can recommend additional training resources to acquire the missing skill.
- **Considerations**: Accuracy of recommendations depends on the quality of data and algorithmic models, which may need customization for different situations.

Table 1.4 Comparison of Analytics 1.0 vs 2.0 vs 3.0

Analytics 1.0	Analytics 2.0	An
Era: mid 1950s to 2009	**2005 to 2012**	**20**
Descriptive statistics (report on events, occurrences, etc. of the past)	Descriptive statistics + predictive statistics (use data from the past to make predictions for the future)	De pre (us pro sa to ad
Key questions asked: What happened? Why did it happen?	Key questions asked: What will happen? Why will it happen?	Ke Wh Wh Wh Wh tak
Data from legacy systems, ERP, CRM, and 3rd party applications.	Big data	A leg 3rd
Small and structured data sources. Data	Big data is being taken up seriously. Data is mainly unstructured, arriving at a much	A an

1.9.2 Comparison:

Table 1.4 shows the comparative study of various analytics in detail. In a nutshell,

- **Descriptive Analytics**: Focuses on historical data.
- **Predictive Analytics**: Uses historical data to forecast future possibilities.
- **Prescriptive Analytics**: Takes forecasted outcomes from predictive analytics and predicts consequences for these outcomes.

These analytics types progressively build upon each other, starting from historical data analysis to forecasting future possibilities and recommending actions based on those forecasts.

1.9.3 Greatest Challenges that Prevent Businesses from Capitalizing on Big Data

The greatest challenges that prevent businesses from capitalizing on big data, aligned and clearly presented:

1. **Executive Sponsorship**: Securing executive sponsorship for investments in big data initiatives and related activities, such as training.
2. **Information Sharing**: Encouraging business units to share information across organizational silos.
3. **Skills Gap**: Finding skilled business analysts and data scientists capable of managing and deriving insights from large, varied datasets.
4. **Scalability**: Developing an approach to scale storage and processing capabilities rapidly and elastically to handle large volumes, velocities, and varieties of big data.
5. **Data Utilization**: Deciding whether to use structured or unstructured, internal or external data for business decision-making.
6. **Reporting and Visualization**: Choosing the best methods for presenting findings and analyses of big data in a clear and understandable visual format.
7. **Actionable Insights**: Determining how to effectively act on the insights generated from big data analysis.

1.9.4 Top Challenges Facing Big Data

1. **Scale**:
    - **Storage Solutions**: Need to choose between RDBMS and NoSQL to handle large volumes, velocity, and variety of data.
    - **Scaling Methods**: Deciding whether to scale vertically (adding more power to existing machines) or horizontally (adding more machines).
2. **Security**:

- **Weak Security in NoSQL**: Many NoSQL platforms lack robust authentication and authorization mechanisms, risking sensitive data like credit card and personal information.

3. **Schema**:
   - **Need for Dynamic Schemas**: Traditional rigid schemas are outdated; dynamic schemas are required to accommodate the flexible nature of big data.
4. **Continuous Availability**:
   - **24/7 Support**: Ensuring continuous availability despite the built-in downtime of most RDBMS and NoSQL platforms.
5. **Consistency**:
   - **Consistency Models**: Decision on whether to opt for strong consistency or eventual consistency in data processing and storage.
6. **Partition Tolerance**:
   - **Building Tolerant Systems**: Creating systems that can handle both hardware and software failures effectively.
7. **Data Quality**:
   - **Maintaining Quality**: Ensuring data accuracy, completeness, timeliness, and having appropriate metadata.

1.9.5 Why is Big Data Analytics Important?

1. **Reactive – Business Intelligence**:
   - **Historical Data Analysis**: Helps businesses make faster, better decisions by analyzing past data and providing relevant information through dashboards, alerts, and notifications.
   - **Reporting**: Supports pre-specified reports and ad hoc querying.
2. **Reactive – Big Data Analytics**:
   - **Static Data Analysis**: Analysis is performed on large datasets but remains reactive as it is based on static data.
3. **Proactive – Analytics**:

- **Futuristic Decision Making**: Utilizes data mining, predictive modeling, text mining, and statistical analysis.
- **Limitations**: Traditional database management practices impose storage and processing limitations.

4. **Proactive – Big Data Analytics**:
   - **High-Performance Analytics**: Analyzes terabytes, petabytes, exabytes of data to extract relevant information and solve complex problems quickly.
   - **Advanced Techniques**: Involves high-performance analytics for rapid insights.

1.9.6 Technologies to Meet Big Data Challenges

1. **Cheap and Abundant Storage**:
   - Affordable storage solutions are essential to handle large volumes of data.
2. **Faster Processors**:
   - Necessary to speed up the processing of big data.
3. **Open-Source, Distributed Platforms**:
   - Platforms like Hadoop provide cost-effective solutions for big data processing.
4. **Parallel Processing and Clustering**:
   - Distributing processing tasks across multiple machines using clustering, virtualization, and large grid environments for high connectivity and throughput.
5. **Cloud Computing**

1.10 Data Science Overview

- **Definition:** Data science is the science of extracting knowledge from data by uncovering hidden patterns using statistical and mathematical techniques.
- **Interdisciplinary Nature:** It integrates fields like mathematics, statistics, information technology, machine learning, data engineering, probability models, statistical learning, and pattern recognition.

- **Applications:** Data science is applied in numerous areas including weather predictions, oil drilling, seismic activity monitoring, financial fraud detection, terrorist network analysis, global economic impact studies, sensor logs, social media analytics, customer churn, market basket analysis, collaborative filtering, and regression analysis.

1.10.1 Skills Required for Data Scientists:

1. Business Acumen Skills:

- **Understanding of Domain:** Knowledge of the business field.
- **Business Strategy:** Ability to align data science work with business strategies.
- **Problem Solving:** Skills to tackle business problems using data insights.
- **Communication:** Effective communication skills to convey findings.
- **Presentation:** Proficiency in presenting data insights clearly.
- **Inquisitiveness:** Curiosity and a drive to explore data for insights.

2. Technology Expertise:

- **Technical Skills:** Mastery of relevant technical skills is essential for data scientists. Specific skills were not listed in the provided text, but generally include programming, data manipulation, and knowledge of big data tools and frameworks.

The responsibilities of the Data Scientists is given in the Figure 1.14

Figure 1.14 Data Scientist : your new best friend!!!

1.10.2 Terminologies Used in Big Data Environments

1. In-Memory Analytics:

- **Definition:** Processing data directly in memory (RAM) for faster analysis.
- **Advantage:** Significantly speeds up data analysis by reducing the time needed to read and write to disk.

2. In-Database Processing:

- **Definition:** Performing data processing within the database itself.
- **Advantage:** Enhances performance by minimizing data movement and leveraging database capabilities for analytics.

3. Symmetric Multiprocessor System (SMP):

- **Definition:** A system where multiple processors share a single, coherent memory space and operate under a single operating system instance.
- **Advantage:** Facilitates easier programming and management.

4. Massively Parallel Processing (MPP):

- **Definition:** A system that uses many independent processors to perform tasks simultaneously.
- **Advantage:** Allows for handling large datasets and complex computations efficiently.

5. Difference between Parallel and Distributed Systems:

- **Parallel Systems:** Multiple processors working on different parts of a single task within a shared memory environment.
- **Distributed Systems:** Multiple independent systems working on separate tasks, often communicating over a network.

6. Shared Nothing Architecture:

- **Definition:** Each node in the system operates independently and does not share memory or storage.
- **Advantage:** Enhances scalability and fault tolerance, as failure in one node does not affect others.

1.11. CAP Theorem (Brewer's Theorem): The CAP theorem posits that in a distributed computing environment (a collection of interconnected nodes that share data), it is impossible to achieve all three of the following guarantees simultaneously. At best, only two of the three can be achieved:

1. **Consistency:**
    - **Definition:** Every read fetches the latest write.
    - **Implication:** All nodes return the same, most recent data.
2. **Availability:**
    - **Definition:** Every request (read or write) receives a response, regardless of the individual state of any node.
    - **Implication:** The system is operational and responsive at all times.
3. **Partition Tolerance:**

- **Definition:** The system continues to operate despite network partitions (communication breakdowns between nodes).
- **Implication:** The system remains functional even if parts of it cannot communicate with each other.

1.11.1 Real-Life Analogy: Consider a training institute, "XYZ," with 50 instructors and a training coordinator. Each morning, instructors check their schedules with Amey, the office administrator. When the institute experiences a surge in training requests and expands its instructor pool, the system managing the schedules must adapt:

- **Consistency:** If any schedule is updated, all instructors should see the updated schedule immediately.
- **Availability:** Instructors should always be able to access their schedules, even if Amey is busy or the system is under load.
- **Partition Tolerance:** The scheduling system should continue to function even if some parts of the network are down or overloaded.

1.12 Basic Availability, Soft State and Eventual Consistency

Basic availability ensures that a system remains operational and accessible even in the event of network failures, and it can tolerate temporary inconsistencies.

Soft state denotes that the state of the system can change without new input, which is essential for achieving eventual consistency.

Eventual consistency means that if no additional updates are made to a specific data item for a sufficiently long period, all users will eventually observe the same value for that item.

1.13 Top Analytics Tools

- **R**

R is a language designed for statistical computing and graphics, and it's also utilized for big data analysis. It offers a wide range of statistical tests and features:

Efficient Data Handling: Provides robust facilities for data storage and management.

Array and Matrix Operations: Offers a set of operators for performing calculations on arrays, especially matrices.

Integrated Tools: Features a comprehensive suite of big data tools for data analysis.

Graphical Capabilities: Includes tools for creating visual data representations, both on-screen and for print.

- **Apache Spark**

Apache Spark is a powerful open-source tool for big data analytics. It provides over 80 high-level operators that simplify the development of parallel applications and is used by many organizations to handle large datasets. Key features include:

Enhanced Performance: Runs applications in a Hadoop cluster up to 100 times faster in-memory and ten times faster on disk.

Rapid Processing: Known for its lightning-fast processing capabilities.

Advanced Analytics: Supports complex analytical tasks.

Hadoop Integration: Seamlessly integrates with Hadoop and utilizes existing Hadoop data.

- **Plotly**

Plotly is an analytics tool that enables users to create and share interactive charts and dashboards online. Notable features include:

Data Visualization: Transforms data into visually appealing and informative graphics.

Data Provenance: Offers detailed information on data origins for audited industries.

Public Hosting: Provides unlimited file hosting through its free community plan.

- **Lumify**

Lumify is a platform for big data fusion, analysis, and visualization, designed to help users uncover connections and explore relationships in their data. Its features include:

Graph Visualization: Supports both 2D and 3D graph visualizations with various automatic layouts.

Link Analysis: Offers tools for analyzing relationships between entities in the graph.

Content Ingestion: Includes specific elements for processing and interfacing with textual content, images, and videos.

Workspace Organization: Allows organization of work into projects or workspaces.

Scalable Technology: Built on established, scalable big data technologies.

- **IBM SPSS Modeler**

IBM SPSS Modeler is a predictive analytics platform for big data. It provides predictive modeling and delivers insights to various stakeholders. Features include:

Insight Discovery: Quickly analyze both structured and unstructured data to find insights and solve problems.

User-Friendly Interface: Designed with an intuitive interface suitable for all users.

Deployment Options: Offers flexible deployment choices, including on-premises, cloud, and hybrid.

Algorithm Selection: Facilitates quick selection of the best-performing algorithms based on model performance.

MongoDB

MongoDB is a NoSQL, document-oriented database developed in C and C++. It is an open-source tool available for free and supports various operating systems, including Windows Vista and later, OS X (10.7 and up), Linux, Solaris, and FreeBSD.

Main Features:

Aggregation: Advanced data processing and aggregation capabilities.

Ad Hoc Queries: Supports flexible and dynamic querying.

BSON Format: Uses BSON (Binary JSON) for data storage.

Sharding: Distributes data across multiple servers to manage large datasets.

Indexing: Provides various indexing options for faster data retrieval.

Replication: Ensures data redundancy and high availability.

Server-Side JavaScript: Executes JavaScript code on the server.

Schemaless: Allows for flexible data modeling without a fixed schema.

Capped Collections: Supports collections with a fixed size, where old data is automatically overwritten.

MongoDB Management Service (MMS): Offers tools for monitoring and backup.

Load Balancing: Distributes workloads evenly across servers.

File Storage: Manages and stores large files.

Lets sum up

The content discussed in this section revolves around Apache Hadoop and its role in handling Big Data through distributed processing across clusters. It emphasizes Hadoop's scalability, fault tolerance, and its components like HDFS and MapReduce/YARN. Big Data analytics are highlighted as crucial for gaining insights, enhancing decision-making, and maintaining competitive advantages through technologies like predictive and prescriptive analytics. Challenges in capitalizing on Big Data include executive sponsorship, skills gaps, scalability, and data utilization issues. The importance of analytics tools like R, Apache Spark, Plotly, Lumify, IBM SPSS Modeler, and MongoDB in managing and analyzing large datasets is also covered, along with their respective features and benefits.

UNIT 1 SUMMARY

- **Classification of Digital Data**
  - Structured Data: Organized and easily searchable data, typically stored in databases (e.g., SQL databases).
  - SemiStructured Data: Contains tags or markers to separate data elements, but not strictly organized in relational databases (e.g., JSON, XML).
  - Unstructured Data: Lacks a predefined format or organization, making it difficult to collect, process, and analyze (e.g., text documents, images).

- **Introduction to Big Data**
  - Characteristics: Volume, Velocity, Variety, Veracity, and Value.
  - Evolution: Growth from traditional databases and data processing techniques to handling vast amounts of diverse data types.
  - Definition: Large and complex data sets that traditional data processing software cannot adequately handle.
  - Challenges with Big Data: Data storage, processing speed, data integration, data quality, and security.
- **Other Characteristics of Data**
  - Variability: Data flows can be highly inconsistent with periodic peaks.
  - Complexity: Data comes from multiple sources, making it difficult to link, match, cleanse, and transform data across systems.
- **Big Data versus Traditional Business Intelligence (BI)**
  - Traditional BI: Focuses on descriptive and diagnostic analytics with structured data.
  - Big Data: Encompasses a broader range of data types and advanced analytics, including predictive and prescriptive analytics.
- **Data Warehouse and Hadoop**
  - Data Warehouse: Central repository for structured data from various sources, optimized for query and analysis.
  - Hadoop: Opensource framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.

- **Environment for Big Data Analytics**
  - Classification of Analytics: Descriptive, Diagnostic, Predictive, and Prescriptive analytics.
  - Challenges: Data privacy, data security, data governance, integration of diverse data sources, and scalability of analytics solutions.
- **Importance of Big Data Analytics**

- Enables businesses to gain insights from data, optimize operations, enhance customer experiences, and drive innovation.

- **Data Science and Data Scientist**
  - Data Science: Interdisciplinary field focused on extracting knowledge and insights from data.
  - Data Scientist: Professional skilled in statistics, programming, and domain expertise to analyze and interpret complex data.
- **Terminologies in Big Data Environments**
  - Hadoop Ecosystem: Includes tools like HDFS (Hadoop Distributed File System), MapReduce, YARN, and related projects like Spark, Hive, and Pig.
  - NoSQL: Databases designed for handling unstructured data (e.g., MongoDB, Cassandra).
- **Top Analytics Tools**
  - Hadoop: Framework for distributed storage and processing.
  - Spark: Fast, in-memory data processing engine.
  - Tableau: Data visualization tool.
  - R: Statistical computing and graphics.
  - Python: Versatile programming language for data analysis.
  - SAS: Advanced analytics and business intelligence software.
  - RapidMiner: Data science platform for machine learning and predictive analytics.
  - QlikView: Business intelligence and data visualization tool.
  - Excel: Widely used spreadsheet application for basic data analysis.
  - TensorFlow: Opensource library for machine learning and deep learning.

Glossary

- **Structured Data**: Organized and easily searchable data, typically stored in databases (e.g., SQL databases).
- **Semi-Structured Data**: Contains tags or markers to separate data elements but not strictly organized in relational databases (e.g., JSON, XML).
- **Unstructured Data**: Lacks a predefined format or organization, making it difficult to collect, process, and analyze (e.g., text documents, images).
- **Big Data Characteristics**: Volume, Velocity, Variety, Veracity, and Value.
- **Data Warehouse**: Central repository for structured data from various sources, optimized for query and analysis.
- **Hadoop**: Open-source framework for distributed processing of large data sets across clusters of computers using simple programming models.
- **Descriptive Analytics**: Analytics that describe what has happened.
- **Predictive Analytics**: Analytics that predict what might happen in the future.
- **Prescriptive Analytics**: Analytics that prescribe actions to achieve desired outcomes.
- **NoSQL**: Non-relational databases designed for large volumes of unstructured and semi-structured data.
- **Hadoop Ecosystem**: Includes tools like HDFS (Hadoop Distributed File System), MapReduce, YARN, and related projects like Spark, Hive, and Pig.

- **Data Scientist**: Professional skilled in statistics, programming, and domain expertise to analyze and interpret complex data.
- **HDFS (Hadoop Distributed File System)**: The primary storage system used by Pig in MapReduce mode. It allows Pig to read input files, store intermediate data, and write output files.

Checkup Your Progress

EXERCISE 1: Fill in each gap with the right word from the list

1. The three classifications of digital data are _____, semi-structured data, and unstructured data. (Structured, Categorized, Filtered)
2. Big Data is characterized by its volume, variety, velocity, _____, and value. (Veracity, Validity, Variability)
3. One of the primary challenges with Big Data is managing its vast _____. (Quantity, Quality, Quirkiness)
4. Traditional Business Intelligence typically involves the use of a _____, while Big Data often utilizes Hadoop for data processing. (, Data Lake, Data Mart) Data Warehouse
5. In Big Data environments, a term used to describe a system that is eventually consistent but not always immediately is _____. (BASE, ACID, CAP)

EXERCISE 2: Read the questions below and circle if the answer is True

1. Structured data is typically stored in databases that are difficult to search – True/ False
2. Big Data is defined solely by its large volume – True/ False
3. Traditional Business Intelligence focuses on predictive analytics- True/ False
4. Hadoop is a closed-source framework for data processing– True/ False
5. Data Science involves extracting insights from data using statistical methods- True/ False

EXERCISE 3: Choose the correct answer

1. The term used to describe the rapid increase in the volume of data is:

a) Velocity

b) Volume

c) Variety

2. Hadoop is an example of:

   a) Data Mart

   b) Data Warehouse

   c) Big Data technology

3. The primary language used for querying data in Hadoop is:

   a) SQL

   b) Pig Latin

   c) R

4. A data scientist's role typically includes:

   a) Data entry

   b) Data analysis and interpretation

   c) Database administration

5. In the context of Big Data, the term "velocity" refers to:

   a) The speed at which data is generated and processed

   b) The variety of data formats

   c) The accuracy of data

EXERCISE 4: Match the following

1.	Structured Data	-	The large amount of data generated
2.	Volume	-	Used in traditional Business Intelligence for structured data storage and analysis
3.	Data Warehouse	-	Has a defined data model and format
4.	BASE	-	A framework used in Big Data environments for processing large datasets

5. Hadoop - Eventually consistent, available, and partition-tolerant

EXERCISE 5: Self-Assessment Questions

1. Differentiate between structured, semi-structured, and unstructured data with examples in the context of digital data classification.
2. Explain two characteristics of Big Data and how they differ from traditional data management approaches.
3. List two challenges commonly associated with Big Data analytics.
4. Compare the roles of Data Science and traditional Business Intelligence in handling Big Data.
5. Define the concept of "Basically Available, Soft State, Eventual Consistency" (BASE) in the context of Big Data systems.

Answers for Checkup Your Progress

EXERCISE 1:

1. Structured 2. Veracity 3. Quantity 4. Data Warehouse 5. BASE.

EXERCISE 2:

1. False 2. False 3. False 4. False 5. True

EXERCISE 3:

1. b) 2. c) 3. b) 4. b) 5. a)

EXERCISE 4:

1.C, 2.A, 3.B, 4.E, 5.D.

Open Source E-content Links:

1. https://www.coursera.org/specializations/big-data
2. https://www.cloudera.com/services-and-support/tutorials.html
3. https://hadoop.apache.org/docs/current/
4. https://learn.mongodb.com/

References:

1. Classification of Digital Data: Structured, Semi-Structured, and Unstructured Data. (2023). Retrieved June 28, 2024, from https://www.investopedia.com/terms/u/unstructured-data.asp
2. Introduction to Big Data: Characteristics, Evolution, Definition, Challenges. (2023). Retrieved June 28, 2024, from https://www.sciencedirect.com/topics/computer-science/big-data
3. Big Data vs Traditional Business Intelligence: Comparison. (2023). Retrieved June 28, 2024, from https://www.forbes.com/sites/bernardmarr/2016/04/28/big-data-vs-traditional-business-intelligence-whats-the-difference/
4. Data Warehouse and Hadoop: Overview. (2023). Retrieved June 28, 2024, from https://www.ibm.com/analytics/learn/data-warehouse/
5. Environment of Big Data Analytics: Classification of Analytics, Challenges, Importance. (2023). Retrieved June 28, 2024, from https://www.analyticsvidhya.com/blog/2021/05/introduction-to-big-data-analytics/
6. Data Science and Data Scientist: Roles and Terminologies. (2023). Retrieved June 28, 2024, from https://www.kdnuggets.com/2020/02/data-science-vs-data-analytics-vs-machine-learning.html
7. Basically Available Soft State Eventual Consistency (BASE) Model. (2023). Retrieved June 28, 2024, from

https://www.academia.edu/37899493/BASE_Basically_Available_Soft_State_Eventual_Consistency_A_Model_for_Distributed_Systems

8. Top Analytics Tools: Overview. (2023). Retrieved June 28, 2024, from https://www.analyticsvidhya.com/blog/2021/06/top-10-analytics-tools-for-data-analysis/

UNIT-II:TECHNOLOGY LANDSCAPE

UNIT II OBJECTIVE

The objectives of this unit are to provide a comprehensive understanding of NoSQL databases, including their structure, types, and use cases. It aims to analyze the differences between SQL and NoSQL databases in terms of data models, schema flexibility, consistency models, and scalability. Additionally, the unit provides an overview of Hadoop, focusing on its core components such as HDFS, MapReduce, YARN, and common utilities for distributed data processing. Students will learn about the architecture, scalability, fault tolerance, and high throughput of HDFS, as well as how to process large datasets using the MapReduce programming model.

The unit also covers managing resources and applications with Hadoop YARN, exploring its resource management, job scheduling, and application management capabilities. Finally, it introduces key components of the Hadoop ecosystem, such as Hive, Pig, HBase, Spark, and Oozie, and teaches how to interact with Hadoop through CLI.

Unit Summary

1. **Introduction to NoSQL**
    - Comparison of SQL and NoSQL Databases
    - Overview of NoSQL Databases
2. **Hadoop Ecosystem**
    - Introduction to Hadoop
    - Differences between RDBMS and Hadoop
    - Distributed Computing Challenges
    - Detailed Overview of Hadoop Distributed File System (HDFS)
    - Processing Data with Hadoop
    - Managing Resources and Applications with Hadoop YARN
    - Interaction with Hadoop Ecosystem

2.1 NoSQL (NOT ONLY SQL)

The term NoSQL was first coined by Carlo Strozzi in 1998 to name his lightweight, open-source, relational database that did not expose the standard SQL interface. Johan Oskarsson, who was then a developer at last.fm, in 2009 reintroduced the term NoSQL at an event called to discuss open-source distributed network. The #NoSQL was coined by Eric Evans and few other database people at the event found it suitable to describe these non-relational databases.

Few features of NoSQL databases are as follows:

1. They are open source.
2. They are non-relational.
3. They are distributed.
4. They are schema-less.
5. They are cluster friendly.
6. They are born out of 21st century web applications.

2.1.1 Where is it Used?

NoSQL databases are widely used in big data and other real-time web applications. Refer Figure 2.1. NoSQL databases is used to stock log data which can then be pulled for analysis. Likewise it is used to store social media data and all such data which cannot be stored and analyzed comfortably in RDBMS.

2.1.2 What is it?

NoSQL stands for Not Only SQL. These are non-relational, open source, distributed databases. They are hugely popular today owing to their ability to scale out or scale horizontally and the adeptness at dealing with a rich variety of data; structured, semi-structured and unstructured data. Refer Figure 2.2 for additional features of NoSQL. NoSQL databases.

1. Are non-relational: They do not adhere to relational data model. In fact, they are either key-value pairs or document-oriented or column-oriented or graph-based databases.

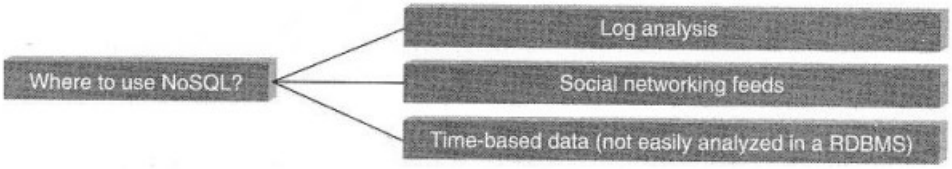


Figure 2.1: Where to use NoSQL?

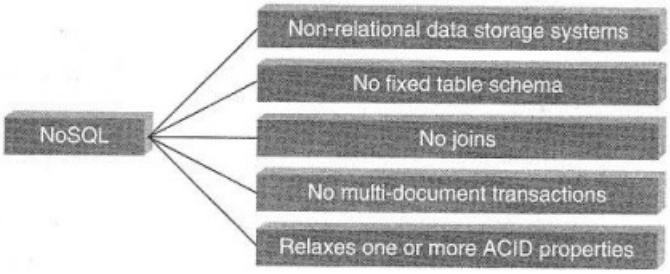


Figure 2.2: What is NoSQL?

2. Are distributed: They are distributed meaning the data is distributed across several nodes in a cluster constituted of low-cost commodity hardware.

3. Offer no support for ACID properties (Atomicity, Consistency, Isolation, and Durability): They do not offer support for ACID properties of transactions. On the contrary, they have adherence to Brewer's CAP (Consistency, Availability, and Partition tolerance) theorem and are often seen compromising on consistency in favor of availability and partition tolerance.

4. Provide no fixed table schema: NoSQL databases are becoming increasingly popular owing to their support for flexibility to the schema. They do not mandate for the data to strictly adhere to any schema structure at the time of storage.

2.1.3 Types of NoSQL Databases

We have already stated that NoSQL databases are non-relational. They can be broadly classified into the following: (Refer Figure 2.3).

1. Key-value or the big hash table.
2. Schema-less.

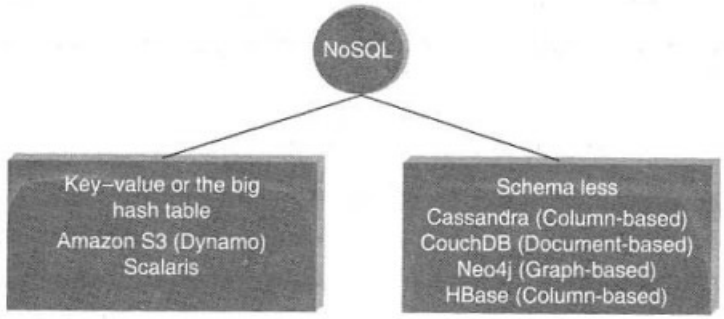


Figure 2.3: Types of NoSQL databases

Let us take a closer look at key-value and few other types of schema-less databases:

1**. Key-value:** It maintains a big hash table of keys and values. For example, Dynamo, Redis, Riak, etc.

Sample Key-Value Pair in Key-Value Database

Key	***Value***
First Name	Simmonds
Last Name	David

2. **Document:** It maintains data in collections constituted of documents. For example, MongoDB, Apache CouchDB, Couchbase, MarkLogic, etc.

Sample Document in Document Database

{
“Book Name”: “Fundamentals of Business Analytics”,
“Publisher”: “ Wiley India”,

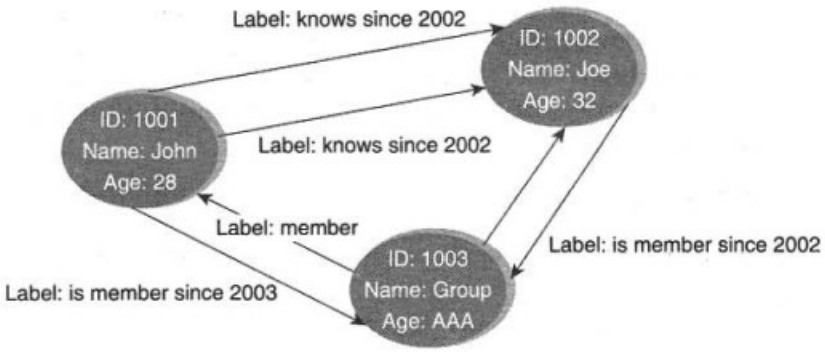
“ Year of Publication”: “2011”

}

3. Column: Each storage block has data from only one column. For example: Cassandra, HBase, etc.

4. Graph: They are also called network database. A graph stores data in nodes. For example, Neo4j, HyperGraphDB, etc.

Sample Graph in Graph Database



Refer Table 2.1 for popular schema-less databases.

Table 2.1 Popular schema-less databases

Key–Value Data Store	*Column-Oriented Data Store*	*Document Data Store*	*Graph Data Store*
Risk	Cassandra	MongoDB	InfiniteGraph
Redis	HBase	CouchDB	Neo4j
Membase	HyperTable	RavenDB	AllegroGraph

2.1.4 Why NoSQL?

1. It has scale out architecture instead of the monolithic architecture of relational databases.

2. It can house large volumes of structured, semi-structured, and unstructured data.

3. Dynamic schema: NoSQL database allows insertion of data without a pre-defined schema. In other words, it facilitates application changes in real time, which thus supports faster development, easy code integration, and requires less database administration.

4. Auto-sharding: It automatically spreads data across an arbitrary number of servers. The application in question is more often not even aware of the composition of the server pool. It balances the load of data and query on the available servers; and if and when a server goes down, it is quickly replaced without any major activity disruptions.

5. Replication: It offers good support for replication which in turn guarantees high availability, fault tolerance, and disaster recovery.

2.1.5 Advantages of NoSQL

Let us enumerate the advantages of NoSQL. Refer Figure 2.5.

1. **Can easily scale up and down:** NoSQL database supports scaling rapidly and elastically and even allows to scale to the cloud.

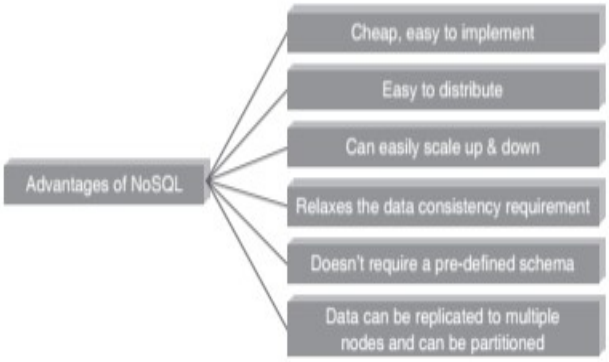


Figure 2.5: Advantages of NoSQL

(a) Cluster scale: It allows distribution of database across 100+ nodes often in multiple data centers.

(b) Performance scale: It sustains over 100,000+ database reads and writes per second.

(c) Data scale: It supports housing of 1 billion+ documents in the database.

2. Doesn't require a pre-defined schema: NoSQL does not require any adherence to pre-defined schema. It is pretty flexible. For example, if we look at MongoDB, the documents (equivalent of records in RDBMS) in a collection (equivalent of table in RDBMS) can have different sets of key–value pairs. {_id: 101,"BookName": "Fundamentals of Business Analytics", "AuthorName": "Seema Acharya", "Publisher": "Wiley India"} {_id:102, "BookName":"Big Data and Analytics"}

3. Cheap, easy to implement: Deploying NoSQL properly allows for all of the benefits of scale, high availability, fault tolerance, etc. while also lowering operational costs.

4. Relaxes the data consistency requirement: NoSQL databases have adherence to CAP theorem (Consistency, Availability, and Partition tolerance). Most of the NoSQL databases compromise on consistency in favor of availability and partition tolerance. However, they do go for eventual consistency.

5. Data can be replicated to multiple nodes and can be partitioned: There are two terms that are discussed here:

(a) Sharding: Sharding is when different pieces of data are distributed across multiple servers. NoSQL databases support auto-sharding; this means that they can natively and automatically spread data across an arbitrary number of servers, without requiring the application to even be aware of the composition of the server pool. Servers can be added or removed from the data layer without application downtime. This would mean that data and query load are automatically balanced across servers, and when a server goes down, it can be quickly and transparently replaced with no application disruption.

(b) Replication: Replication is when multiple copies of data are stored across the cluster and even across data centers. This promises high availability and fault tolerance.

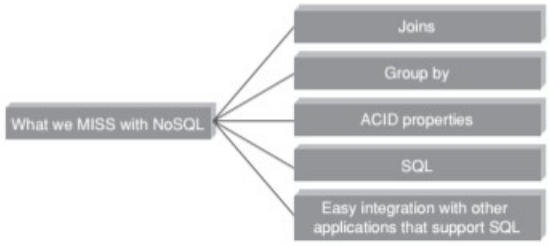


Figure 2.6: What we miss with NoSQL?

2.1.6 What We Miss with NoSQL? With NoSQL around, we have been able to counter the problem of scale (NoSQL scales out). There is also the flexibility with respect to schema design. However, there are few features of conventional RDBMS that are greatly missed. Refer Figure 2.6.

NoSQL does not support joins. However, it compensates for it by allowing embedded documents as in MongoDB. It does not have provision for ACID properties of transactions. However, it obeys the Eric Brewer's CAP theorem. NoSQL does not have a standard SQL interface but NoSQL databases such as MongoDB and Cassandra

have their own rich query language [MongoDB query language and Cassandra query language (CQL)] to compensate for the lack of it. One thing which is dearly missed is the easy integration with other applications that support SQL.

2.1.7 Use of NoSQL in Industry

NoSQL is being put to use in varied industries. They are used to support analysis for applications such as web user data analysis, log analysis, sensor feed analysis, making recommendations for upsell and cross-sell, etc. Refer Figure 2.7.

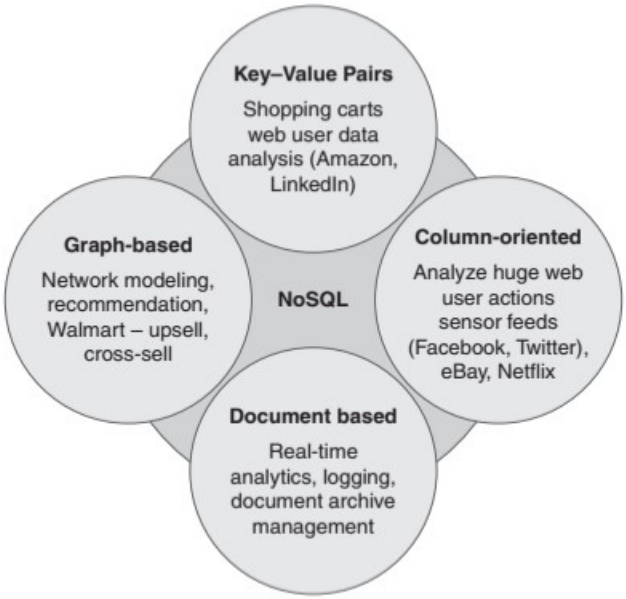


Figure 2.7: Use of NoSQL in industry

2.1.8 NoSQL Vendors

Refer Table 2.2 for few popular NoSQL vendors.

Table 2.2 : Few popular NoSQL vendors

Company	*Product*	*Most Widely Used by*
Amazon	DynamoDB	LinkedIn, Mozilla
Facebook	Cassandra	Netflix, Twitter, eBay

Google	BigTable	Adobe Photoshop

2.1.9 SQL versus NoSQL

Refer Table 2.3 for few salient differences between SQL and NoSQL.

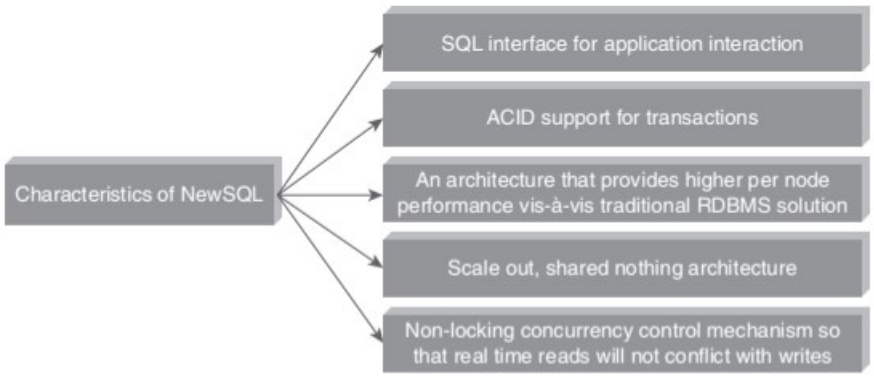


Figure 2.8 : Characterstics of NewSQL

Table 2.3 SQL versus NoSQL

SQL (Relational Database)	*NoSQL (Non-relational Database)*
Relational database	Non-relational, distributed database
Relational model	Model-less approach
Pre-defined schema	Dynamic schema for unstructured data
Table based databases	Document-based, graph-based, wide column store, key–value pairs
Vertically scalable (by increasing system resources)	Horizontally scalable (by creating a cluster of commodity machines)
Uses SQL	Uses UnQL (Unstructured Query Language)
Not preferred for large datasets	Largely preferred for large datasets
Not a best fit for hierarchical data	Best fit for hierarchical storage as it follows the key–value pair of storing data similar to JSON (Java Script Object

	Notation)
Emphasis on ACID properties	Follows Brewer's CAP theorem
Excellent support from vendors	Relies heavily on community support
Supports complex querying and data keeping needs	Does not have good support for complex querying
Can be configured for strong consistency	Few support strong consistency (e.g., MongoDB), some others can be configured for eventual consistency (e.g., Cassandra)
Examples: Oracle, DB2, MySQL, MS SQL, PostgreSQL, etc.	Examples: MongoDB, HBase, Cassandra, Redis, Neo4j, CouchDB, Couchbase, Riak, etc.

2.1.10 NewSQL

NewSQL is a modern type of database that combines the best features of both SQL and NoSQL databases. It aims to deliver the scalable performance of NoSQL systems for online transaction processing (OLTP) while maintaining the ACID (Atomicity, Consistency, Isolation, Durability) guarantees of traditional relational databases. NewSQL databases use the relational data model and SQL as their primary interface, providing the benefits of high scalability and robust transaction support.

2.1.10.1 Characteristics of NewSQL

Refer Figure 4.7 to learn about the characteristics of NewSQL. NewSQL is based on the shared nothing architecture with a SQL interface for application interaction.

2.1.11 Comparison of SQL, NoSQL, and NewSQL

Refer Table 2.4 for a comparative study of SQL, NoSQL and NewSQL.

Table 2.4 Comparative study of SQL, NoSQL and NewSQL

	SQL	*NoSQL*	*NewSQL*
Adherence to ACID properties	Yes	No	Yes
OLTP/OLAP	Yes	No	Yes
Schema rigidity	Yes	No	Maybe
Adherence to data model	Adherence to relational model		
Data Format Flexibility	No	Yes	Maybe
Scalability	Scale up (Vertical Scaling)	Scale out (Horizontal Scaling)	Scale out
Distributed Computing	Yes	Yes	Yes
Community Support	Huge	Growing	Slowly growing

Lets sum up

NoSQL databases, coined to denote "Not Only SQL," emerged as a response to the limitations of traditional relational databases in handling modern web applications and big data scenarios. They are characterized by their non-relational, distributed nature, allowing them to store and process vast amounts of structured, semi-structured, and unstructured data efficiently.

Unlike traditional SQL databases, NoSQL databases like key-value stores (e.g., DynamoDB), document stores (e.g., MongoDB), column-oriented databases (e.g., Cassandra), and graph databases (e.g., Neo4j) offer schema flexibility, horizontal scalability, and robust fault tolerance through features like auto-sharding and replication. While they sacrifice ACID properties for improved scalability and availability based on the CAP theorem, they excel in scenarios requiring high performance and massive data throughput, such as social media analytics, real-time data processing, and IoT applications. Despite their advantages, NoSQL databases lack standardized SQL

interfaces and may require more application-specific integration efforts compared to traditional relational databases. They represent a critical evolution in database technology, addressing new challenges posed by modern data management needs.

2.2 Hadoop

Hadoop is an open-source project of the Apache foundation. It is a framework written in Java, originally developed by Doug Cutting in 2005 who named it after his son's toy elephant. He was working with Yahoo then. It was created to support distribution for "Nutch", the text search engine. Hadoop uses Google's MapReduce and Google File System technologies as its foundation. Hadoop is now a core part of the computing infrastructure for companies such as Yahoo, Facebook, LinkedIn, Twitter, etc. Refer Figure 2.9.

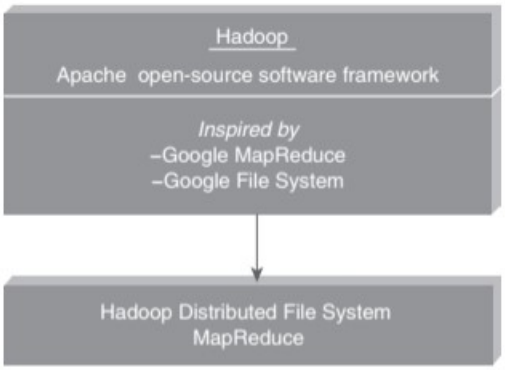


Figure 2.9 : Hadoop

2.2.1 Features of Hadoop

Here are a few key features of Hadoop:

1. It is optimized to handle massive quantities of structured, semi-structured, and unstructured data, using commodity hardware, that is, relatively inexpensive computers.
2. Hadoop has a shared nothing architecture.

3. It replicates its data across multiple computers so that if one goes down, the data can still be processed from another machine that stores its replica.
4. Hadoop is for high throughput rather than low latency. It is a batch operation handling massive quantities of data; therefore, the response time is not immediate.
5. It complements On-Line Transaction Processing (OLTP) and On-Line Analytical Processing (OLAP). However, it is not a replacement for a relational database management system.
6. It is NOT good when work cannot be parallelized or when there are dependencies within the data.
7. It is NOT optimized for processing small files; it works best with large data files and includes built-in replication for fault tolerance.

2.2.2 Key Advantages of Hadoop

Refer Figure 2.10 for a quick look at the key advantages of Hadoop. Some of them are as follows:

1. **Stores data in its native format:** Hadoop's data storage framework (HDFS – Hadoop Distributed File System) can store data in its native format. There is no structure that is imposed while keying in data or storing data. HDFS is pretty much schema-less. It is only later when the data needs to be processed that structure is imposed on the raw data.
2. **Scalable:** Hadoop can store and distribute very large datasets (involving thousands of terabytes of data) across hundreds of inexpensive servers that operate in parallel.

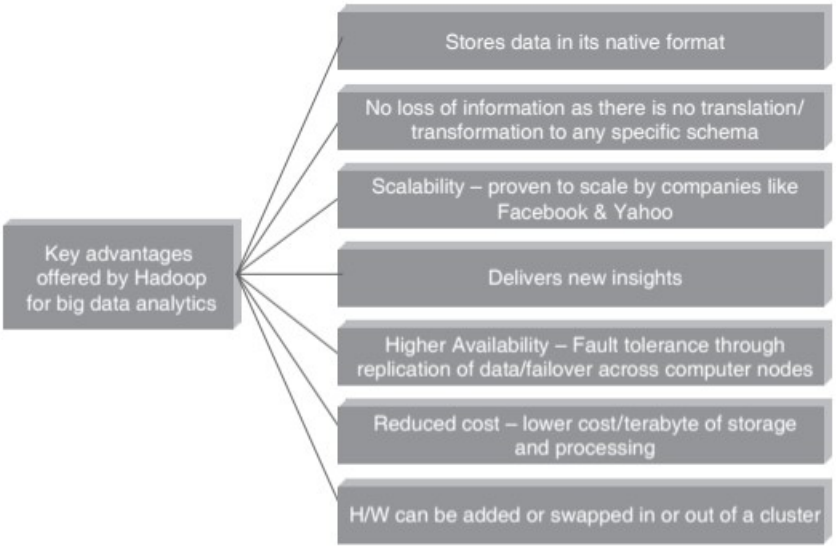


Figure 2.10: Key advantages of Hadoop.

3. **Cost-effective:** Owing to its scale-out architecture, Hadoop has a much-reduced cost/terabyte of storage and processing.
4. **Resilient to failure:** Hadoop is fault-tolerant. It practices replication of data diligently which means whenever data is sent to any node, the same data also gets replicated to other nodes in the cluster, thereby ensuring that in the event of a node failure, there will always be another copy of data available for use.
5. **Flexibility:** One of the key advantages of Hadoop is its ability to work with all kinds of data: structured, semi-structured, and unstructured data. It can help derive meaningful business insights from email conversations, social media data, click-stream data, etc. It can be put to several purposes such as log analysis, data mining, recommendation systems, market campaign analysis, etc.
6. **Fast:** Processing is extremely fast in Hadoop as compared to other conventional systems owing to the "move code to data" paradigm. Hadoop has a shared-nothing architecture.

2.2.3 Versions of Hadoop

There are two versions of Hadoop available (Refer Figure 2.11):

1. Hadoop 1.0
2. Hadoop 2.0

2.2.3.1 Hadoop 1.0

It has two main parts:

1. **Data storage framework:** It is a general-purpose file system called Hadoop Distributed File System (HDFS). HDFS is schema-less. It simply stores data files. These data files can be in just about any format. The idea is to store files as close to their original form as possible. This is turn provides the business units and the organization the much needed flexibility and agility without being overly worried by what it can implement.
2. **Data processing framework:** This is a simple functional programming model initially popularized by Google as MapReduce. It essentially uses two functions: the MAP and the REDUCE functions to process data. The "Mappers" take in a set of key–value pairs and generate intermediate data (which is another list of key–value pairs). The "Reducers" then act on this input to produce the output data. The two functions seemingly work in isolation from one another, thus enabling the processing to be highly distributed in a highly-parallel, fault-tolerant, and scalable way.

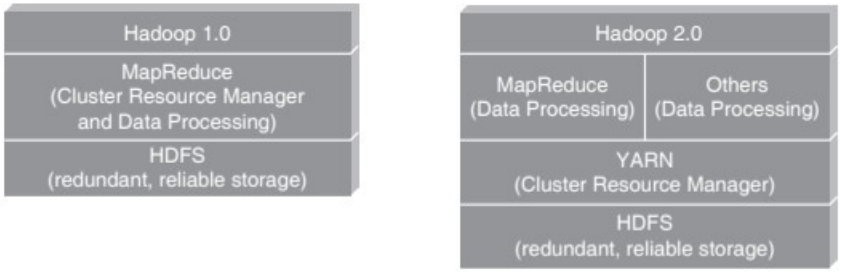


Figure 2.11: Versions of Hadoop

There were, however, a few limitations of Hadoop 1.0. They are as follows:

1. The first limitation was the requirement for MapReduce programming expertise along with proficiency required in other programming languages, notably Java.
2. It supported only batch processing which although is suitable for tasks such as log analysis, large-scale data mining projects but pretty much unsuitable for other kinds of projects.
3. One major limitation was that Hadoop 1.0 was tightly computationally coupled with MapReduce, which meant that the established data management vendors were left with two options: Either rewrite their functionality in MapReduce so that it could be executed in Hadoop or extract the data from HDFS and process it outside of Hadoop. None of the options were viable as it led to process inefficiencies caused by the data being moved in and out of the Hadoop cluster.

Now, let's discuss how Hadoop 2.0 addresses some of these limitations.

2.2.3.2 Hadoop 2.0

In Hadoop 2.0, HDFS continues to be the data storage framework. However, a new and separate resource management framework called Yet Another Resource Negotiator (YARN) has been added. Any application capable of dividing itself into parallel tasks is supported by YARN. YARN coordinates the allocation of subtasks of the submitted application, thereby further enhancing the flexibility, scalability, and efficiency of the applications. It works by having an ApplicationMaster in place of the erstwhile JobTracker, running applications on resources governed by a new NodeManager (in place of the erstwhile TaskTracker). ApplicationMaster is able to run any application and not just MapReduce.

This, in other words, means that the MapReduce Programming expertise is no longer required. Furthermore, it not only supports batch processing but also real-time processing. MapReduce is no longer the only data processing option; other alternative data processing functions such as data standardization, master data management can now be performed natively in HDFS.

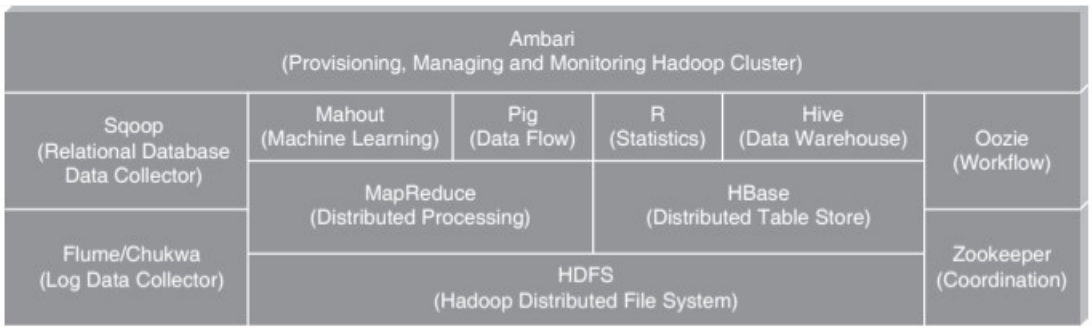


Figure 2.12: Hadoop ecosystem

2.2.4 Overview of Hadoop Ecosystems

The components of the Hadoop ecosystem are shown in Figure 2.12. There are components available in the Hadoop ecosystem for data ingestion, processing, and analysis.

Data Ingestion → Data Processing → Data Analysis

Components that help with **Data Ingestion** are:

1. Sqoop
2. Flume

Components that help with **Data Processing** are:

1. MapReduce
2. Spark

Components that help with **Data Analysis** are:

1. Pig
2. Hive
3. Impala

HDFS

It is the distributed storage unit of Hadoop. It provides streaming access to file system data as well as file permissions and authentication. It is based on GFS (Google File System). It is used to scale a single cluster node to hundreds and thousands of nodes. It handles large datasets running on commodity hardware. HDFS is highly fault-tolerant. It stores files across multiple machines. These files are stored in redundant fashion to allow for data recovery in case of failure.

HBase

It stores data in HDFS. It is the first non-batch component of the Hadoop Ecosystem. It is a database on top of HDFS. It provides a quick random access to the stored data. It has very low latency compared to HDFS. It is a NoSQL database, is non-relational and is a column-oriented database. A table can have thousands of columns. A table can have multiple rows. Each row can have several column families. Each column family can have several columns. Each column can have several key values. It is based on Google BigTable. This is widely used by Facebook, Twitter, Yahoo, etc.

Difference between HBase and Hadoop/HDFS

1. HDFS is the file system whereas HBase is a Hadoop database. It is like NTFS and MySQL.
2. HDFS is WORM (Write once and read multiple times or many times). Latest versions support appending of data but this feature is rarely used. However, HBase supports real-time random read and write.
3. HDFS is based on Google File System (GFS) whereas HBase is based on Google Big Table.
4. HDFS supports only full table scan or partition table scan. Hbase supports random small range scan or table scan.
5. Performance of Hive on HDFS is relatively very good but for HBase it becomes 4–5 times slower.
6. The access to data is via MapReduce job only in HDFS whereas in HBase the access is via Java APIs, Rest, Avro, Thrift APIs.

7. HDFS does not support dynamic storage owing to its rigid structure whereas HBase supports dynamic storage.
8. HDFS has high latency operations whereas HBase has low latency operations.
9. HDFS is most suitable for batch analytics whereas HBase is for real-time analytics.

Hadoop Ecosystem Components for Data Ingestion

1. **Sqoop:** Sqoop stands for SQL to Hadoop. Its main functions are

   a) Importing data from RDBMS such as MySQL, Oracle, DB2, etc. to Hadoop file system (HDFS, HBase, Hive).

   b) Exporting data from Hadoop File system (HDFS, HBase, Hive) to RDBMS (MySQL, Oracle, DB2).

   Uses of Sqoop

   a) It has a connector-based architecture to allow plug-ins to connect to external systems such as MySQL, Oracle, DB2, etc.

   b) It can provision the data from external system on to HDFS and populate tables in Hive and HBase.

   c) It integrates with Oozie allowing you to schedule and automate import and export tasks.

2. **Flume:** Flume is an important log aggregator (aggregates logs from different machines and places them in HDFS) component in the Hadoop ecosystem. Flume has been developed by Cloudera. It is designed for high volume ingestion of event-based data into Hadoop. The default destination in Flume (called as sink in flume parlance) is HDFS. However, it can also write to HBase or Solr.

Hadoop Ecosystem Components for Data Processing

1. **MapReduce:** It is a programing paradigm that allows distributed and parallel processing of huge datasets. It is based on Google MapReduce. Google

released a paper on MapReduce programming paradigm in 2004 and that became the genesis of Hadoop processing model. The MapReduce framework gets the input data from HDFS. There are two main phases: Map phase and the Reduce phase. The map phase converts the input data into another set of data (key–value pairs). This new intermediate dataset then serves as the input to the reduce phase. The reduce phase acts on the datasets to combine (aggregate and consolidate) and reduce them to a smaller set of tuples. The result is then stored back in HDFS.

2. **Spark:** It is both a programming model as well as a computing model. It is an open-source big data processing framework. It was originally developed in 2009 at UC Berkeley's AmpLab and became an open-source project in 2010. It is written in Scala. It provides in-memory computing for Hadoop. In Spark, workloads execute in memory rather than on disk owing to which it is much faster (10 to 100 times) than when the workload is executed on disk. However, if the datasets are too large to fit into the available system memory, it can perform conventional disk-based processing. It serves as a potentially faster and more flexible alternative to MapReduce. It accesses data from HDFS (Spark does not have its own distributed file system) but bypasses the MapReduce processing.

Spark can be used with Hadoop coexisting smoothly with MapReduce (sitting on top of Hadoop YARN) or used independently of Hadoop (standalone). As a programming model, it works well with Scala, Python (it has API connectors for using it with Java or Python) or R programming language. The following are the Spark libraries:

a) **Spark SQL:** Spark also has support for SQL. Spark SQL uses SQL to help query data stored in disparate applications.
b) **Spark streaming:** It helps to analyze and present data in real time.
c) **MLib:** It supports machine learning such as applying advanced statistical operations on data in Spark Cluster.
d) **GraphX:** It helps in graph parallel computation.

Spark and Hadoop are usually used together by several companies. Hadoop was primarily designed to house unstructured data and run batch processing operations on it. Spark is used extensively for its high speed in memory computing and ability to run advanced real-time analytics. The two together have been giving very good results.

Hadoop Ecosystem Components for Data Analysis

1. Pig: It is a high-level scripting language used with Hadoop. It serves as an alternative to MapReduce. It has two parts:

    a) **Pig Latin:** It is SQL-like scripting language. Pig Latin scripts are translated into MapReduce jobs which can then run-on YARN and process data in the HDFS cluster. It was initially developed by Yahoo. It is immensely popular with developers who are not comfortable with MapReduce. However, SQL developers may have a preference for Hive. How it works? There is a "Load" command available to load the data from "HDFS" into Pig. Then one can perform functions such as grouping, filtering, sorting, joining etc. The processed or computed data can then be either displayed on screen or placed back into HDFS. It gives you a platform for building data flow for ETL (Extract, Transform and Load), processing and analyzing huge data sets.

    b) **Pig runtime:** It is the runtime environment.

2. **Hive:** Hive is a data warehouse software project built on top of Hadoop. Three main tasks performed by Hive are summarization, querying and analysis. It supports queries written in a language called HQL or HiveQL which is a declarative SQL-like language. It converts the SQL-style queries into MapReduce jobs which are then executed on the Hadoop platform.

Difference between Hive and RDBMS

Both Hive and traditional databases such as MySQL, MS SQL Server, and PostgreSQL support SQL interfaces. However, while traditional databases are typically used as

transactional databases, Hive is better known as a data warehouse (D/W). Hive and traditional databases differ in their approach to schema enforcement, usage patterns, suitability for OLTP and OLAP, data analysis capabilities, scalability, and computing paradigm:

- **Schema Enforcement:** Hive enforces schema on read time, whereas traditional databases enforce it on write time.
- **Usage:** Hive is designed for scenarios where data is written once and read many times, whereas traditional databases support frequent read and write operations.
- **OLTP and OLAP Suitability:** Due to its batch-oriented nature, Hive is more suitable for OLAP, while traditional databases excel in OLTP with real-time operations.
- **Data Analysis:** Hive is well-suited for static data analysis with slower query response times, while traditional databases are more adept at handling dynamic, real-time data.
- **Scalability and Cost:** Hive can be scaled at a lower cost compared to traditional databases due to its use of HDFS. Traditional databases typically incur higher costs for storage and management.
- **Computing Paradigm:** Hive leverages parallel computing, enhancing its performance and scalability, while traditional databases rely on serial computing, which can impact scalability in certain scenarios.

The summary of their difference is given in Table 2.5.

Table 2.5: Hive versus RDBMS

Aspect	Hadoop	RDBMS
Data Variety	Used for structured, semi-structured, and unstructured data. Supports a variety of data formats in real time such as XML, JSON, and text-based flat file formats.	Used for structured data

Data Storage	Usually datasets of size terabytes, petabytes	Usually, datasets of size gigabytes
Querying	HiveQL	SQL
Query Response	Latency due to batch processing	Immediate query response time
Schema	Schema required on read	Schema required on write
Speed	Writes are faster compared to reads as there is no adherence to schema required at the time of inserting or writing data. Schema is enforced at read time.	Reads are very fast (supported by building indexes on required columns). Designed for read and write many times.
Cost	Apache Hadoop is open-source, large-scale, distributed, scalable, data-intensive computing.	Available as proprietary RDBMS such as Oracle, MS SQL Server, IBM DB2, etc. Also open-source RDBMS are available such as MySQL, PostgreSQL, etc.
Use Cases	Analytics, data discovery	OLTP (Online Transaction Processing). Mainly used to store and process day-to-day business data.
Throughput	High	Low
Scalability	Horizontal (Hadoop scales by adding nodes to a Hadoop cluster of easily available commodity machines).	Vertical (RDBMS scales vertically by increasing the horsepower (CPU, Hard Disk Capacity, RAM, etc.) of the machine).
Hardware	Commodity/Utility Hardware	High-End Servers
Integrity	Low	High. Obeys ACID properties: A – Atomicity; C – Consistency ; I – Isolation ; D – Durability

Difference between Hive and HBase

1. Hive is a MapReduce-based SQL engine that runs on top of Hadoop. HBase is a key–value NoSQL database that runs on top of HDFS.
2. Hive is for batch processing of big data. HBase is for real-time data streaming.

Impala*:* It is a high-performance SQL engine that runs on Hadoop cluster. It is ideal for interactive analysis. It has very low latency measured in milliseconds. It supports a dialect of SQL called Impala SQL.

ZooKeeper: It is a coordination service for distributed applications.

Oozie: It is a workflow scheduler system to manage Apache Hadoop jobs.

Mahout : It is a scalable machine learning and data mining library.

Chukwa : It is a data collection system for managing large distributed systems.

Ambari : It is a web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters.

2.2.5 Hadoop Distributions

Hadoop is an open-source Apache project. Anyone can freely download the core aspects of Hadoop. The core aspects of Hadoop include the following:

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

There are few companies such as IBM, Amazon Web Services, Microsoft, Teradata, Hortonworks, Cloudera, etc. that have packaged Hadoop into a more easily consumable distributions or services. Although each of these companies have a slightly different strategy, the key essence remains its ability to distribute data and workloads across

potentially thousands of servers thus making big data manageable data. A few Hadoop distributions are given in Figure 2.13.

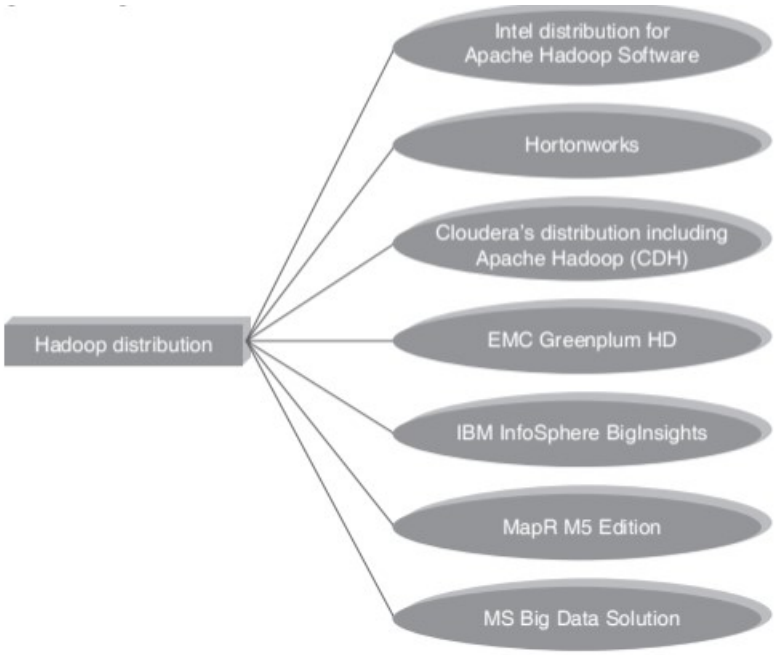


Figure 2.13: Hadoop Distribution

2.2.6 Hadoop versus SQL

Table 2.6 lists the differences between Hadoop and SQL.

Table 2.6 Hadoop versus SQL

Hadoop	*SQL*
Scale out	Scale up
Key–Value pairs	Relational table
Functional Programming	Declarative Queries
Offline batch processing	Online transaction processing

2.2.7 Integrated Hadoop Systems Offered by Leading Market Vendors

Refer Figure 2.14 to get a glimpse of the leading market vendors offering integrated Hadoop systems.

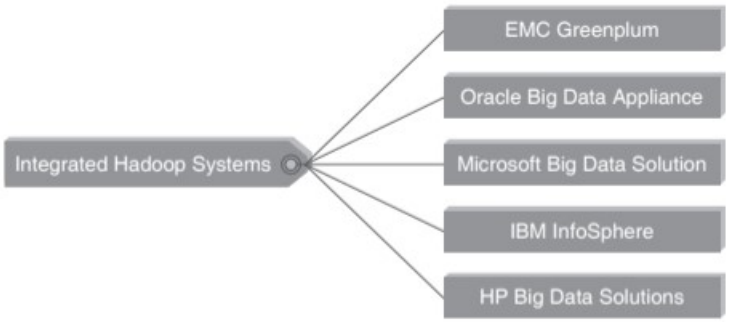


Figure 2.14: Integrated Hadoop systems.

2.2.8 Cloud-Based Hadoop Solutions

Amazon Web Services holds out a comprehensive, end-to-end portfolio of cloud computing services to help manage big data. The aim is to achieve this and more along with retaining the emphasis on reducing costs, scaling to meet demand, and accelerating the speed of innovation. The Google Cloud Storage connector for Hadoop empowers one to perform MapReduce jobs directly on data in Google Cloud Storage, without the need to copy it to local disk and running it in the Hadoop Distributed File System (HDFS). The connector simplifies Hadoop deployment, and at the same time reduces cost and provides performance comparable to HDFS, all this while increasing reliability by eliminating the single point of failure of the name node. Refer Figure 2.15.

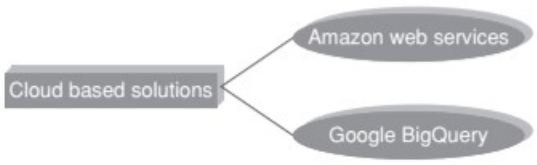


Figure 2.15:Cloud-based solutions

2.3 Introducing Hadoop

Today, Big Data seems to be the buzz word! Enterprises, the world over, are beginning to realize that there is a huge volume of untapped information before them in the form of structured, semi-structured, and unstructured data. This varied variety of data is spread across the networks.

Let us look at few statistics to get an idea of the amount of data which gets generated every day, every minute, and every second.

1. **Every day:**

a) NYSE (New York Stock Exchange) generates 1.5 billion shares and trade data.
b) Facebook stores 2.7 billion comments and likes.
c) Google processes about 24 petabytes of data.

2. **Every minute:**

a) Facebook users share nearly 2.5 million pieces of content.
b) Twitter users tweet nearly 300,000 times.
c) Instagram users post nearly 220,000 new photos.
d) YouTube users upload 72 hours of new video content.
e) Apple users download nearly 50,000 apps.
f) Email users send over 200 million messages.
g) Amazon generates over $80,000 in online sales.
h) Google receives over 4 million search queries.

3. **Every second:**

a) Banking applications process more than 10,000 credit card transactions.

2.3.1 Data: The Treasure Trove

1. Provides business advantages such as generating product recommendations, inventing new products, analyzing the market, and many, many more…
2. Provides few early key indicators that can turn the fortune of business.
3. Provides room for precise analysis. If we have more data for analysis, then we have greater precision of analysis.

To process, analyze, and make sense of these different kinds of data, we need a system that scales and addresses the challenges shown in Figure 2.16

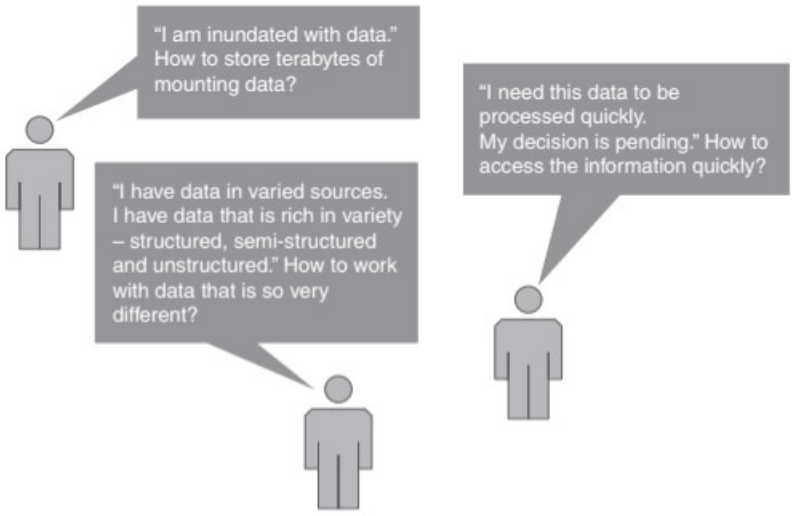


Figure 2.16: Challenges with big volume, variety, and velocity of data.

2.3.2 Why Hadoop?

Hadoop can handle large volumes of data, as well as diverse categories of data, quickly and efficiently. Hadoop's popularity stems from several key factors (Refer Figure 2.17):

1. **Handling Massive Data:** Hadoop efficiently manages huge volumes and different types of data, ensuring quick processing.
2. **Low Cost:** Being open-source and utilizing commodity hardware makes Hadoop a cost-effective solution for storing vast amounts of data.

3. **Computing Power:** Hadoop's distributed computing model enables it to process large data volumes swiftly, with more computing nodes translating to increased processing power.
4. **Scalability:** It's easy to scale Hadoop by simply adding nodes as needed, requiring minimal administration effort.
5. **Storage Flexibility:** Unlike traditional databases, Hadoop allows storing data without prior processing. It offers the freedom to store any amount of data and decide later how to utilize it, including unstructured data like images, videos, and text.
6. **Inherent Data Protection:** Hadoop safeguards data and applications against hardware failures. It automatically redistributes tasks from failed nodes to functional ones and stores multiple copies of data across the cluster to prevent loss.

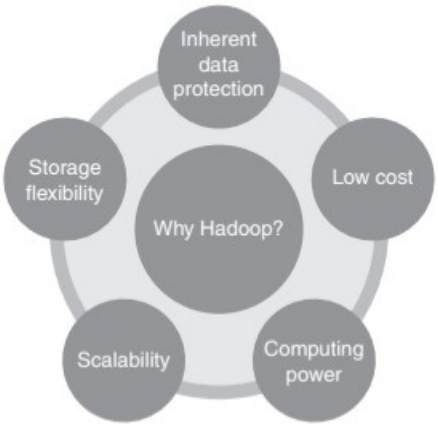


Figure 2.17 Key considerations of Hadoop.

Hadoop makes use of commodity hardware, distributed file system, and distributed computing as shown in Figure 2.18. In this new design, groups of machines are gathered together; it is known as a Cluster.

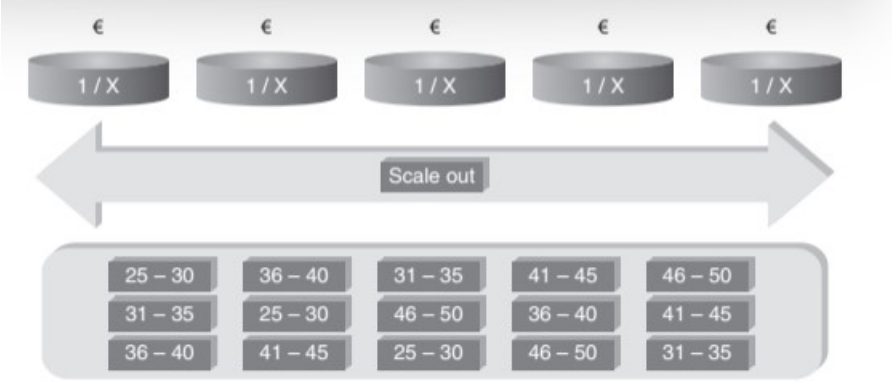


Figure 2.18: Hadoop framework (distributed file system, commodity hardware).

With this new paradigm, the data can be managed with Hadoop as follows:

1. Distributes the data and duplicates chunks of each data file across several nodes, for example, 25–30 is one chunk of data as shown in Figure 2.18.
2. Locally available compute resource is used to process each chunk of data in parallel.
3. Hadoop Framework handles failover smartly and automatically.

2.3.3 Why NOT RDBMS?

RDBMS is not suitable for storing and processing large files, images, and videos. RDBMS is not a good choice when it comes to advanced analytics involving machine learning. Figure 2.19 describes the RDBMS system with respect to cost and storage. It calls for huge investment as the volume of data shows an upward trend.

Figure 2.19 RDBMS with respect to cost/GB of storage.

2.3.4 RDBMS versus Hadoop

Table 2.7 describes the difference between RDBMS and Hadoop.

Table 2.7 RDBMS versus Hadoop

PARAMETERS	*RDBMS*	*HADOOP*
System	Relational Database Management System	Node Based Flat Structure
Data	Suitable for structured data	Suitable for structured, unstructured data. Supports a variety of data formats in real time such as XML, JSON, text-based flat file formats, etc.
Processing	OLTP	Analytical, Big Data Processing
Choice	When the data needs a consistent relationship	Big Data processing, which does not require any consistent relationships between data.
Processor	Needs expensive hardware or high-end processors to store huge volumes of data.	In a Hadoop Cluster, a node requires only a processor, a network card, and a few hard drives.
Cost	Cost around $10,000 to	Cost around $4,000 per terabyte of

	$14,000 per terabyte of storage.	storage.

2.3.5 Distributed Computing Challenges

This section is focusing on two major challenges in distributed computing.

2.3.5.1 Hardware Failure

- Distributed systems face frequent hardware failures due to the interconnected nature of servers.
- Hadoop addresses this challenge with Replication Factor (RF), ensuring data redundancy across the network.
- RF indicates the number of data copies stored for each data item or block, reducing the risk of data loss. (Refer to figure 2.20)

2.3.5.2 How to Process This Gigantic Store of Data?

- Data in distributed systems is distributed across multiple machines, posing a challenge for integration and processing.
- Hadoop employs MapReduce Programming to address this challenge effectively.
- MapReduce breaks down data processing tasks into smaller units, distributes them across the cluster, and consolidates the results.

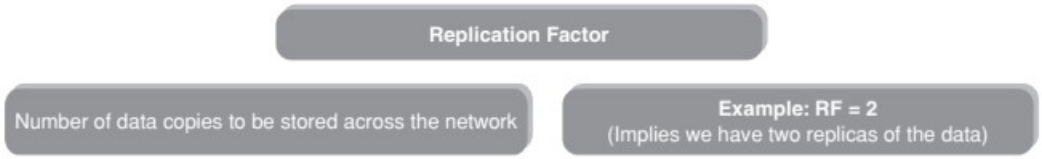


Figure 2.20 Replication factor

2.4 History of Hadoop

Hadoop was created by Doug Cutting, the creator of Apache Lucene (a commonly used text search library). Hadoop is a part of the Apache Nutch (Yahoo) project (an open-source web search engine) and also a part of the Lucene project. Refer Figure 5.6 for more details.

2.4.1 The Name "Hadoop"

The name Hadoop is not an acronym; it's a made-up name. The project creator, Doug Cutting, explains how the name came about: *"The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term".*

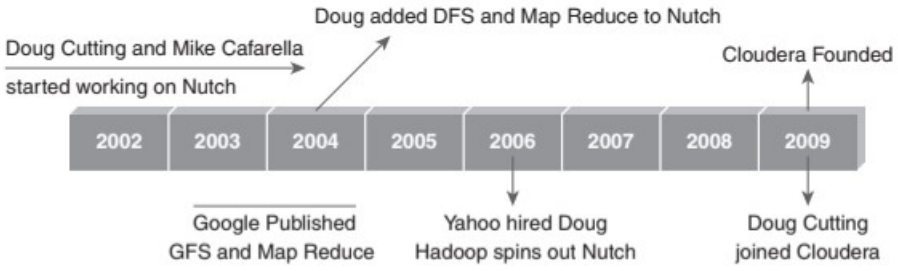


Figure 2.21 Hadoop history.

Subprojects and "contrib" modules in Hadoop also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig", for example).

2.4.2 Hadoop Overview

Open-source software framework to store and process massive amounts of data in a distributed fashion on large clusters of commodity hardware. Basically, Hadoop accomplishes two tasks:

1. Massive data storage.

2. Faster data processing.

2.4.3 Key Aspects of Hadoop

Figure 2.22 describes the key aspects of Hadoop.

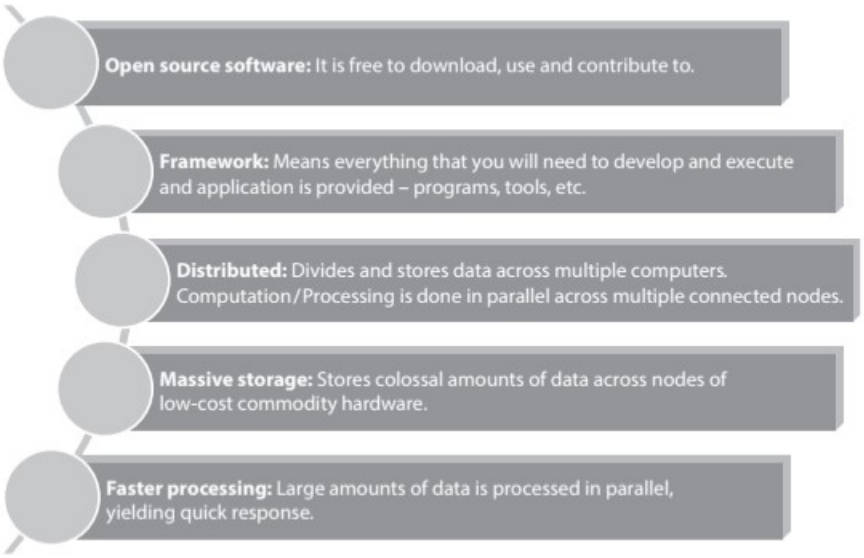


Figure 2.22 Key aspects of Hadoop.

2.4.4 Hadoop Components

Figure 2.23 depicts the Hadoop components.

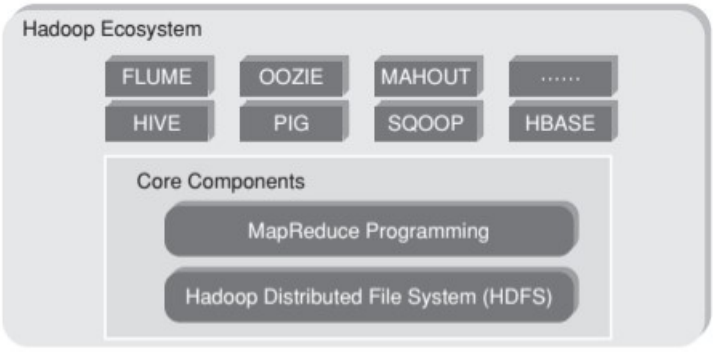


Figure 2.23 Hadoop components.

Hadoop Core Components

1. HDFS:

    a) Storage component.
    b) Distributes data across several nodes.
    c) Natively redundant.

2. MapReduce:

    a) Computational framework.
    b) Splits a task across multiple nodes.
    c) Processes data in parallel.

Hadoop Ecosystem: Hadoop Ecosystem are support projects to enhance the functionality of Hadoop Core Components. The Eco Projects are as follows:

1. HIVE
2. PIG
3. SQOOP
4. HBASE
5. FLUME
6. OOZIE
7. MAHOUT

2.4.5 Hadoop Conceptual Layer

It is conceptually divided into Data Storage Layer which stores huge volumes of data and Data Processing Layer which processes data in parallel to extract richer and meaningful insights from data (Refer Figure 2.24).

2.4.6 High-Level Architecture of Hadoop

Hadoop is a distributed **Master-Slave** Architecture. Master node is known as **NameNode** and slave nodes are known as **DataNodes**. Figure 2.25 depicts the Master–Slave Architecture of Hadoop Framework.

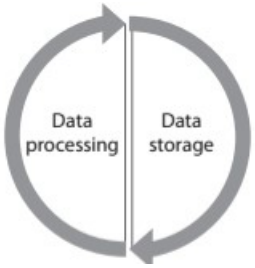


Figure 2.24 Hadoop conceptual layer.

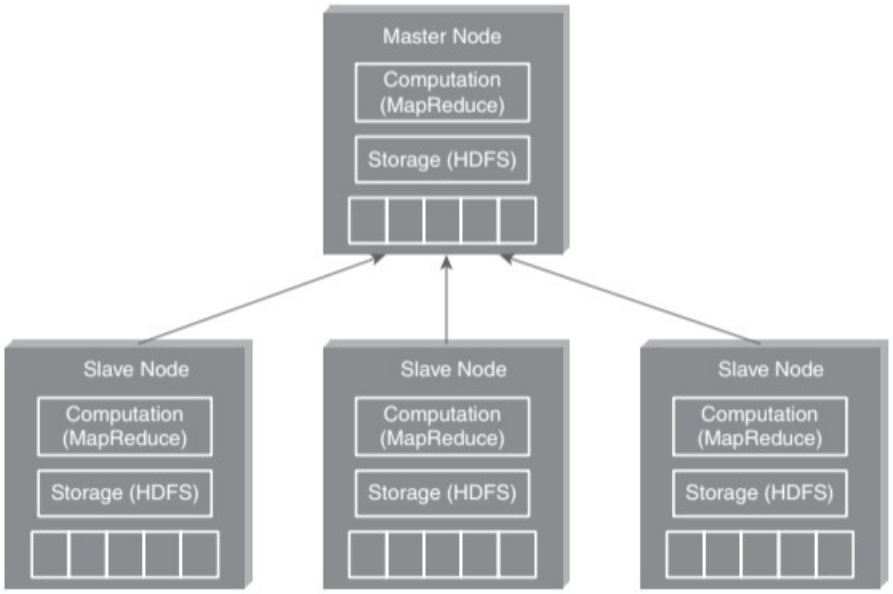


Figure 2.25 Hadoop high-level architecture. (Reference: Hadoop in Practice, Alex Holmes**)**

Key components of the Master Node:

1. **Master HDFS:** Its main responsibility is partitioning the data storage across the slave nodes. It also keeps track of locations of data on DataNodes.
2. **Master MapReduce:** It decides and schedules computation task on slave nodes.

2.5 Use Case of Hadoop : ClickStream Data

ClickStream data (mouseclicks) helps you to understand the purchasing behavior of customers. ClickStream analysis helps online marketers optimize their product web pages, promotional content, etc. to improve their business.

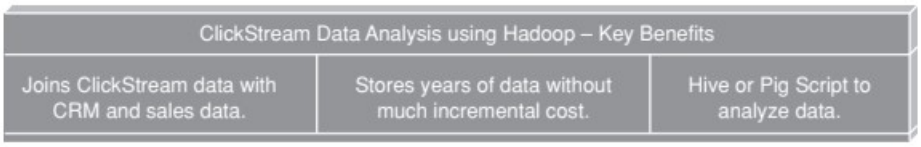


Figure 2.26 ClickStream data analysis.

The ClickStream analysis (Figure 2.26) using Hadoop provides three key benefits:

1. Hadoop helps to join ClickStream data with other data sources such as Customer Relationship Management Data (Customer Demographics Data, Sales Data, and Information on Advertising Campaigns). This additional data often provides the much needed information to understand customer behavior.
2. Hadoop's scalability property helps you to store years of data without ample incremental cost. This helps you to perform temporal or year over year analysis on ClickStream data which your competitors may miss.
3. Business analysts can use Apache Pig or Apache Hive for website analysis. With these tools, you can organize ClickStream data by user session, refine it, and feed it to visualization or analytics tools. (Reference:http://hortonworks.com/wp-content/uploads/2014/05/Hortonworks.BusinessValueofHadoop. v1.0.pdf)

2.6 Hadoop Distributors

The companies shown in Figure 2.27 provide products that include Apache Hadoop, commercial support, and/or tools and utilities related to Hadoop.

Figure 2.27 Common Hadoop distributors.

2.7 HDFS (Hadoop Distributed File System)

Some key Points of Hadoop Distributed File System are as follows:

1. Storage component of Hadoop.
2. Distributed File System.
3. Modeled after Google File System.
4. Optimized for high throughput (HDFS leverages large block size and moves computation where data is stored).
5. You can replicate a file for a configured number of times, which is tolerant in terms of both software and hardware.
6. Re-replicates data blocks automatically on nodes that have failed.
7. You can realize the power of HDFS when you perform read or write on large files (gigabytes and larger).
8. Sits on top of native file system such as ext3 and ext4, which is described in Figure 2.28.

Figure 2.29 describes important key points of HDFS. Figure 2.30 describes Hadoop Distributed File System Architecture. Client Application interacts with NameNode for metadata related activities and communicates with DataNodes to read and write files. DataNodes converse with each other for pipeline reads and writes.

Let us assume that the file “Sample.txt” is of size 192 MB. As per the default data block size(64 MB), it will be split into three blocks and replicated across the nodes on the cluster based on the default replication factor.

2.7.1 HDFS Daemons

2.7.1.1 NameNode

HDFS breaks a large file into smaller pieces called blocks. NameNode uses a rack ID to identify DataNodes in the rack. A rack is a collection of DataNodes within the cluster. NameNode keeps tracks of blocks of a file as it is placed on various DataNodes. NameNode manages file-related operations such as read, write, create, and delete. Its main job is managing the File System Namespace. A file system namespace is collection of files in the cluster. NameNode stores HDFS namespace. File system namespace includes mapping of blocks to file, file properties and is stored in a file called FsImage. NameNode uses an EditLog (transaction log) to record every transaction that happens to the filesystem metadata. Refer Figure 2.31. When NameNodestarts up, it reads FsImage and EditLog from disk and applies all transactions from the EditLog to in-memory representation of the FsImage. Then it flushes out new version of FsImage on disk and truncates the old EditLog because the changes are updated in the FsImage. There is a single NameNode per cluster. (Reference: http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html)

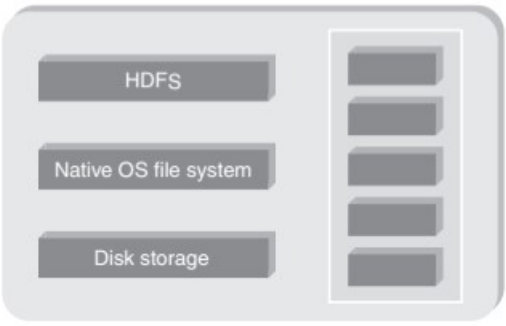


Figure 2.28 Hadoop Distributed File System.

Figure 2.29 Hadoop Distributed File System – key points.

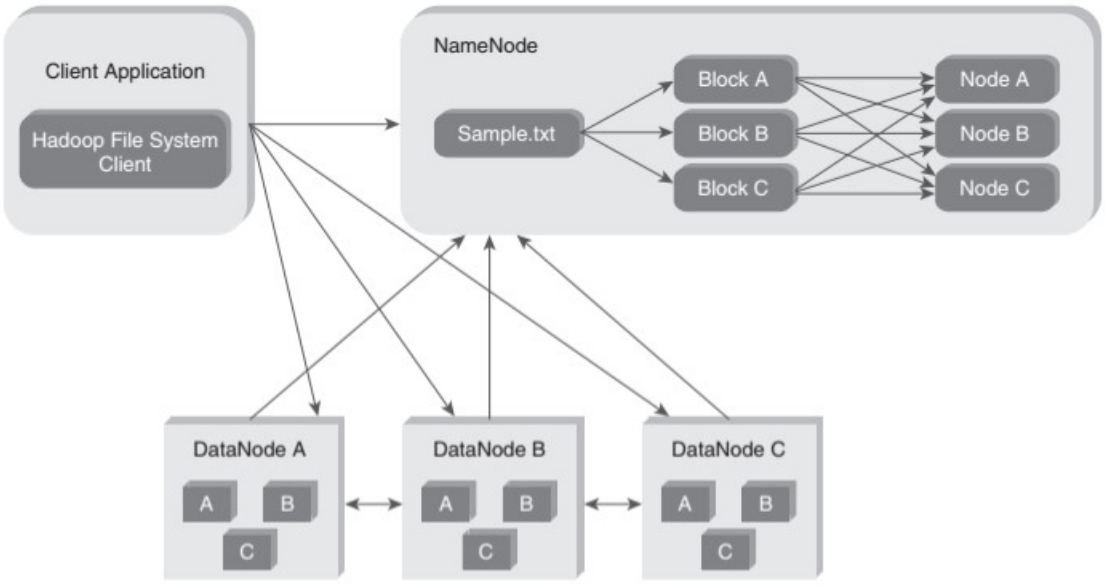


Figure 2.30 Hadoop Distributed File System Architecture.
Reference: Hadoop in Practice, Alex Holmes

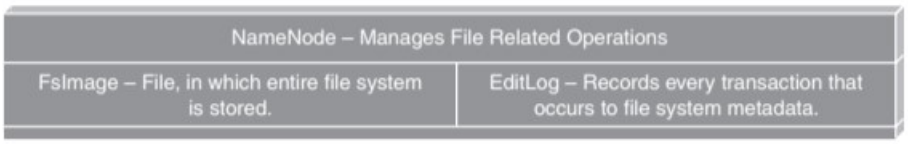


Figure 2.31 NameNode.

2.7.1.2 DataNode

There are multiple DataNodes per cluster. During Pipeline read and write DataNodes communicate with each other. A DataNode also continuously sends **"heartbeat"** message to NameNode to ensure the connectivity between the NameNode and DataNode. In case there is no heartbeat from a DataNode, the NameNode replicates that DataNode within the cluster and keeps on running as if nothing had happened. Let us explain the concept behind sending the heartbeat report by the DataNodes to the

NameNode. (*Reference:* Wrox Certified Big Data Developer.) Communications of name node and data node Refer figure 2.32.

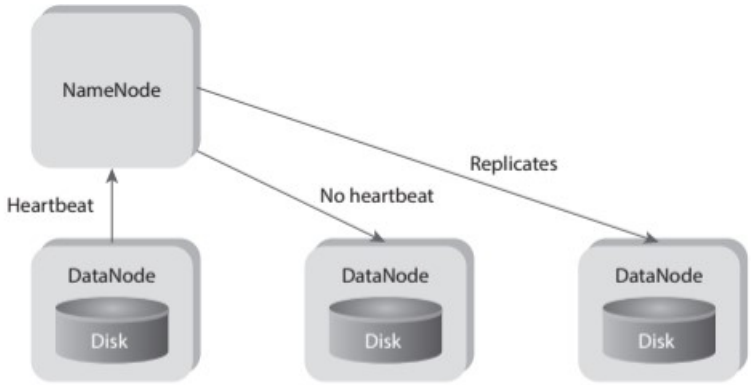


Figure 2.32 NameNode and DataNode Communication.

2.7.1.3 Secondary NameNode

The Secondary NameNode takes a snapshot of HDFS metadata at intervals specified in the Hadoop configuration. Since the memory requirements of Secondary NameNode are the same as NameNode, it is better to run NameNode and Secondary NameNode on different machines. In case of failure of the NameNode, the Secondary NameNode can be configured manually to bring up the cluster. However, the Secondary NameNode does not record any real-time changes that happen to the HDFS metadata.

2.8 Anatomy of File Read

Figure 2.33 describes the anatomy of File Read. The steps involved in reading a file in Hadoop's DistributedFileSystem (HDFS):

1. **Open File:** The client opens the file it wants to read by calling open() on DistributedFileSystem.
2. **Get Data Block Locations:** DistributedFileSystem communicates with the NameNode to get the locations of data blocks, which are stored on DataNodes.

3. **Create InputStream:** The client receives an FSDataInputStream to read from the file.
4. **Read Data:** The client reads data from the DFSInputStream by calling read(). It connects to the closest DataNode for the first block of the file.
5. **Streaming Data:** The client continues reading data by repeatedly calling read() until the end of the block is reached. Once done, it closes the connection with the DataNode and repeats the process for subsequent blocks.
6. **Close Connection:** After reading the entire file, the client calls close() on FSDataInputStream to close the connection.

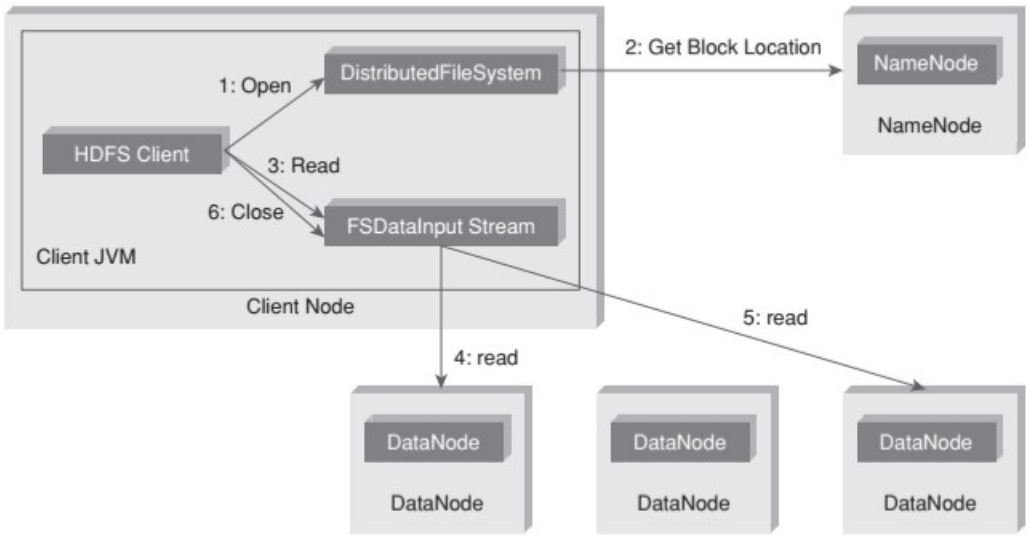


Figure 2.33 File Read.

2.8.1 Anatomy of File Write

Figure 2.34 describes the anatomy of File Write. The steps involved in writing a file in Hadoop's DistributedFileSystem(HDFS):

1. **Create File:** The client creates a file by calling create() on DistributedFileSystem.
2. **Request to NameNode:** An RPC call is made to the NameNode through DistributedFileSystem to create a new file. The NameNode performs checks and

creates the file without data blocks initially. The client receives an FSDataOutputStream for writing.

3. **Write Data:** As the client writes data, it's split into packets by DFSOutputStream and added to a data queue. DataStreamer consumes this queue and requests the NameNode to allocate new blocks by selecting suitable DataNodes, forming a pipeline.
4. **Stream Data:** DataStreamer streams packets to the first DataNode in the pipeline. Each DataNode stores and forwards the packet to the next. This process continues until all replicas are stored.
5. **Acknowledge Queue:** DFSOutputStream manages an "Ack queue" of packets awaiting acknowledgment from DataNodes. A packet is removed from this queue only after acknowledgment from all DataNodes in the pipeline.
6. **Close Stream:** After writing the file, the client calls close() on the stream.
7. **Finalize Write:** Remaining packets are flushed to the DataNode pipeline, and acknowledgments are awaited. Once all acknowledgments are received, the client informs the NameNode that the file creation is complete.

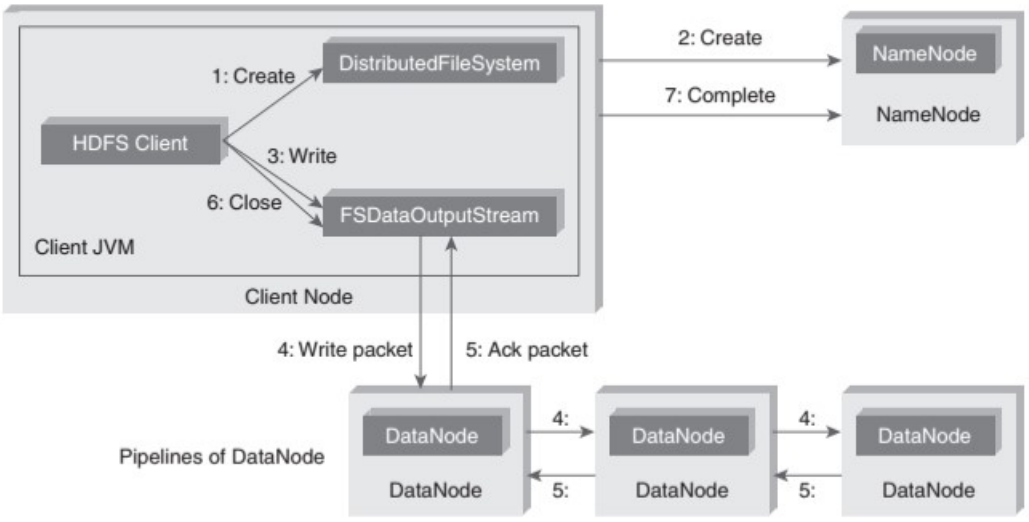


Figure 2.34 File Write.

2.8.2 Replica Placement Strategy

2.8.2.1 Hadoop Default Replica Placement Strategy

As per the Hadoop Replica Placement Strategy, first replica is placed on the same node as the client. Then it places second replica on a node that is present on different rack. It places the third replica on the same rack as second, but on a different node in the rack. Once replica locations have been set, a pipeline is built. This strategy provides good reliability. Figure 2.35 describes the typical replica pipeline. (*Reference:* Hadoop, the Definite Guide, 3rd Edition, O'Reilly Publication).

2.9 Working with HDFS Commands

Objective: To get the list of directories and files at the root of HDFS.
Act: hadoop fs -ls/

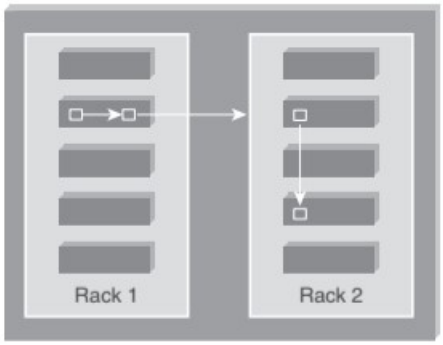


Figure 2.35 Replica Placement Strategy.

Objective: To get the list of complete directories and files of HDFS.
Act: hadoop fs -ls -R/

Objective: To create a directory (say, sample) in HDFS.
Act: hadoop fs -mkdir /sample

Objective: To copy a file from local file system to HDFS.
Act: hadoop fs -put /root/sample/test.txt /sample/test.txt

Objective: To copy a file from HDFS to local file system.
Act: hadoop fs -get /sample/test.txt /root/sample/testsample.txt

Objective: To copy a file from local file system to HDFS via copyFromLocal command.

Act: hadoop fs -copyFromLocal /root/sample/test.txt /sample/testsample.txt

Objective: To copy a file from Hadoop file system to local file system via copyToLocal command.

Act: hadoop fs -copyToLocal /sample/test.txt /root/sample/testsample1.txt

Objective: To display the contents of an HDFS file on console.

Act: hadoop fs -cat /sample/test.txt

Objective: To copy a file from one directory to another on HDFS.

Act: hadoop fs -cp /sample/test.txt /sample1

Objective: To remove a directory from HDFS.

Act: hadoop fs-rm-r /sample1

2.9.1 Special Features of HDFS

1. **Data Replication:** There is absolutely no need for a client application to track all blocks. It directs the client to the nearest replica to ensure high performance.
2. **Data Pipeline:** A client application writes a block to the first DataNode in the pipeline. Then this DataNode takes over and forwards the data to the next node in the pipeline. This process continues for all the data blocks, and subsequently all the replicas are written to the disk.

Reference: Wrox Certified Big Data Developer.

2.10 Processing Data with Hadoop

MapReduce Programming is a software framework. MapReduce Programming is a framework used to process large amounts of data in parallel. Here's how it works:

1. **Input Splitting:** The input data is divided into independent chunks, and map tasks process these chunks in parallel.
2. **Mapping:** Each map task processes its chunk of data and produces intermediate results, which are stored locally on the server.
3. **Shuffling and Sorting:** The framework automatically shuffles and sorts the intermediate data based on keys.
4. **Reducing:** Reduce tasks combine the intermediate data from multiple maps to produce the final output.
5. **Data Locality:** Hadoop's Distributed File System and MapReduce Framework run on the same nodes, allowing tasks to be scheduled where the data resides, resulting in high throughput.
6. **JobTracker and TaskTracker:** MapReduce has two daemons - JobTracker (master) and TaskTracker (slave). JobTracker schedules tasks to TaskTrackers, monitors task execution, and re-executes failed tasks.
7. **Job Submission:** MapReduce applications are implemented via suitable interfaces, and job parameters are configured. The job client submits the job to the JobTracker, which schedules tasks to slaves and monitors their execution. MapReduce programming phases and daemons Refer figure 2.36.

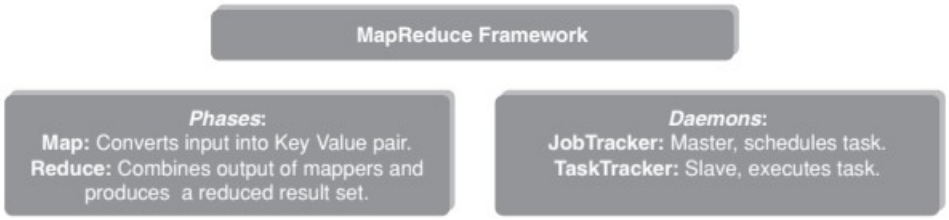


Figure 2.36 MapReduce Programming phases and daemons.

Reference: http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html

2.10.1 MapReduce Daemons

In MapReduce, there are two key daemons:

1. **JobTracker:**
    - The JobTracker is the master daemon responsible for managing and coordinating MapReduce jobs in the cluster.
    - It schedules tasks to TaskTrackers, monitors their execution, and handles task failures by re-executing them as necessary.
    - JobTracker ensures efficient utilization of cluster resources and maintains overall job execution status.
2. **TaskTracker:**
    - TaskTracker is a slave daemon running on each cluster node.
    - It executes map and reduce tasks assigned by the JobTracker.
    - TaskTracker communicates with the JobTracker to report task status and request new tasks.
    - It manages task execution, including task initialization, progress monitoring, and cleanup upon task completion or failure.

These daemons work together to efficiently process data using the MapReduce framework in a distributed Hadoop cluster.

Lets sum up

The Hadoop Ecosystem offers a robust framework for handling large-scale data processing and analysis, distinctly differing from traditional RDBMS by emphasizing distributed computing across clusters of commodity hardware. Key challenges in distributed computing, such as fault tolerance and data distribution, are adeptly managed by Hadoop's architecture. Central to this is the Hadoop Distributed File System (HDFS), which provides high-throughput access to data and fault tolerance. Data processing is efficiently handled using Hadoop's MapReduce paradigm, while resource management and application coordination are streamlined through Hadoop YARN. The ecosystem's interaction extends to a suite of tools and technologies that integrate seamlessly, enhancing its versatility and functionality for big data applications.

UNIT 2 : SUMMARY

- **NoSQL**
  - Nonrelational databases designed for large volumes of unstructured and semi-structured data.
  - Offers flexibility, scalability, and high performance.
  - Types: Document stores (e.g., MongoDB), key-value stores (e.g., Redis), column-family stores (e.g., Cassandra), and graph databases (e.g., Neo4j).
- **Comparison of SQL and NoSQL**
  - **SQL Databases**: Use structured schema and ACID properties for consistency.
  - **NoSQL Databases**: Provide schemaless design, BASE properties, and horizontal scalability.
  - **ACID**: Atomicity, Consistency, Isolation, Durability.
  - **BASE**: Basically Available, Soft state, Eventual consistency.
- **Hadoop**
  - An open-source framework for distributed storage and processing of big data.
  - Utilizes the MapReduce programming model.
  - Key components: Hadoop Distributed File System (HDFS) and MapReduce.
- **RDBMS vs. Hadoop**
  - **RDBMS**: Suited for structured data with ACID compliance.
  - **Hadoop**: Handles vast amounts of varied data with a distributed approach and eventual consistency.
  - RDBMS is typically used for OLTP (Online Transaction Processing), whereas Hadoop is used for OLAP (Online Analytical Processing).
- **Distributed Computing Challenges**
  - Issues include data distribution, fault tolerance, resource management, and efficient processing across distributed systems.

- Network latency and bandwidth constraints.
- Data consistency and synchronization across nodes.

- **Hadoop Overview**
  - Consists of modules for storing and processing large datasets, primarily using HDFS and MapReduce.
  - Hadoop ecosystem includes projects like Hive, Pig, HBase, and Spark for various big data tasks.
- **Hadoop Distributed File System (HDFS)**
  - A scalable, fault-tolerant storage system that splits data into blocks distributed across multiple nodes.
  - Replication of data blocks to ensure fault tolerance.
- **Processing Data with Hadoop**
  - Utilizes MapReduce for parallel processing, breaking tasks into smaller subtasks processed across the cluster.
  - JobTracker and TaskTracker manage and monitor tasks and resource allocation.
- **Managing Resources and Applications with Hadoop YARN**
  - YARN (Yet Another Resource Negotiator) manages resources and schedules jobs.
  - Improves cluster utilization and scalability.
  - Separates resource management and job scheduling/monitoring functions.
- **Interacting with the Hadoop Ecosystem**
  - Tools and frameworks like Hive, Pig, HBase, and Spark are used to perform data analytics, processing, and storage tasks within the Hadoop environment.
  - **Hive**: Data warehouse infrastructure built on Hadoop.
  - **Pig**: Platform for analyzing large data sets with a high-level scripting language.
  - **HBase**: Distributed, scalable, big data store.
  - **Spark**: Fast, in-memory data processing engine.
- **Query Processing and Optimization**

- Efficient query processing techniques.
- Query optimization strategies to enhance performance in a distributed environment.

- **Data Integration**
  - Techniques for integrating data from various sources.
  - Ensuring data quality and consistency during integration.
- **Security and Privacy**
  - Implementing security measures for data protection.
  - Ensuring compliance with data privacy regulations.
- **Scalability and Performance**
  - Techniques to scale systems horizontally.
  - Performance tuning and optimization practices for big data systems.
- **Future Trends**
  - Emerging technologies and trends in big data and distributed computing.
  - The role of AI and machine learning in enhancing big data analytics.

Glossary

- **NoSQL**: Databases designed for handling unstructured data (e.g., MongoDB, Cassandra).
- **SQL**: Structured Query Language, used for managing and manipulating relational databases.
- **ACID Properties**: Atomicity, Consistency, Isolation, Durability.
- **BASE Properties**: Basically Available, Soft state, Eventual consistency.
- **HDFS**: Hadoop Distributed File System, a scalable, fault-tolerant storage system.
- **MapReduce**: Programming model for processing and generating large datasets with a parallel, distributed algorithm on a cluster.
- **YARN**: Yet Another Resource Negotiator, manages resources and schedules jobs in Hadoop.
- **Hive:** Data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.
    - **Pig:** High-level platform for creating programs that run on Hadoop.

Checkup Your Progress

EXERCISE 1: Fill in each gap with the right word from the list

1. NoSQL databases are designed to handle large volumes of data with diverse _____ requirements. (structured, consistency, rapid)
2. SQL databases are typically used for applications requiring strong _____ and structured data, whereas NoSQL databases excel in scenarios requiring flexibility and scalability. (consistency, schema, performance)
3. Hadoop is an open-source framework designed for distributed storage and _____ of large datasets across clusters of computers. (processing, analysis, security)
4. The Hadoop Distributed File System (HDFS) is designed to store data reliably even on _____ hardware and provide high throughput access to data. (commodity, enterprise, cloud)
5. Hadoop YARN (Yet Another Resource Negotiator) is responsible for managing resources and scheduling tasks on a Hadoop _____ to optimize performance. (cluster, node, server)

EXERCISE 2: Read the questions below and circle if the answer is True

1. NoSQL databases are primarily designed for handling structured data with rigid schemas.
2. Hadoop is a distributed computing framework designed for processing and analyzing large datasets across clusters of computers.
3. Hadoop Distributed File System (HDFS) ensures high availability and fault tolerance by replicating data across multiple nodes in a cluster.
4. Hadoop YARN (Yet Another Resource Negotiator) is responsible for managing resources and scheduling tasks in a Hadoop cluster to optimize performance.
5. SQL databases and NoSQL databases serve similar purposes and can be used interchangeably in all types of applications.

EXERCISE 3: Choose the correct answer

1. **NoSQL databases are designed primarily for:**

   a) Handling structured data with rigid schemas

   b) Rapid development and scalability of unstructured data

   c) Ensuring transactional consistency

2. **Hadoop is best described as:**

   a) A relational database management system (RDBMS) for structured data

   b) A distributed computing framework for processing large datasets

   c) A real-time data processing tool

3. **Hadoop Distributed File System (HDFS) ensures fault tolerance by:**

   a) Storing data in a single location for quick access

   b) Replicating data across multiple nodes in a cluster

   c) Encrypting data to prevent unauthorized access

4. **Hadoop YARN (Yet Another Resource Negotiator) is responsible for:**

   a) Managing data storage in HDFS

   b) Processing complex SQL queries

   c) Managing resources and scheduling tasks in a Hadoop cluster

5. **SQL databases are typically preferred for applications that require:**

   a) Flexible schema and scalability

b) Strong consistency and structured data

c) Real-time data processing

EXERCISE 4: Match the following

1.	NoSQL databases	-	A. Designed for handling large volumes of unstructured data
2.	Hadoop Distributed File System (HDFS)	-	B. Responsible for managing resources and scheduling tasks in a Hadoop cluster
3.	Hadoop YARN (Yet Another Resource Negotiator)	-	C. Ensures fault tolerance by replicating data across multiple nodes
4.	Distributed computing	-	D. Primarily used for structured data with strong consistency requirements
5.	SQL databases	-	E. Framework for processing large datasets across clusters of computers

EXERCISE 5: Self-Assessment Questions

1. What is one key difference between SQL and NoSQL databases in terms of data schema flexibility?
2. What is the primary purpose of the Hadoop Distributed File System (HDFS)?
3. Name one major challenge faced in distributed computing that Hadoop aims to address.
4. Briefly explain what a MapReduce job in Hadoop entails.
5. What is the role of YARN in the Hadoop ecosystem?

Answers for Checkup Your Progress

EXERCISE 1:

1. Consistency 2. Consistency 3. Processing 4. Commodity 5. cluster

EXERCISE 2:

1. False 2. True 3. True 4. True 5. False

EXERCISE 3:

1. b) 2. b) 3. b) 4. c) 5. b)

EXERCISE 4:

1.A, 2.C, 3.B, 4.E, 5.D.

Open Source E-content Links:

1. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
2. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
3. https://hadoop.apache.org/docs/current/
4. https://www.mongodb.com/resources/basics/databases/nosql-explained

References:

1. NoSQL Databases Overview. (2023). Retrieved June 28, 2024, from https://www.mongodb.com/nosql-explained
2. SQL vs NoSQL Databases: What's the Difference? (2023). Retrieved June 28, 2024, from https://www.mongodb.com/sql-vs-nosql

3. Apache Hadoop. (2023). Retrieved June 28, 2024, from https://hadoop.apache.org/
4. Hadoop Distributed File System (HDFS) Architecture Guide. (2023). Retrieved June 28, 2024, from https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
5. Apache YARN Documentation. (2023). Retrieved June 28, 2024, from https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

UNIT-III MONGODB AND MAP REDUCE PROGRAMMING

UNIT III: OBJECTIVE

This unit objective is to provide a comprehensive understanding of MongoDB and MapReduce programming. In MongoDB, students will compare terms used in relational databases with MongoDB, explore various data types supported by MongoDB, and learn the MongoDB Query Language (MQL) for querying databases efficiently. On the other hand, in MapReduce programming, students will delve into essential components such as Mapper, Reducer, Combiner, and Partitioner, understanding their roles in processing and aggregating data. Additionally, students will learn about searching, sorting, and compression techniques in MapReduce, enabling efficient data analysis and processing

Unit Summary

1. MongoDB

- Introduction to MongoDB
- Comparison of Terms used in RDBMS and MongoDB
- Data Types in MongoDB
- MongoDB Query Language

2. MapReduce Programming

- Introduction to MapReduce
- Components of MapReduce: Mapper, Reducer, Combiner, Partitioner

Techniques for Searching, Sorting, and Compression

3.1 What is MongoDB?

MongoDB is

1.Cross-platform.

2. Open source.
3. Non-relational.
4. Distributed.
5. No.
6. Document-oriented data store.

3.2 Why MongoDB?

Few of the major challenges with traditional RDBMS are dealing with large volumes of data, rich variety of data – particularly unstructured data, and meeting up to the scale needs of enterprise data. The need is for a database that can scale out or scale horizontally to meet the scale requirements, has flexibility with respect to schema, is fault tolerant, is consistent and partition tolerant, and can be easily distributed over a multitude of nodes in a cluster. Refer Figure 3.1.

3.2.1 Using Script Object Notation (JSON)

JSON is extremely expressive. MongoDB actually does not use JSON but BSON (pronounced Bee Son) − it is Binary JSON. It is an open standard. It is used to store complex data structures. MongoDB Document oriented

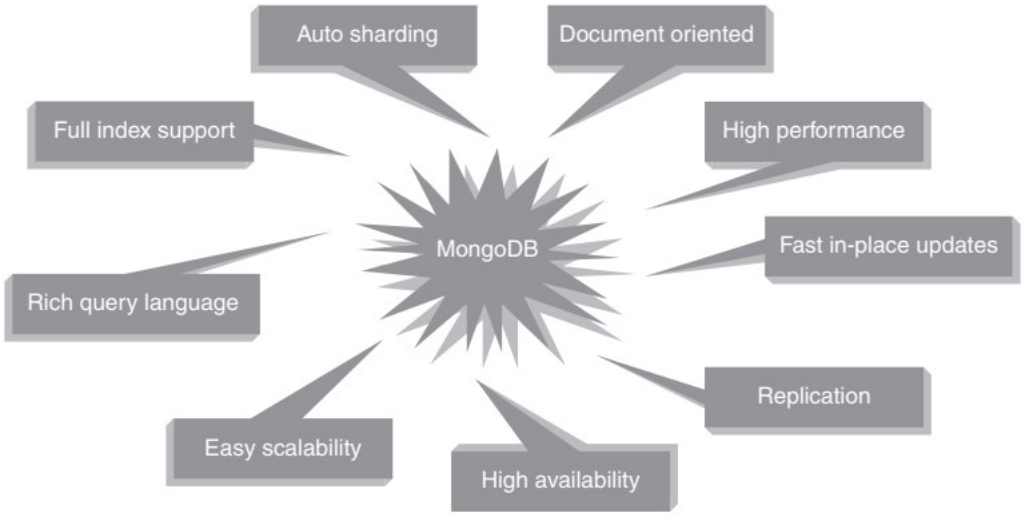


Figure 3.1 Why MongoDB?

3.2.2 Creating or Generating a Unique Key

Each JSON document should have a unique identifier. It is the _id key. It is similar to the primary key in relational databases. This facilitates search for documents based on the unique identifier. An index is automatically built on the unique identifier. The user has the option to either provide unique values themselves or allow the Mongo shell to generate them.

0	1	2	3	4	5	6	7	8	9	10	11
Timestamp				Machine ID			Process ID		Counter		

3.2.2.1 Database

It is a collection of collections. In other words, it is like a container for collections. It gets created the first time that your collection makes a reference to it. This can also be created on demand. Each database gets its own set of files on the file system. A single MongoDB server can house several databases.

3.2.2.2 Collection

A collection is analogous to a table of RDBMS. A collection is created on demand. It gets created the first time that you attempt to save a document that references it. A collection exists within a single database. A collection holds several MongoDB documents. A collection does not enforce a schema. This implies that documents within a collection can have different fields. Even if the documents within a collection have same fields, the order of the fields can be different.

3.2.2.3 Document

A document is analogous to a row/record/tuple in an RDBMS table. A document has a dynamic schema. This implies that a document in a collection need not necessarily have the same set of fields/key–value pairs. Shown in Figure 3.2 is a collection by the name “students” containing three documents.

3.2.3 Support for Dynamic Queries

MongoDB has extensive support for dynamic queries. This is in keeping with traditional RDBMS wherein we have static data and dynamic queries. CouchDB, another document-oriented, schema-less No database and MongoDB’s biggest competitor,

works on quite the reverse philosophy. It has support for dynamic data and static queries.

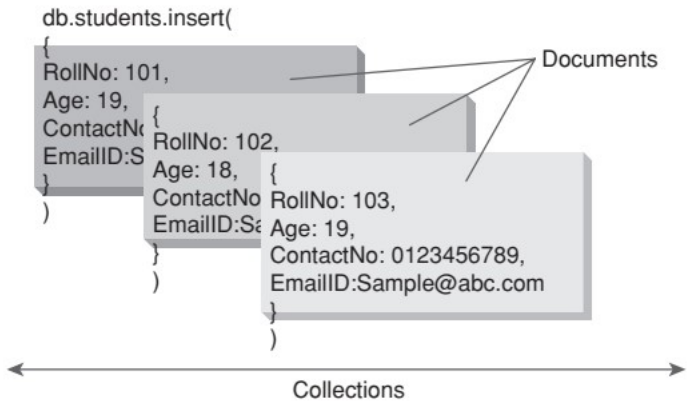


Figure 3.2 A Collection "students" containing 3 documents.

3.2.4 Storing Binary Data

MongoDB offers GridFS to handle binary data storage, capable of storing files up to 4 MB in size, suitable for small images or audio clips like profile pictures. For larger files such as movie clips, MongoDB uses a different approach. It stores metadata, or information about the data, in a collection named "file" and breaks down the actual data into smaller chunks stored in the "chunks" collection. This method ensures easy scalability, allowing MongoDB to efficiently manage large files by distributing them across multiple smaller pieces.

3.2.5 Replication

Why replication? Replication is essential in databases like MongoDB for several reasons. Firstly, it provides data redundancy and high availability, ensuring that copies of data are stored across multiple servers. This redundancy helps in recovering from hardware failures and service interruptions, as there are alternative copies of the data available. In MongoDB's replica set architecture, there is a single primary server responsible for handling write requests from clients. The primary logs all write operations into its Oplog (operations log), which is then used by secondary replica members to synchronize their data. This ensures strict adherence to consistency across the replica set. While clients typically read from the primary server for up-to-date data, they can also specify a read preference to direct read operations to secondary servers if needed, providing flexibility in accessing data. (Refer Figure 3.3)

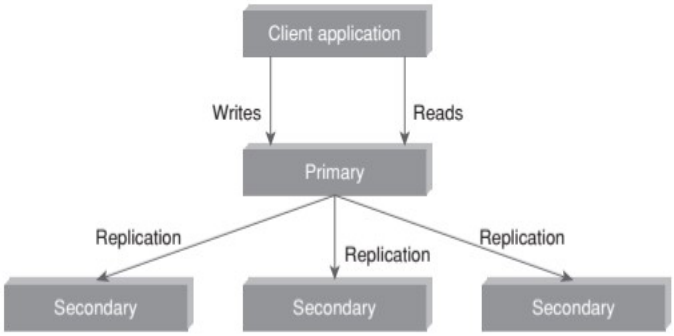


Figure 3.3 Replication Process in MongoDB

3.2.6 Sharding

Sharding is akin to horizontal scaling. It means that the large dataset is divided and distributed over multiple servers or shards. Each shard is an independent database and collectively they would constitute a logical database.

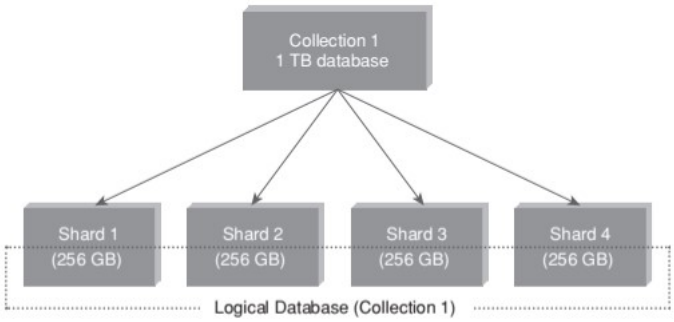


Figure 3.4 Sharding Process in MongoDB

The prime advantages of sharding are as follows:

1. Sharding reduces the amount of data that each shard needs to store and manage. For example, if the dataset was 1 TB in size and we were to distribute this over four shards, each shard would house just 256 GB data. Refer Figure 3.4. As the cluster grows, the amount of data that each shard will store and manage will decrease.
2. Sharding reduces the number of operations that each shard handles. For example, if we were to insert data, the application needs to access only that shard which houses that data.

3.2.7 Updating Information In-Place

MongoDB updates data directly where it's stored, without allocating separate space or altering indexes. It prefers lazy-writes, meaning it writes to disk only once every second, as disk operations are slower than memory operations. This approach improves performance by reducing disk reads and writes. However, MongoDB doesn't guarantee immediate storage of data on disk, which is a tradeoff for its speed compared to competitors that write data to disk immediately.

3.3 Terms Used in RDBMS and MongoDB

RDBMS vs MongoDB Terminology Comparison

RDBMS	*MongoDB*
Database	Database
Table	Collection
Record	Document
Columns	Fields/Key-Value pairs
Index	Index
Joins	Embedded documents
Primary Key	Primary key (_id is a unique identifier)
My	Oracle
Database Server	Myd
Database Client	My

3.3.1 Create Database in MongoDB

Creating a database in MongoDB is straightforward. Below are the commands and their usage with examples.

Syntax for Creating a Database

To create a new database, use the following syntax:

use DATABASE_Name

Example

To create a database named myDB, use the following command:

db

This will return the name of the current database you are using, which should be myDB.

Listing All Databases

To get a list of all databases in your MongoDB server, use the following command:

show dbs

This command will list all the databases along with their sizes.

Example Session

Here is an example session that demonstrates these commands:

shell
> use myDB
switched to db myDB
> db
myDB

> show dbs
admin 0.000GB
config 0.000GB

local 0.000GB

In this example:

- The command use myDB switches the context to myDB.
- The command db confirms that the current database is myDB.
- The command show dbs lists all databases. Note that myDB might not appear in the list until it contains some data because MongoDB creates databases lazily.

This simple sequence of commands helps in managing and verifying database creation in MongoDB.

3.3.2 Drop Database in MongoDB

The syntax to drop a database is as follows:

db.dropDatabase();

To drop the database “myDB”, first ensure that you are currently using the “myDB” database and then use the db.dropDatabase() command.

use myDB;
db.dropDatabase();

Example Session:

> use myDB
switched to db myDB
> db.dropDatabase();
{ "dropped" : "myDB", "ok" : 1 }
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB

If no database is selected, the default database “test” is dropped.

3.4 Datatypes in MongoDB

Data Type	*Description*
String	Must be UTF-8 valid. Most commonly used data type to store data.
Integer	Used to store numerical values. Can be 32-bit or 64-bit, depending on the server.
Boolean	Used to store a true/false value.
Double	Used to store floating point (real) values.
Min/Max Keys	Used internally to compare a value against the lowest or highest BSON elements.
Arrays	Used to store arrays or lists of multiple values in a single key.
Timestamp	Used to record when a document has been modified or added.
Null	Used to store a NULL value. Represents missing or unknown values.
Date	Used to store the current date or time in Unix time format.
Object ID	Used to store the document's unique identifier.
Binary Data	Used to store binary data such as images, binaries, etc.
Code	Used to store Script code in the document.
Regular Expression	Used to store regular expressions for pattern matching.

3.5 MongoDB Query Language

MongoDB Query Language (MQL) is a set of commands and operations used to interact with MongoDB databases. It includes methods for inserting, updating, querying, and managing data within MongoDB collections.

3.5.1 Insert Method: The insert method is used to add one or more documents to a MongoDB collection. It accepts one or more documents as arguments and inserts them into the specified collection.

3.5.2 Save() Method: The save() method is used to save a document to a MongoDB collection. If the document already exists, it updates it; otherwise, it inserts a new document.

3.5.3 Adding a New Field to an Existing Document – Update Method: The update method is used to modify existing documents in a collection. It can be used to add new fields to an existing document by specifying the $set operator.

3.5.4 Removing an Existing Field from an Existing Document – Remove Method: The remove method is used to delete fields from existing documents in a collection. It can be used to remove specific fields from documents using the $unset operator.

3.5.5 Finding Documents based on Search Criteria – Find Method: The find method is used to query documents in a collection based on specified criteria. It returns documents that match the specified query conditions.

3.5.6 Dealing with NULL Values: MongoDB allows fields in documents to have null values. Queries can be constructed to search for documents where specific fields are null or not null.

3.5.7 Count, Limit, Sort, and Skip: These methods provide additional functionality for querying documents. Count is used to count the number of documents that match a query. Limit restricts the number of documents returned by a query. Sort is used to sort the results of a query based on specified criteria. Skip is used to skip a specified number of documents in the result set.

3.5.8 Arrays: MongoDB supports arrays as field values in documents. Arrays can be queried, updated, and manipulated using various array operators.

3.5.9 Aggregate Function: The aggregate function is used to perform aggregation operations on documents in a collection. It allows for complex data analysis, grouping, and computation of aggregate values.

3.5.10 MapReduce Function: The MapReduce function is a data processing model used in MongoDB for performing complex computations on large datasets. It involves two main stages: map and reduce, and can be used for tasks such as data aggregation and analysis.

3.5.11 JavaScript Programming: MongoDB allows users to execute JavaScript code for various database operations, including querying, updating, and data manipulation.

3.5.12 Cursors in MongoDB: Cursors are used to iterate over query results in MongoDB. They allow for efficient retrieval of large result sets by fetching documents in batches.**Table : CRUD Methods**

Method	Description	Example
Insert Method	Inserts a new document into a collection.	db.Students.insert({StudRollNo: 'S101', StudName: 'Simon David', Grade: 'VII'})
Save() Method	Inserts or updates a document (if _id exists).	db.Students.save({_id: 1, StudRollNo: 'S101', StudName: 'Simon David', Grade: 'VII'})
Adding a New Field to an Existing Document – Update Method	Adds a new field to an existing document.	db.Students.update({StudRollNo: 'S101'}, {$set: {Hobbies: 'Reading'}})
Removing an Existing Field from an Existing Document – Remove Method	Removes an existing field from a document.	db.Students.update({StudRollNo: 'S101'}, {$unset: {Hobbies: ''}})
Finding Documents based	Finds documents based on search	db.Students.find({StudRollNo: 'S101'})

on Search Criteria – Find Method	criteria.	
Dealing with NULL Values	Handles documents with NULL values.	db.Students.find({Grade: null})
Count, Limit, Sort, and Skip	Counts, limits, sorts, and skips documents in a result set.	db.Students.find().count(), db.Students.find().limit(5), db.Students.find().sort({Grade: 1}), db.Students.find().skip(2)
Arrays	Handles arrays within documents.	db.Students.insert({StudRollNo: 'S102', Subjects: ['Math', 'Science']})
Aggregate Function	Performs aggregation operations.	db.Students.aggregate([{$match: {Grade: 'VII'}}, {$group: {_id: '$Grade', count: {$sum: 1}}}])
MapReduce Function	Uses the MapReduce function for data processing.	db.Students.mapReduce(mapFunction, reduceFunction, {out: 'GradeCounts'})
Script Programming	Embeds Script in queries.	db.eval(function() { return db.Students.find().count(); })
Cursors in MongoDB	Uses cursors to iterate over query results.	var cursor = db.Students.find(); while(cursor.hasNext()) { printjson(cursor.next()); }
Indexes	Creates indexes to improve query performance.	db.Students.createIndex({StudRollNo: 1})
MongoImport	Imports data from a JSON or CSV file.	mongoimport --db myDB --collection Students --file students.json

MongoExport	Exports data to a JSON or CSV file.	mongoexport --db myDB --collection Students --out students.json
Automatic Generation of Unique Numbers for the "_id" Field	Generates unique numbers for the _id field.	db.Students.insert({_id: ObjectId(), StudRollNo: 'S103', StudName: 'Jane Doe'})

This table provides a quick reference to various methods and operations in MongoDB with simple examples.

3.5.13 Indexes: Indexes are used to improve query performance by providing efficient access to data in MongoDB collections. They can be created on single fields or compound fields to speed up query execution.

3.5.14 MongoImport: MongoImport is a command-line tool used to import data from external files into MongoDB collections. It supports various input formats such as JSON, CSV, and TSV.

3.5.15 MongoExport: MongoExport is a command-line tool used to export data from MongoDB collections to external files. It allows users to extract data from MongoDB databases in formats such as JSON or CSV for analysis or backup purposes.

These definitions provide an overview of the functionalities and capabilities of each MongoDB Query Language (MQL) method.

CRUD (Create, Read, Update, and Delete) operations are fundamental to interacting with data in MongoDB. Below is a summary of the MongoDB query language methods with simple examples, presented in table format.

CRUD Operations in RDBMS vs MongoDB

Operation	***RDBMS***	***MongoDB***
Insert	Insert into Students (StudRollNo, StudName, Grade, Hobbies,	db.Students.insert({_id: 1, StudRollNo: 'S101', StudName:

	DOJ) Values ('S101', 'Simon David', 'VII', 'Net Surfing', '10-Oct-2012')	'Simon David', Grade: 'VII', Hobbies: 'Net Surfing', DOJ: '10-Oct-2012'})
Select StudRollNo, StudName, Hobbies from Students where StudRollNo = 'S101'	db.Students.find({StudRollNo: 'S101'}, {StudRollNo: 1, StudName: 1, Hobbies: 1, _id: 0})	db.Students.update({StudRollNo: 'S101'}, {$set: {Hobbies: 'Ice Hockey'}})
Update	Update Students set Hobbies = 'Ice Hockey'	db.Students.update({}, {$set: {Hobbies: 'Ice Hockey'}}, {multi: true})
Delete	Delete from Students where StudRollNo = 'S101'	db.Students.remove({StudRollNo: 'S101'})
	Delete from Students	db.Students.remove({})
Select	Select * from Students	db.Students.find()
		db.Students.find().pretty()
	Select * from Students where StudRollNo = 'S101'	db.Students.find({StudRollNo: 'S101'})
	Select StudRollNo, StudName, Hobbies from Students	db.Students.find({}, {StudRollNo: 1, StudName: 1, Hobbies: 1, _id: 0})

Here are examples of each MongoDB Query Language (MQL) method along with explanations:

1. **Insert Method:**

   db.students.insertOne({ name: "Alice", age: 25, grade: "A" });

Explanation: This command inserts a document with fields "name", "age", and "grade" into the "students" collection.

2. **Save() Method:**

   db.students.save({ name: "Bob", age: 30, grade: "B" });

   Explanation: The save() method saves a document into the collection. If the document already exists, it will be updated; otherwise, it will be inserted.

3. **Update Method:**

   db.students.updateOne({ name: "Alice" }, { $set: { city: "New York" } });

   Explanation: This command adds a new field "city" with the value "New York" to the document where the name is "Alice".

4. **Remove Method:**

   db.students.updateOne({ name: "Bob" }, { $unset: { grade: "" } });

   Explanation: This command removes the "grade" field from the document where the name is "Bob".

5. **Find Method:**

   db.students.find({ age: { $gt: 25 } });

   Explanation: This command retrieves documents from the "students" collection where the "age" field is greater than 25.

6. **Dealing with NULL Values:**

   db.students.find({ city: null });

Explanation: This query retrieves documents from the "students" collection where the "city" field is null.

7. **Count, Limit, Sort, and Skip:**

```
db.students.find().count();
db.students.find().limit(5);
db.students.find().sort({ age: 1 });
db.students.find().skip(10);
```

Explanation: These commands respectively count the number of documents, limit the result to 5 documents, sort the result by age in ascending order, and skip the first 10 documents in the result set.

8. **Arrays:**

```
db.students.updateOne({ name: "Alice" }, { $push: { hobbies: "Reading" } });
```

Explanation: This command adds the value "Reading" to the "hobbies" array field in the document where the name is "Alice".

9. **Aggregate Function:**

```
db.students.aggregate([
   { $group: { _id: "$grade", count: { $sum: 1 } } }
]);
```

Explanation: This aggregation pipeline groups documents by the "grade" field and counts the number of documents in each group.

10. **MapReduce Function:**

```
var mapFunction = function() {
   emit(this.grade, 1);
};
```

```
var reduceFunction = function(key, values) {
    return Array.sum(values);
};
db.students.mapReduce(mapFunction, reduceFunction, { out: "grade_counts" });
```

Explanation: This MapReduce operation counts the number of documents per grade by emitting each grade with a value of 1, then reducing the values for each grade to get the total count.

9. **Programming:**

```
// Writing a  function to find students above a certain age
var findStudentsAboveAge = function(age) {
    return db.students.find({ age: { $gt: age } });
}
```

Explanation: This function defines a reusable query to find students above a certain age, specified by the parameter "age".

10. **Cursors in MongoDB:**

```
// Iterating over documents using a cursor
var cursor = db.students.find();
while (cursor.hasNext()) {
    printjson(cursor.next());
}
```

Explanation: This code snippet retrieves all documents from the "students" collection using a cursor, then iterates over each document and prints it to the console.

11. **MongoImport:**

```
// Importing data from a JSON file into the "students" collection
```

```
mongoimport --db mydb --collection students --file students.json
```

Explanation: This command imports data from a JSON file ("students.json") into the "students" collection in the "mydb" database.

12. **MongoExport:**

```
// Exporting data from the "students" collection to a JSON file
mongoexport --db mydb --collection students --out students.json
```

Explanation: This command exports data from the "students" collection in the "mydb" database to a JSON file named "students.json".

These examples illustrate various functionalities and operations available in MongoDB Query Language (MQL), providing flexibility and power in managing MongoDB databases.

Lets sum up

MongoDB is a NoSQL database known for its flexible document-oriented data model, which contrasts with the rigid schema of traditional RDBMS. It employs various unique terminologies; for instance, 'collections' in MongoDB are equivalent to 'tables' in RDBMS, and 'documents' are analogous to 'rows'. MongoDB supports diverse data types, including arrays and embedded documents, providing rich data structuring capabilities. Its powerful query language allows for dynamic querying and indexing, supporting a wide range of operations to efficiently retrieve and manipulate data.

3.6. MapReduce Programming

3.6.1 Introduction

In MapReduce Programming, Jobs (Applications) are split into a set of map tasks and reduce tasks. Then these tasks are executed in a distributed fashion on Hadoop cluster.

Each task processes small subset of data that has been assigned to it. This way, Hadoop distributes the load across the cluster.

MapReduce job takes a set of files that is stored in HDFS (Hadoop Distributed File System) as input. Map task takes care of loading, parsing, transforming, and filtering.

The responsibility of reduce task is grouping and aggregating data that is produced by map tasks to generate final output. Each map task is broken into the following phases:

1. RecordReader.
2. Mapper.
3. Combiner.
4. Partitioner.

The output produced by map task is known as intermediate keys and values. These intermediate keys and values are sent to reducer. The reduce tasks are broken into the following phases:

1. Shuffle.
2. Sort.
3. Reducer.
4. Output Format.

Hadoop assigns map tasks to the DataNode where the actual data to be processed resides. This way, Hadoop ensures data locality. Data locality means that data is not moved over network; only computational code moved to process data which saves network bandwidth.

3.6.2 Mapper

A mapper maps the input key−value pairs into a set of intermediate key–value pairs. Maps are individual tasks that have the responsibility of transforming input records into intermediate key–value pairs.

1. **RecordReader:** RecordReader converts a byte-oriented view of the input (as generated by the InputSplit) into a record-oriented view and presents it to the

Mapper tasks. It presents the tasks with keys and values. Generally, the key is the positional information and value is a chunk of data that constitutes the record.

2. **Map:** Map function works on the key–value pair produced by RecordReader and generates zero or more intermediate key–value pairs. The MapReduce decides the key–value pair based on the context.
3. **Combiner:** It is an optional function but provides high performance in terms of network bandwidth and disk space. It takes intermediate key–value pair provided by mapper and applies user-specific aggregate function to only that mapper. It is also known as local reducer.
4. **Partitioner:** The partitioner takes the intermediate key–value pairs produced by the mapper, splits them into shard, and sends the shard to the particular reducer as per the user-specific code. Usually, the key with same values goes to the same reducer. The partitioned data of each map task is written to the local disk of that machine and pulled by the respective reducer.

3.6.3 Reducer

The primary chore of the Reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values. The Reducer has three primary phases: Shuffle and Sort, Reduce, and Output Format.

1. **Shuffle and Sort:** This phase takes the output of all the partitioners and downloads them into the local machine where the reducer is running. Then these individual data pipes are sorted by keys which produce larger data list. The main purpose of this sort is grouping similar words so that their values can be easily iterated over by the reduce task.
2. **Reduce:** The reducer takes the grouped data produced by the shuffle and sort phase, applies reduce function, and processes one group at a time. The reduce function iterates all the values associated with that key. Reducer function provides various operations such as aggregation, filtering, and combining data. Once it is done, the output (zero or more key–value pairs) of reducer is sent to the output format.
3. **Output Format:** The output format separates key–value pair with tab (default) and writes it out to a file using record writer.

Figure 3.5 describes the chores of Mapper, Combiner, Partitioner, and Reducer for the word count problem. The Word Count problem has been discussed under “Combiner” and “Partitioner”.

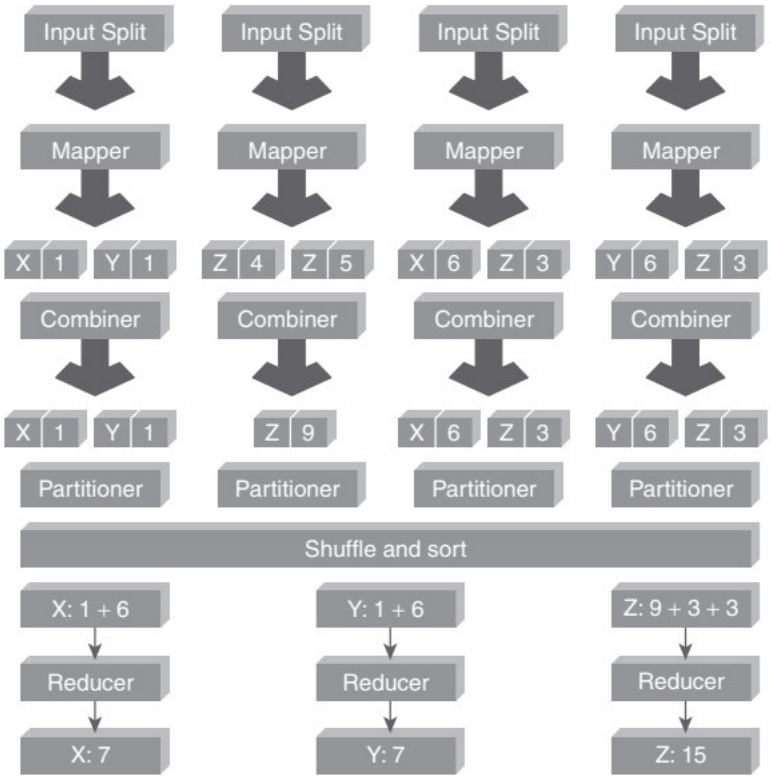


Figure 3.5 Chores of Mapper, Combiner, Partitioner, and Reducer.

3.6.4 Combiner

It is an optimization technique for MapReduce Job. Generally, the reducer class is set to be the combiner class. The difference between combiner class and reducer class is as follows:

1. Output generated by combiner is intermediate data and it is passed to the reducer.
2. Output of the reducer is passed to the output file on disk.

The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?

Input Data: What is the input that has been given to us to act upon?

Act: The actual statement/command to accomplish the task at hand.

Output: The result/output as a consequence of executing the statement.

Objective: Write a MapReduce program to count the occurrence of similar words in a file. Use combiner for optimization. Note: Refer Chapter 5 – Hadoop for Mapper Class and Reduce Class and Driver Program.

Input Data:

Welcome to Hadoop Session

Introduction to Hadoop

Introducing Hive

Hive Session

Pig Session

Act: In the driver program, set the combiner class as shown below.

job.setCombinerClass(WordCounterRed.class);

// Input and Output Path

FileInputFormat.addInputPath(job, new Path("/mapreducedemos/lines.txt"));
FileOutputFormat.setOutputPath(job, new
Path("/mapreducedemos/output/wordcount/"));

hadoop jar <<jar name>> <<driver class>> <<input path>> <<output path>>

Here driver class name, input path, and output path are optional arguments.

Output:

```
[root@volgalnx010 mapreducedemos]# hadoop jar wordcount.jar
```

Contents of directory /mapreducedemos

Goto : /mapreducedemos go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
lines.txt	file	91 B	3	128 MB	2015-03-01 21:05	rw-r--r--	root	supergroup
output	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Contents of directory /mapreducedemos/output

Goto : /mapreducedemos/output go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
wordcount	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

The reducer output will be stored in part-r-00000 file by default.

Contents of directory /mapreducedemos/output/wordcount

Goto : /mapreducedemos/output/wo go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup
part-r-00000	file	76 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

File: /mapreducedemos/output/wordcount/part-r-00000

Goto : /mapreducedemos/output/wo go

Go back to dir listing
Advanced view/download options

```
Hadoop  2
Hive    2
Introducing     1
Introduction    1
Pig     1
Session 3
Welcome 1
to      2
```

3.7 Compression

In MapReduce programming, you can compress the MapReduce output file. Compression provides two benefits as follows:

1. Reduces the space to store files.
2. Speeds up data transfer across the network.

You can specify compression format in the Driver Program as shown below:

```
conf.setBoolean("mapred.output.compress", true);
conf.setClass("mapred.output.compression.codec",
GzipCodec.class,CompressionCodec.class);
```

Here, codec is the implementation of a compression and decompression algorithm. GzipCodec is the compression algorithm for gzip. This compresses the output file.

Lets sum up

MapReduce is a programming model designed for processing and generating large datasets in a distributed computing environment. It divides tasks into two primary components: the Mapper, which processes input data and produces intermediate key-

value pairs, and the Reducer, which merges these intermediate outputs to produce the final result. The Combiner acts as a mini-reducer to optimize and reduce data transfer between Mappers and Reducers. The Partitioner determines how the intermediate key-value pairs are distributed across the Reducers, ensuring balanced and efficient processing. This model is fundamental to handling big data in parallel across numerous nodes in a cluster.

UNIT III: SUMMARY

- **Terms used in RDBMS and MongoDB**:
  - MongoDB uses collections (analogous to tables in RDBMS) to store documents (similar to rows/records in RDBMS). Each document is a JSON-like structure composed of field-value pairs (similar to columns in RDBMS). Indexes can be created to optimize query performance.
- **Data Types**:
  - MongoDB supports a variety of data types including string, integer, double, boolean, date, array, object, and others. This flexibility allows for schema-less or schema-flexible designs, accommodating changes in data structure over time.
- **Query Language (MQL)**:
  - MongoDB Query Language (MQL) supports CRUD operations (Create, Read, Update, Delete) for document management. It also includes an aggregation framework for performing complex operations like grouping, sorting, and transforming data within the database.
- **Replication and High Availability**:
  - MongoDB utilizes replica sets to provide data redundancy and fault tolerance. It automatically elects a primary node and maintains multiple secondary nodes that replicate data asynchronously from the primary.
- **Sharding**:
  - MongoDB sharding distributes data across multiple servers (or shards) to handle large datasets and high throughput. It partitions data based on a shard key and distributes queries across shards for parallel execution.
- **Mapper**:
  - In the MapReduce paradigm, the Mapper function processes input data in key-value pairs and generates intermediate key-value pairs based on a specified logic or computation.
- **Reducer**:

- Reducer functions receive intermediate key-value pairs from multiple mappers, aggregate or process them according to a defined logic, and produce final output key-value pairs.

- **Combiner**:
  - Optional in MapReduce, a Combiner function locally aggregates intermediate data outputted by mappers before transmitting it to reducers. This reduces the amount of data transferred over the network, optimizing performance.
- **Partitioner**:
  - Determines which reducer will receive specific keys' intermediate data, ensuring that data with the same key ends up on the same reducer. This optimizes data processing and reduces network traffic.
- **Searching and Sorting**:
  - MapReduce frameworks like Hadoop MapReduce distribute tasks across nodes in a cluster, enabling efficient searching and sorting of large datasets by leveraging parallel processing and distributed computing.
- **Compression**:
  - MapReduce supports data compression techniques to reduce storage requirements and improve data transfer efficiency between nodes in the cluster. This is crucial for optimizing performance when dealing with large volumes of data.

Glossary

- **Collection**: A MongoDB concept analogous to a table in relational databases, containing documents.
- **Document**: A record in MongoDB, similar to a row in relational databases, representing data as JSON-like structures.
- **Field**: A key-value pair within a document, similar to a column in relational databases.
- **Index**: A feature in MongoDB for optimizing query performance by facilitating fast data retrieval.
- **MQL (MongoDB Query Language)**: Language used to query and manipulate data in MongoDB, supporting CRUD operations and aggregation.
- **Mapper**: Function in the MapReduce paradigm that processes input data and emits intermediate key-value pairs.
- **Reducer**: Function that processes intermediate key-value pairs generated by mappers and produces final output.
- **Combiner**: Optional function that performs local reduction of data before sending it to reducers, optimizing network bandwidth.
- **Partitioner**: Determines which reducer instance receives a given key's intermediate data, ensuring efficient data distribution.
- **Searching and Sorting**: MapReduce framework capability to search and sort large datasets by distributing tasks across nodes in a cluster.
- **Compression**: Techniques used in MapReduce to reduce storage requirements and enhance data transfer efficiency.

Checkup Your Progress

EXERCISE 1: Fill in each gap with the right word from the list

1. MongoDB uses collections and documents instead of tables and rows found in traditional _____ (RDBMS, NoSQL, SQL).
2. MongoDB supports various data types including string, integer, double, boolean, date, array, and _____ (object, function, structure).
3. MongoDB Query Language (MQL) allows for performing CRUD operations and advanced _____ (queries, processing, administration).
4. In MapReduce programming, the _____ function processes input data and emits intermediate key-value pairs (Mapper, Reducer, Combiner).
5. The _____ function in MapReduce takes intermediate data from the Mapper and produces the final output (Mapper, Reducer, Combiner).

EXERCISE 2: Read the questions below and circle if the answer is True

1. MongoDB uses collections and documents, similar to tables and rows in relational databases – True / False

2. MongoDB supports only a limited set of data types compared to traditional relational databases - True / False

3. MongoDB Query Language (MQL) supports CRUD operations but does not offer aggregation capabilities - True / False

4. In MapReduce programming, the Reducer function processes input data and emits intermediate key-value pairs - True / False

5. MapReduce can be used for tasks like searching, sorting, and aggregating data across distributed systems - True / False

EXERCISE 3: Choose the correct answer

1. Which of the following is true about MongoDB compared to traditional relational databases?

   a) MongoDB uses tables and rows for data storage.

b) MongoDB supports a fixed schema for data.

c) MongoDB uses collections and documents for data storage.

2. What type of data does MongoDB support?

a) Only structured data with a fixed schema.

b) Only unstructured data with a flexible schema.

c) Various data types including string, integer, array, and more.

3. Which language is used for querying data in MongoDB?

a) SQL

b) MQL (MongoDB Query Language)

c) NoSQL

4. In MapReduce programming, what is the role of the Reducer function?

a) Processes input data and emits intermediate key-value pairs.

b) Filters data based on a condition.

c) Aggregates data and produces the final output.

5. What tasks can MapReduce be used for in distributed computing?

a) Real-time data processing only.

b) Searching, sorting, and aggregating large datasets.

c) Database transactions.

EXERCISE 4: Match the following

1.	MongoDB Query Language (MQL)	-	A. Aggregates data and produces the final output in MapReduce
2.	Reducer	-	B. Determines which Reducer instance will receive a given key's intermediate data
3.	Combiner	-	C. Performs a local reduction of data before sending it to the Reducer
4.	Partitioner	-	D. Used to query and manipulate data in MongoDB

5. Searching and Sorting - Tasks that can be performed using MapReduce in distributed systems

EXERCISE 5: Self-Assessment Questions

1. What is the equivalent of a "table" in RDBMS in MongoDB?
2. Name two data types supported by MongoDB.
3. Write a basic MongoDB query to find all documents in a collection where the "age" field is greater than 25.
4. What is the primary function of the Mapper in a MapReduce job?
5. What role does the Reducer play in the MapReduce framework?

Answer for Checkup Your Progress

EXERCISE 1:

1.SQL, 2. Object 3. Queries 4. Mapper 5. Reducer

EXERCISE 2:

1. False 2. False 3. False 4. False 5. True

EXERCISE 3:

1. c) 2. c) 3. b) 4. c) 5. b)

EXERCISE 4:

1.D, 2.A, 3.C, 4.B, 5.E.

Open Source E-content Links:

1. https://www.mongodb.com/docs/manual/
2. https://www.mongodb.com/docs/manual/core/map-reduce/

3. https://www.mongodb.com/docs/manual/tutorial/query-documents/
4. https://www.mongodb.com/docs/manual/reference/bson-types/

References:

1. MongoDB Documentation. (2023). Retrieved June 28, 2024, from https://docs.mongodb.com/manual/
2. Chodorow, K. (2013). MongoDB: The Definitive Guide (2nd ed.). Sebastopol, CA: O'Reilly Media.
3. MongoDB Data Types. (2023). Retrieved June 28, 2024, from https://docs.mongodb.com/manual/reference/bson-types/
4. MapReduce Concepts in MongoDB. (2023). Retrieved June 28, 2024, from https://docs.mongodb.com/manual/core/map-reduce/
5. Apache Hadoop MapReduce Tutorial. (2023). Retrieved June 28, 2024, from https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Unit IV: HIVE

UNIT IV OBJECTIVE

The objective of this unit is to provide a comprehensive understanding of Apache Hive, including its architecture, data types, and file formats. Learners will explore the Hive Query Language (HQL) for data manipulation and analysis, delve into advanced features such as partitions and bucketing, and learn how to create and manage views, sub-queries, joins, aggregations, and the GROUP BY and HAVING clauses. The unit will cover the RCFile format and its implementation, guide the creation of Hive User Defined Functions (UDFs), and explain the concepts of serialization and deserialization for handling various data formats. By the end of this unit, learners will be equipped with the knowledge to efficiently utilize Hive for data warehousing and analysis in a Hadoop ecosystem.

Unit Summary

1.Introduction to Hive

- Architecture of Hive
- Data Types and File Formats in Hive
- Hive Query Language Statements
- Partitions and Bucketing in Hive
- Views and Sub-Queries in Hive
- Joins, Aggregations, Group by, and Having in Hive
- RCFile Implementation
- User Defined Functions (UDFs) in Hive
- Serialization and Deserialization

4.1 What is Hive?

Hive is a Data Warehousing tool that sits on top of Hadoop. Refer Figure 8.1. Hive is used to process structured data in Hadoop. The three main tasks performed by Apache Hive are:

1. Summarization
2. Querying
3. Analysis

Facebook initially created Hive component to manage their ever-growing volumes of log data. Later Apache software foundation developed it as open-source and it came to be known as Apache Hive. Hive makes use of the following:

1. HDFS for Storage.
2. MapReduce for execution.
3. Stores metadata/schemas in an RDBMS.

Hive provides HQL (Hive Query Language) or HiveQL which is similar to SQL. Hive compiles queries into MapReduce jobs and then runs the job in the Hadoop Cluster.

Hive provides extensive data type functions and formats for data summarization and analysis. **Note:**

1. Hive is not RDBMS.
2. It is not designed to support OLTP (Online Transaction Processing).
3. It is not designed for real-time queries.
4. It is not designed to support row-level updates.

4.1.1 History of Hive and Recent Releases of Hive

The history of Hive and recent releases of Hive are illustrated pictorially in Figures 4.1 and 4.2, respectively.

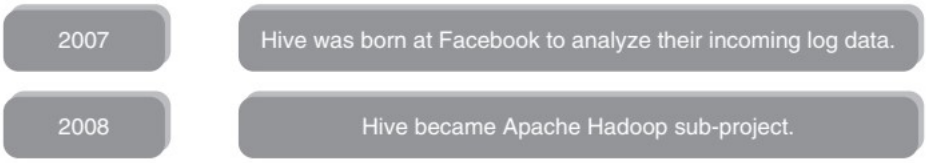


Figure 4.1 History of Hive

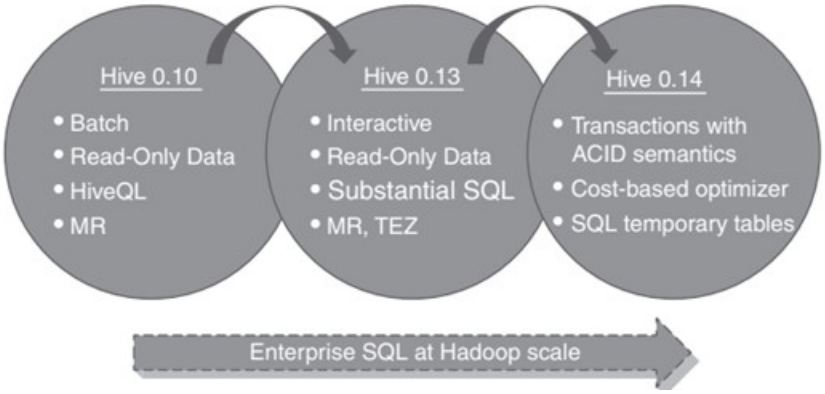


Figure 4.2 Recent releases of Hive

4.1.2 Hive Features

1. It is similar to SQL.
2. HQL is easy to code.
3. Hive supports rich data types such as structs, lists and maps.
4. Hive supports filters, group-by and order-by clauses.
5. Custom Types, Custom Functions can be defined.

4.1.3 Hive Integration and Work Flow

Figure 4.3 illustrates the workflow for log file analysis. The process begins with the storage of hourly log data directly into the Hadoop Distributed File System (HDFS). Once stored, the log data undergoes a cleansing process to remove any irrelevant or erroneous information. After cleansing, the log data is compressed to optimize storage and processing efficiency. Subsequently, Hive tables are created to facilitate querying and analyzing the log data. Typically, multiple Hive tables (e.g., Hive Table 1 and Hive Table 2) are generated to organize the cleansed and compressed data for different analytical purposes. This workflow streamlines the handling, processing, and querying of log files, enabling efficient data analysis and insights extraction.

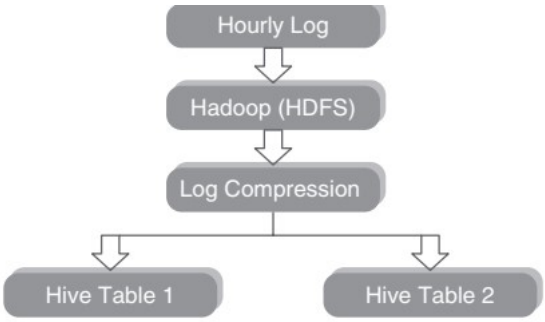


Figure 4.3 Flow of log analysis file

4.1.4 Hive Data Units

1. **Databases:** The namespace for tables.
2. **Tables:** Set of records that have similar schema.
3. **Partitions:** Logical separations of data based on classification of given information as per specific attributes. Once hive has partitioned the data based on a specified key, it starts to assemble the records into specific folders as and when the records are inserted.
4. **Buckets (or Clusters):** Similar to partitions but uses hash function to segregate data and determines the cluster or bucket into which the record should be placed.

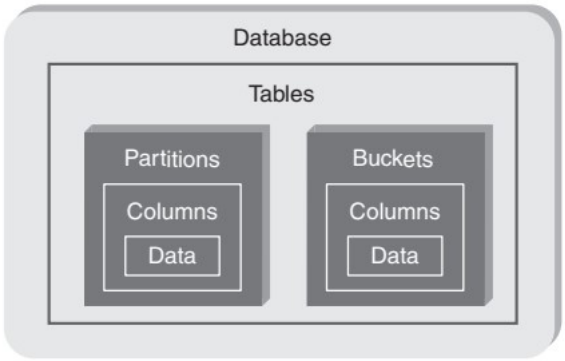


Figure 4.4 Data units as arranged in a Hive

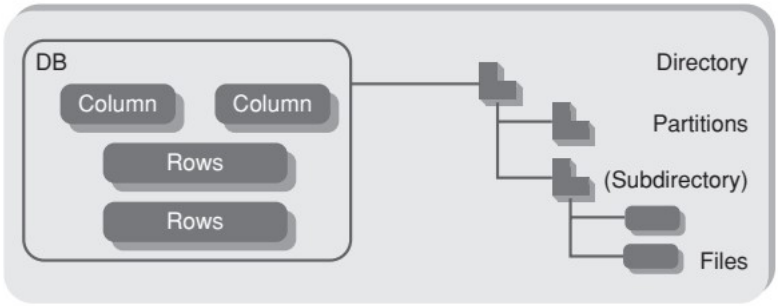


Figure 4.5 Semblance of Hive Sturcute with database

When to Use Partitioning/Bucketing?

Bucketing works well when the field has high cardinality (cardinality is the number of values a column or field can have) and data is evenly distributed among buckets. Partitioning works best when the cardinality of the partitioning field is not too high.

Partitioning can be done on multiple fields with an order (Year/Month/ Day) whereas bucketing can be done on only one field.

Figure 4.4 shows how these data units are arranged in a Hive Cluster. Figure 4.5 describes the semblance of Hive structure with database.

A database contains several tables. Each table is constituted of rows and columns. In Hive, tables are stored as a folder and partition tables are stored as a sub-directory. Bucketed tables are stored as a file.

4.2 Hive Architecture

Hive Architecture is depicted in Figure 4.6. The various parts are as follows:

1. **Hive Command-Line Interface (Hive CLI):** The most commonly used interface to interact with Hive.
2. **Hive Web Interface:** It is a simple Graphic User Interface to interact with Hive and to execute query.
3. **Hive Server:** This is an optional server. This can be used to submit Hive Jobs from a remote client.

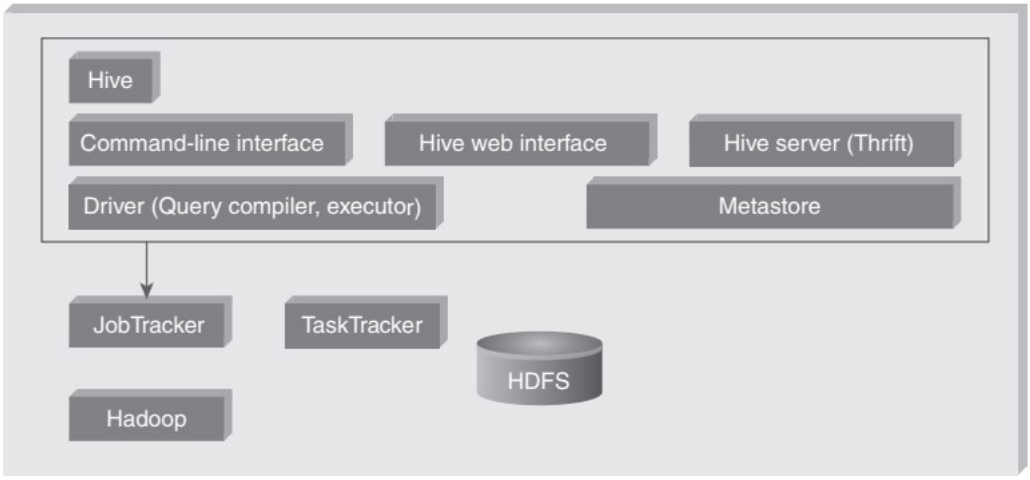


Figure 4.6 Hive Architecture

4. **JDBC/ODBC:** Jobs can be submitted from a JDBC Client. One can write a code to connect to Hive and submit jobs on it.
5. **Driver:** Hive queries are sent to the driver for compilation, optimization and execution.

6. **Metastore:** Hive table definitions and mappings to the data are stored in a Metastore. A Metastore consists of the following:
    - **Metastore service:** Offers interface to the Hive.
    - **Database:** Stores data definitions, mappings to the data and others.

    The metadata which is stored in the metastore includes IDs of Database, IDs of Tables, IDs of Indexes, etc., the time of creation of a Table, the Input Format used for a Table, the Output Format used for a Table, etc. The metastore is updated whenever a table is created or deleted from Hive. There are three kinds of metastore.
    1. **Embedded Metastore:** This metastore is mainly used for unit tests. Here, only one process is allowed to connect to the metastore at a time. This is the default metastore for Hive. It is Apache Derby Database. In this metastore, both the database and the metastore service run embedded in the main Hive Server process. Figure 4.7 shows an Embedded Metastore.
    2. **Local Metastore:** Metadata can be stored in any RDBMS component like My. Local metastore allows multiple connections at a time. In this mode, the Hive metastore service runs in the main Hive Server process, but the metastore database runs in a separate process, and can be on a separate host. Figure 4.8 shows a Local Metastore.
    3. **Remote Metastore:** In this, the Hive driver and the metastore interface run on different JVMs (which can run on different machines as well) as in Figure 4.9 This way the database can be fire-walled from the Hive user and also database credentials are completely isolated from the users of Hive.

Figure 4.7 Embedded Metastore

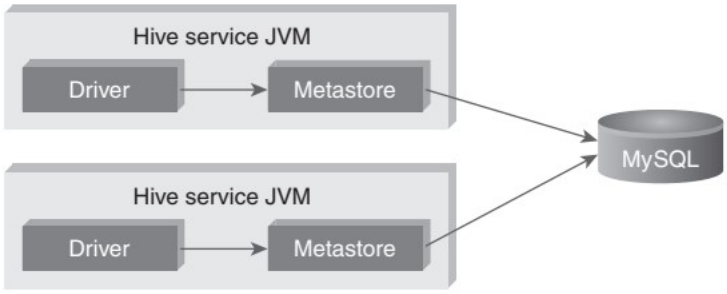


Figure 4.8 Local Metastore

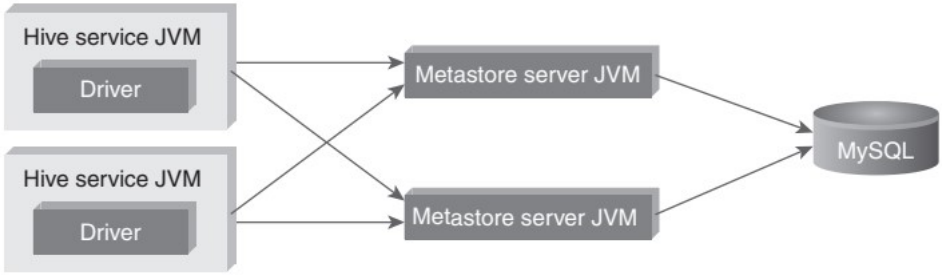


Figure 4.9 Remote Metastore

4.3 Hive Data Types

In Apache Hive, data types define the nature of the data that can be stored in Hive tables. Hive supports a wide range of data types, which can be categorized into primitive types and complex types.

4.3.1 Primitive Data Types

Table 4.1 Numeric Data Types

Data Type	*Description*
TINYINT	1-byte signed integer
SMALLINT	2-byte signed integer
INT	4-byte signed integer
BIGINT	8-byte signed integer
FLOAT	4-byte single-precision floating-point
DOUBLE	8-byte double-precision floating-point number

Table 4.2 String Types

Data Type	*Description*
STRING	String type
VARCHAR	Variable-length string type (only available starting with Hive 0.12.0)
CHAR	Fixed-length string type (only available starting with Hive 0.13.0)
	Strings can be expressed in either single quotes (‘) or double quotes (“)

Table 4.3 Miscellaneous Types

Data Type	*Description*
BOOLEAN	Boolean type (true/false)
BINARY	Binary data type (only available starting with Hive)

Tables 4.1, 4.2 and 4.3 show numeric data types, string types and miscellaneous type. Various collection data types are shown in table 4.4.

4.3.2 Collection Data Types

Table 4.4 Collection Data Types

Collection Type	***Description***	***Example***
STRUCT	Similar to ‘C’ struct. Fields are accessed using dot notation.	struct('John', 'Doe')
MAP	A collection of key–value pairs. Fields are accessed using [] notation.	map('first', 'John', 'last', 'Doe')
ARRAY	Ordered sequence of same types. Fields are accessed using array index.	array('John', 'Doe')

4.4 Hive File Format

The file formats in Hive specify how records are encoded in a file.

4.4.1 Text File

- **Description**: The default file format. Each record is a line in the file. Different control characters are used as delimiters:
  - ^A (octal 001): Separates all fields.
  - ^B (octal 002): Separates elements in the array or struct.
  - ^C (octal 003): Separates key–value pairs.
  - \n: New line character.
- **Supported Text Files**: CSV, TSV, JSON, XML

4.4.2 Sequential File

- **Description**: Flat files that store binary key–value pairs. Includes compression support, reducing CPU and I/O requirements.

4.4.3 RCFile (Record Columnar File)

- RCFile stores data in a column-oriented manner, making aggregation operations more efficient. Below are examples illustrating how a table is partitioned and serialized in RCFile format. Table 4.5 shows with four columns. Table 4.6 shows with two columns and table 4.7 shows the RCF file format.

Table 4.5 A table with four columns

Row Group 1			
C1	**C2**	**C3**	**C4**
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54

Table 4.6 Table with two row groups

Row Group 1	Row Group 2
11, 21, 31	41, 51
12, 22, 32	42, 52
13, 23, 33	43, 53
14, 24, 34	44, 54

Table 4.7 Table in RCFile Format

Row Group 1			
C1	**C2**	**C3**	**C4**
11	12	13	14
21	22	23	24
31	32	33	34

- In RCFile format, the table data is partitioned both horizontally into row groups and vertically within each row group. This structure enhances the efficiency of data aggregation operations.

- **Partitioning**: Instead of only partitioning the table horizontally like row-oriented DBMS (row-store), RCFile partitions the table first horizontally and then vertically to serialize the data.
    - First, the table is partitioned into multiple row groups horizontally. For example, Table 4.6 is partitioned into two row groups, each containing three rows.
    - Next, within each row group, RCFile partitions the data vertically like column-store.

4.5 Hive Query Language (HQL)

Hive Query Language (HQL) provides -like operations for data management and manipulation. Below are some key tasks that can be easily performed using HQL:

1. Create and manage tables and partitions.
2. Support various relational, arithmetic, and logical operators.
3. Evaluate functions.
4. Download the contents of a table to a local directory or results of queries to HDFS directory.

4.5.1 DDL (Data Definition Language) Statements

These statements are used to create and modify tables and other objects in the database. The DDL commands include:

1. Create/Drop/Alter Database
2. Create/Drop/Truncate Table

3. Alter Table/Partition/Column
4. Create/Drop/Alter View
5. Create/Drop/Alter Index
6. Show
7. Describe

4.5.2 DML (Data Manipulation Language) Statements

These statements are used to retrieve, store, modify, delete, and update data in the database. The DML commands include:

1. Loading files into table
2. Inserting data into Hive Tables from queries

Note: Hive 0.14 supports update, delete, and transaction operations.

4.5.3 Hive DDL and DML Examples

Describe Database

The DESCRIBE DATABASE command provides metadata about a specified database. This includes information about the database location, owner, and parameters.

Syntax:

DESCRIBE DATABASE [EXTENDED] database_name;

Example:

DESCRIBE DATABASE my_database;

Output:

Database Name: my_database
Description: Database for storing student records

Location: hdfs://path/to/my_database
Owner: user
Parameters: {...}

Show Databases

The SHOW DATABASES command lists all the databases available in the Hive metastore.

Syntax:

SHOW DATABASES;

Example:

SHOW DATABASES;

Output:

arduino
default
my_database
sales_database
employee_database

Table 4.8 Data Definition Language (DDL)

DDL Command	*Description*
CREATE DATABASE mydatabase;	Creates a new database named "mydatabase".
SHOW DATABASES;	Displays a list of all databases.
USE mydatabase;	Sets the current database context to "mydatabase".
CREATE TABLE students	Creates a table named "students" with two

(student_id INT, name STRING);	columns: "student_id" of type INT and "name" of type STRING.
SHOW TABLES;	Displays a list of all tables in the current database.
ALTER TABLE students ADD COLUMN age INT;	Adds a new column named "age" of type INT to the "students" table.
DESCRIBE students;	Shows the structure of the "students" table.
CREATE VIEW student_view AS SELECT * FROM students;	Creates a view named "student_view" based on the "students" table.
DROP TABLE students;	Deletes the "students" table.
CREATE EXTERNAL TABLE ext_students (student_id INT, name STRING) LOCATION '/path/to/data';	Creates an external table named "ext_students" with two columns: "student_id" of type INT and "name" of type STRING. The table is stored externally at the specified location.

Table 4.9 Data Manipulation Language

DML Command	*Description*
LOAD DATA INPATH '/path/to/data' INTO TABLE students;	Loads data from a file into the "students" table.
INSERT INTO TABLE students VALUES (1, 'John');	Inserts a new row into the "students" table.
INSERT INTO TABLE students SELECT * FROM other_table;	Inserts data from another table into the "students" table.
SELECT * FROM students;	Retrieves all rows from the "students" table.
SELECT name, age FROM students WHERE age > 18;	Retrieves the name and age of students older than 18.
UPDATE students SET age = 25 WHERE name = 'John';	Updates the age of the student named "John".

DELETE FROM students WHERE age < 18;	Deletes records of students younger than 18.

These examples illustrate various DDL (Data Definition Language) and DML (Data Manipulation Language) commands in Hive, including the creation of external tables is shown in tables 4.8 and 4.9.

4.5.4 Tables in Hive

Hive provides two kinds of tables:

1. Internal or Managed Table
2. External Table

4.5.4.1 Managed Table

1. **Storage**: Hive stores the managed tables under the warehouse folder in Hive.
2. **Lifecycle Management**: The complete lifecycle of the table and data is managed by Hive.
3. **Data and Metadata Deletion**: When an internal table is dropped, both the data and the metadata are dropped.

When you create a table in Hive, by default, it is an internal or managed table. If one needs to create an external table, the keyword "EXTERNAL" must be used.

Table Type	*Description*
Internal Table (Managed Table)	1. Stored under the warehouse folder in Hive.2. Hive manages the complete lifecycle of the table and data.3. Dropping the table also drops the data and metadata.
External Table	1. The table schema is managed by Hive, but the data is stored externally.2. Dropping the table only drops the metadata, not the data. The data remains in the specified location.

Example Commands

Creating an Internal Table (Managed Table)

```
CREATE TABLE students (
    student_id INT,
    name STRING,
    age INT,
    grade STRING
);
```

Creating an External Table

```
CREATE EXTERNAL TABLE ext_students (
    student_id INT,
    name STRING,
    age INT,
    grade STRING
)
LOCATION '/path/to/external/data';
```

4.5.5 Partitions

Partitions allow Hive to divide a table into parts based on the values of a particular column, improving query performance by scanning only relevant partitions.

```
CREATE TABLE students (
    student_id INT,
    name STRING,
    age INT
)
PARTITIONED BY (grade STRING);
```

Loading Data into a Partitioned Table

LOAD DATA INPATH '/path/to/data' INTO TABLE students PARTITION (grade='A');

Partition is of two types:

***4.5.5.1 STATIC PARTITION*:** It is upon the user to mention the partition (the segregation unit) where the data from the file is to be loaded.
Static partitions comprise columns whose values are known at compile time.

Example:

Objective: To create static partition based on “gpa” column. Act: CREATE TABLE IF NOT EXISTS STATIC_PART_STUDENT (rollno INT, name STRING) PARTITIONED BY (gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'; Outcome: Objective: Load data into partition table from table. Act: INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION

Objective: Load data into partition table from table. Act: INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION (gpa =4.0) SELECT rollno, name from EXT_STUDENT where gpa=4.0; Outcome:

***4.5.5.2 DYNAMIC PARTITION*:** The user is required to simply state the column, basis which the partitioning will take place. Hive will then create partitions basis the unique values in the column on which partition is to be carried out.

Dynamic partition has columns whose values are known only at Execution Time.

Example:

Objective: To create dynamic partition on column date. Act: CREATE TABLE IF NOT EXISTS DYNAMIC_PART_STUDENT(rollno INT,name STRING) PARTITIONED BY (gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

4.5.6 Bucketing

Bucketing further subdivides data into manageable parts within a partition, enhancing the performance of specific types of queries.

```
CREATE TABLE bucketed_students (
    student_id INT,
    name STRING,
    age INT
)
CLUSTERED BY (student_id) INTO 4 BUCKETS;
```

Loading Data into a Bucketed Table

```
INSERT INTO TABLE bucketed_students
SELECT * FROM students;
```

4.5.7 Views

Views are virtual tables representing the result of a stored query. They help in simplifying complex queries and data abstraction.

```
CREATE VIEW student_view AS
SELECT student_id, name, grade
FROM students
WHERE age > 18;
```

Querying the View

```
SELECT * FROM student_view;
```

4.5.8 Sub-Query

Sub-queries are nested queries within a main query. They allow for complex data retrieval operations and are useful for breaking down complex logic into manageable parts.

SELECT name FROM students WHERE age IN (SELECT age FROM students WHERE grade = 'A');

4.6 Tables

4.6.1 Creating an Internal Table (Managed Table)

```
CREATE TABLE students (
    student_id INT,
    name STRING,
    age INT,
    grade STRING
);
```

4.6.2 Creating an External Table

```
CREATE EXTERNAL TABLE ext_students (
    student_id INT,
    name STRING,
    age INT,
    grade STRING
)
LOCATION '/path/to/external/data';
```

4.6.2.1 External or Self-Managed Table

1. When the table is dropped, it retains the data in the underlying location.
2. External keyword is used to create an external table.
3. Location needs to be specified to store the dataset in that particular location.

4.7 Partitions

Creating a Partitioned Table

```
CREATE TABLE students (
    student_id INT,
    name STRING,
    age INT
)
PARTITIONED BY (grade STRING);
```

Loading Data into a Partitioned Table

```
LOAD DATA INPATH '/path/to/data' INTO TABLE students PARTITION (grade='A');
```

4.8 Joins

Joins are used to combine rows from two or more tables based on related columns. Hive supports various types of joins including INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.

Inner Join

```
SELECT s.student_id, s.name, g.grade
FROM students s
JOIN grades g ON s.student_id = g.student_id;
```

Left Join

```
SELECT s.student_id, s.name, g.grade
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id;
```

Right Join

```
SELECT s.student_id, s.name, g.grade
FROM students s
RIGHT JOIN grades g ON s.student_id = g.student_id;
```

Full Join

```
SELECT s.student_id, s.name, g.grade
FROM students s
FULL JOIN grades g ON s.student_id = g.student_id;
```

4.9 Aggregation

Using Aggregation Functions

COUNT

```
SELECT grade, COUNT(*) as student_count
FROM students
GROUP BY grade;
```

Sum

```
SELECT grade, SUM(age) as total_age
FROM students
GROUP BY grade;
```

Average

```
SELECT grade, AVG(age) as average_age
FROM students
GROUP BY grade;
```

4.10 Group By and Having

- **Group By**: Groups rows that have the same values into summary rows, like "find the number of customers in each country".
- **Having**: Used to filter records that work on summarized Group By results.

  SELECT grade, COUNT(*) as student_count FROM students GROUP BY grade;;

Having

SELECT grade, COUNT(*) as student_count
FROM students
GROUP BY grade
HAVING COUNT(*) > 10;
.6 9.6 RCFILE Implementation RCFile (Record Columnar File) is a data placement structure that determines how to store relational tables on computer clusters. Objective: To work
Objective: To work with RCFILE Format.

Act: CREATE TABLE STUDENT_RC(rollno int, name string,gpa float) STORED AS RCFILE; INSERT OVERWRITE table STUDENT_RC SELECT * FROM STUDENT; SELECT SUM(gpa) FROM STUDENT_RC;

The output of the provided commands in a Hive environment would typically be as follows:

1. **Creating Table:** The table STUDENT_RC will be created with the specified schema (columns rollno of type int, name of type string, and gpa of type float) stored in the RCFILE format.
2. **Inserting Data:** The data from the existing STUDENT table will be overwritten into the STUDENT_RC table. Assuming the schema of STUDENT matches that of

STUDENT_RC, all records from STUDENT will replace any existing data in STUDENT_RC.

3. **Selecting Sum of GPA:** After the data insertion, the query SELECT SUM(gpa) FROM STUDENT_RC; will calculate the sum of the gpa column from the STUDENT_RC table. The result will be a single value representing the sum of all GPA values in the STUDENT_RC table.

The specific numerical result of the SUM(gpa) query will depend on the data stored in the STUDENT table and the values of the gpa column.

4.11 SERDE: SerDe stands for Serializer/Deserializer:

1. Contains the logic to convert unstructured data into records.

2. Implemented using Java

3. Serializers are used at the time of writing.

4. Deserializers are used at query time (SELECT Statement).

Deserializer interface takes a binary representation or string of a record, converts it into an object that Hive can then manipulate. Serializer takes a object that Hive has been working with and translates it into something that Hive can write to HDFS.

Example

Objective: To manipulate XML data using SERDE (Serializer/Deserializer).

Input XML Data

xml

```
<employee>
   <empid>1001</empid>
   <name>John</name>
```

```
  <designation>Team Lead</designation>
</employee>
<employee>
  <empid>1002</empid>
  <name>Smith</name>
  <designation>Analyst</designation>
</employee>
```

Steps to Manipulate XML Data

1. **Creating Table:**

   ```
   CREATE TABLE XMLSAMPLE (xmldata STRING);
   ```

2. **Loading XML Data:**

   ```
   LOAD DATA LOCAL INPATH '/root/hivedemos/input.xml' INTO TABLE
   XMLSAMPLE;
   ```

3. **Creating XPath Table:**

   ```
   CREATE TABLE xpath_table AS
   SELECT xpath_int(xmldata,'employee/empid'),
        xpath_string(xmldata,'employee/name'),
        xpath_string(xmldata,'employee/designation')
   FROM xmlsample;
   ```

4. **Selecting Data from XPath Table:**

   ```
   SELECT * FROM xpath_table;
   ```

Expected Outcome

The xpath_table will contain records extracted from the XML data with columns empid, name, and designation. The SELECT query will display these records.

Example Session

Step	Hive Commands	Output
1. Creating Table	CREATE TABLE XMLSAMPLE(xmldata STRING);	OK
2. Loading Data	LOAD DATA LOCAL INPATH '/root/hivedemos/input.xml' INTO TABLE XMLSAMPLE;	Loading data to table...
3. Creating Table	CREATE TABLE xpath_table AS SELECT xpath_int(xmldata,'employee/empid'), xpath_string(xmldata,'employee/name'), xpath_string(xmldata,'employee/designation') FROM xmlsample;	OK
4. Selecting Data	SELECT * FROM xpath_table;	Resulting records

4.12 User-Defined Function (UDF) in Hive

In Hive, custom functions can be defined using User-Defined Functions (UDFs). The objective is to write a Hive function that converts the values of a field to uppercase.

Act: Writing the Hive UDF

```
package com.example.hive.udf;

import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;

@Description(name="SimpleUDFExample")
public class UppercaseUDF extends UDF {

   /**
    * UDF method to convert a string to uppercase.
    *
    * @param str the input string
```

```
     * @return the uppercase version of the input string
     */
    public String evaluate(String str) {
        if (str == null) return null;
        return str.toUpperCase();
    }
}
```

Description of the UDF

- **Package and Import:** The UDF is defined within the package com.example.hive.udf. Necessary Hive classes are imported, including org.apache.hadoop.hive.ql.exec.Description and org.apache.hadoop.hive.ql.exec.UDF.
- **Annotation:** The @Description annotation provides metadata about the UDF. The name attribute specifies the name of the UDF, which is "SimpleUDFExample" in this case.
- **UDF Implementation:** The UppercaseUDF class extends the UDF class provided by Hive. It contains a single method named evaluate, which takes a string as input and returns its uppercase version. If the input string is null, the method returns null.

Outcome

After compiling and deploying this UDF in Hive, you can use it in Hive queries to convert strings to uppercase. For example:

```
SELECT SimpleUDFExample('hello') AS upper_case_string;
```

This would return 'HELLO' as the upper_case_string.

Lets sum up

Hive is a data warehousing solution built on top of Hadoop, enabling efficient querying and analysis of large datasets using a SQL-like language known as Hive Query Language (HQL). Its architecture consists of components like the Metastore, Driver, Compiler, and Execution Engine. Hive supports a variety of data types and file formats, including text, ORC, and Parquet. It provides robust querying capabilities with HQL statements, allowing for complex operations such as joins, aggregations, and sub-queries. Advanced data organization features like partitions and bucketing enhance query performance. Hive also supports views for abstracting queries, and its RCFile implementation improves storage efficiency. User Defined Functions (UDFs) enable custom operations, while serialization and deserialization mechanisms facilitate the handling of diverse data formats.

UNIT IV : SUMMARY

- **Introduction**:
  - Hive is a data warehousing infrastructure built on Hadoop, designed for querying and managing large datasets stored in Hadoop Distributed File System (HDFS).
- **Architecture**:
  - **Components**: Hive includes clients for query submission, a Hive Driver for query compilation and execution, a Metastore for metadata storage, and execution engines like MapReduce or Tez for processing queries.
- **Data Types**:
  - Hive supports primitive data types (int, string, boolean, etc.) and complex types (array, map, struct), facilitating flexible data modeling.
- **File Formats**:
  - Supports various file formats (text files, ORCFile, Parquet, RCFile) for efficient data storage and processing within HDFS.
- **Hive Query Language (HQL) Statements**:
  - HQL is SQL-like and supports commands like SELECT, INSERT, UPDATE, DELETE, facilitating data querying and management in Hive.
- **Partitions**:
  - Enable data organization based on one or more columns, improving query performance by filtering data at the partition level.
- **Bucketing**:
  - Divides data into manageable parts (buckets) using a hash function, enhancing data retrieval efficiency in Hive queries.
- **Views**:
  - Virtual tables defined by queries, allowing users to query predefined logic as if querying a table directly.
- **Sub-Query**:
  - Supports nested queries within another query to perform complex operations and facilitate data analysis.

- **Joins**:
  - Supports various join types (INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER) to combine data from multiple tables based on specified conditions.
- **Aggregations, Group by and Having**:
  - Aggregation functions (SUM, AVG, COUNT) and GROUP BY for summarizing data, with HAVING used to filter aggregated results.
- **RCFile**:
  - Columnar storage format in Hive, storing related data in columns rather than rows to improve query performance.
- **Implementation**:
  - Involves setting up Hive components, defining tables, loading data into tables stored in HDFS, and executing queries using HQL.
- **Hive User Defined Function (UDF)**:
  - Allows users to define custom functions in various programming languages to process and manipulate data during query execution.
- **Serialization and Deserialization (SerDe)**:
  - Hive uses SerDe libraries to read and write data between HDFS and Hive tables in different formats, supporting data interchange and integration.

Glossary

- **Hive**: Data warehousing infrastructure built on Hadoop for querying and managing large datasets stored in HDFS.
- **Hive Query Language (HQL)**: SQL-like language used for querying and managing data in Hive.
- **Partitions**: Organizes data based on one or more columns to improve query performance by filtering data at a partition level.
- **Bucketing**: Technique in Hive to divide data into manageable parts (buckets) based on a hash function.
- **Views**: Virtual tables defined by queries in Hive, allowing users to query predefined logic as if querying a table directly.
- **User Defined Function (UDF)**: Allows users to define custom functions to process and manipulate data during query execution.
- **Serialization and Deserialization (SerDe)**: Libraries used in Hive to read and write data between HDFS and Hive tables in various formats.
- **File Formats**: Various formats supported by Hive for data storage and processing, such as text files, ORCFile, Parquet, and RCFile.

Checkup Your Progress

EXERCISE 1: Fill in each gap with the right word from the list

1. Hive is a data warehousing infrastructure built on top of _____ for querying and managing large datasets stored in Hadoop's HDFS. (Hive, HBase, Hadoop)
2. Hive Query Language (HQL) is SQL-like and used for querying and managing data in _____. (Hive, HBase, Hadoop)
3. Hive supports various file formats for data storage and processing, including text files, ORCFile, Parquet, and _____. (JSON, RCFile, Avro)
4. Partitions in Hive enable data organization based on one or more columns, improving query performance by filtering data at the _____ level. (row, column, partition)
5. Views in Hive are virtual tables defined by a query, allowing users to query predefined queries as if they were _____. (tables, views, databases)

EXERCISE 2: Read the questions below and circle if the answer is True

1. Hive is primarily designed for transactional processing and real-time data analytics.
2. Hive Query Language (HQL) is SQL-like and used for querying and managing data in Hive.
3. Hive supports only one file format for data storage and processing.
4. Partitions in Hive improve query performance by filtering data at the row level.
5. Hive User Defined Function (UDF) allows users to define custom functions to process and manipulate data during query execution.

EXERCISE 3: Choose the correct answer

1. Hive is primarily built on top of which framework for querying and managing large datasets stored in Hadoop's HDFS?
   a) Hive
   b) HBase
   c) Hadoop

2. Which language is used in Hive for querying and managing data?
   a) Java
   b) SQL
   c) Python

3. Which of the following is a columnar storage format supported by Hive?
   a) JSON
   b) Avro
   c) RCFile

4. What does partitioning in Hive improve?
a) Data replication
b) Query performance
c) Data compression

5. What does Hive User Defined Function (UDF) allow users to define?
a) Custom data types
b) Custom functions
c) Custom tables

EXERCISE 4: Match the following

1. Hive Query Language (HQL)	-	A. Allows users to define custom functions to process and manipulate data

2. Partitioning - B. SQL-like language used for querying and managing data in Hive
3. RCFile - C. Improves query performance by storing related data in columns rather than rows
4. Hive User Defined Function (UDF) - D. Virtual tables defined by a query, allowing users to query predefined queries
5. Views - E. Enables data organization based on one or more columns, improving query performance

EXERCISE 5: Self-Assessment Questions

1. What is the primary purpose of Hive in the Hadoop ecosystem?
2. Name two common data types supported by Hive.
3. How does partitioning in Hive improve query performance?
4. Write a basic HiveQL statement to create a table.
5. What is the difference between serialization and deserialization in the context of Hive?

Answers for Checkup Your Progress

EXERCISE 1:

1. Hadoop 2.Hive 3.RCFile 4.Partition 5. views

EXERCISE 2:

1. False 2. True 3. False 4. False 5. Ture

EXERCISE 3:

1. c) 2. b) 3. c) 4. b) 5. b)

EXERCISE 4:

1. B, 2.E, 3.C, 4.A, 5.D

Open Source E-content Links:

1. https://www.guru99.com/introduction-hive.html
2. https://www.tutorialspoint.com/hive/hive_introduction.htm
3. https://www.javatpoint.com/hive-architecture
4. https://data-flair.training/blogs/hive-tutorial/

References:

1. Apache Hive. (2023). Retrieved June 28, 2024, from https://hive.apache.org/documentation/
2. Capriolo, E., Wampler, D., & Rutherglen, J. (2013). Programming Hive. Sebastopol, CA: O'Reilly Media.
3. White, T. (2015). Hadoop: The definitive guide (4th ed.). Sebastopol, CA: O'Reilly Media.
4. Cloudera. (2023). Apache Hive Tutorial. Retrieved June 28, 2024, from https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/hive.html
5. Hortonworks. (2023). Apache Hive Essentials. Retrieved June 28, 2024, from https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/hive.html

UNIT- V : PIG

UNIT V OBJECTIVE

The objective of this unit is to provide an in-depth understanding of Apache Pig, including its architecture, features, and underlying philosophy. Learners will explore the practical use cases for Pig, gain a thorough overview of Pig Latin, and become familiar with Pig's primitive and complex data types. The unit will cover how to run Pig in different execution modes, utilize HDFS commands, and apply relational operators and eval functions. Additionally, learners will learn about Piggy Bank for reusable functions, creating user-defined functions, parameter substitution, and using diagnostic operators. Practical applications, such as a word count example, will be demonstrated. The unit will also delve into Pig's application at Yahoo! and provide a comparative analysis of Pig versus Hive, equipping learners with the knowledge to leverage Pig effectively for large-scale data processing.

Unit Summary

1. Introduction to Pig

- Anatomy and Features of Pig
- Philosophy and Use Cases for Pig
- Overview of Pig Latin
- Pig Primitive Data Types
- Execution Modes of Pig
- HDFS Commands for Pig
- Relational Operators and Eval Functions in Pig
- Complex Data Types and Piggy Bank
- User-Defined Functions in Pig
- Parameter Substitution and Diagnostic Operator

- Word Count Example using Pig
- Comparison between Pig and Hive

5.1 What is Pig?

Apache Pig is a platform for data analysis. It is an alternative to MapReduce Programming. Pig was developed as a research project at Yahoo.

5.1.1 Key Features of Pig

1. It provides an engine for executing data flows (how your data should flow). Pig processes data in parallel on the Hadoop cluster.
2. It provides a language called “Pig Latin” to express data flows.
3. Pig Latin contains operators for many of the traditional data operations such as join, filter, sort, etc.
4. It allows users to develop their own functions (User Defined Functions) for reading, processing, and writing data.

5.2 The Anatomy of Pig

The main components of Pig are as follows:

1. Data flow language (Pig Latin).
2. Interactive shell where you can type Pig Latin statements (Grunt).
3. Pig interpreter and execution engine. Refer Figure 5.1.

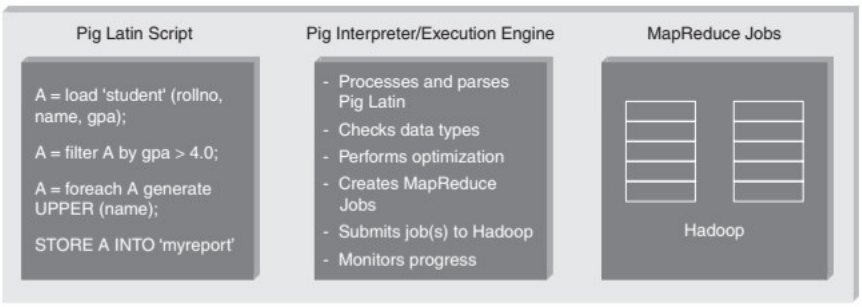


Figure 5.1 The anatomy of Pig

5.3 Pig on Hadoop

Pig runs on Hadoop. Pig uses both Hadoop Distributed File System and MapReduce Programming. By default, Pig reads input files from HDFS. Pig stores the intermediate data (data produced by MapReduce jobs) and the output in HDFS. However, Pig can also read input from and place output to other sources. Pig supports the following:

1. HDFS commands.
2. UNIX shell commands.
3. Relational operators.
4. Positional parameters.
5. Common mathematical functions.
6. Custom functions.
7. Complex data structures.

5.4 Pig Philosophy

Figure 5.2 describes the Pig philosophy.

1. Pigs Eat Anything: Pig can process different kinds of data such as structured and unstructured data.
2. Pigs Live Anywhere: Pig not only processes files in HDFS, it also processes files in other sources such as files in the local file system.
3. Pigs are Domestic Animals: Pig allows you to develop user-defined functions and the same can be included in the script for complex operations.
4. Pigs Fly: Pig processes data quickly.

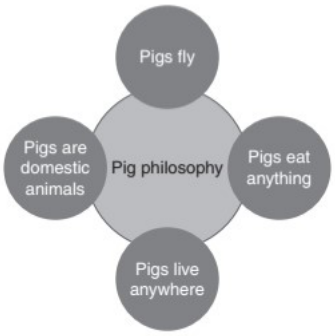


Figure 5.2 Pig Philosophy

5.5 Use Case for Pig: ETL Processing

Pig is widely used for “ETL” (Extract, Transform, and Load). Pig can extract data from different sources such as ERP, Accounting, Flat Files, etc. Pig then makes use of various operators to perform transformation on the data and subsequently loads it into the data warehouse. Refer Figure 5.3.

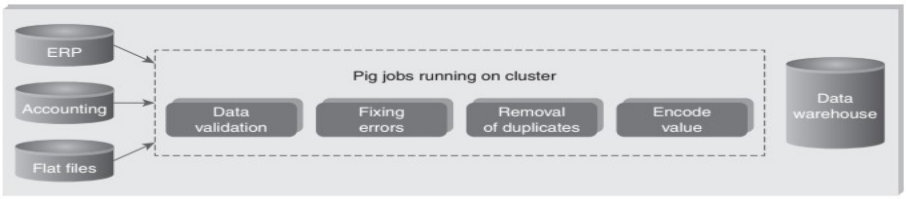


Figure 5.3 Pig: ETL Processing

5.6 Pig Latin Overview

5.6.1 Pig Latin Statements

1. Pig Latin statements are basic constructs to process data using Pig.
2. Pig Latin statement is an operator.
3. An operator in Pig Latin takes a relation as input and yields another relation as output.
4. Pig Latin statements include schemas and expressions to process data.
5. Pig Latin statements should end with a semi-colon.

Pig Latin Statements are generally ordered as follows:

1. LOAD statement that reads data from the file system.
2. Series of statements to perform transformations.
3. DUMP or STORE to display/store result.

The following is a simple Pig Latin script to load, filter, and store “student” data.

A = load 'student' (rollno, name, gpa);

A = filter A by gpa > 4.0;

A = foreach A generate UPPER (name);

STORE A INTO ‘myreport’

Note: In the above example A is a relation and NOT a variable.

5.6.2 Pig Latin: Keywords

Keywords are reserved. It cannot be used to name things.

5.6.3 Pig Latin: Identifiers

1. Identifiers are names assigned to fields or other data structures.
2. It should begin with a letter and should be followed only by letters, numbers, and underscores.

Figure 5.3 Pig: ETL Processing. Data warehouse ERP Accounting Flat files Data validation Fixing errors Removal of duplicates Encode value Pig jobs running on cluster.

Table 5.1 describes valid and invalid identifiers.

Table 5.1 Valid and Invalid Identifiers

Valid Identifier	Y	A1	A1_2014	Sample
Invalid Identifier	**5**	**Sales $**	**Sales %**	**_Sales**

5.6.4 Pig Latin: Comments

In Pig Latin, two types of comments are supported:

1. Single line comments that begin with "--".
2. Multiline comments that begin with "/*" and end with "*/".

5.6.5 Pig Latin: Case Sensitivity

1. Keywords are not case-sensitive such as LOAD, STORE, GROUP, FOREACH, DUMP, etc.
2. Relations and paths are case-sensitive.
3. Function names are case-sensitive such as PigStorage, COUNT.

5.6.6 Operators in Pig Latin

Table 5.2 describes operators in Pig Latin.

Table 5.2 Operators in Pig Latin

Arithmetic	*Comparison*	*Null*	*Boolean*
+	=	IS NULL	AND
-	!=	IS NOT NULL	OR
*****	<		NOT

/	>
%	<=
	>=

5.7 Data Types in Pig

5.7.1 Simple Data Types

Table 5.3 describes simple data types supported in Pig. In Pig, fields of unspecified types are considered as an array of bytes, which is known as bytearray.

Null: In Pig Latin, NULL denotes a value that is unknown or non-existent.

5.7.2 Complex Data Types

Table 5.4 describes complex data types in Pig.

Table 5.3 Simple Data Types Supported in Pig

Name	*Description*
Int	Whole numbers
Long	Large whole numbers
Float	Decimals
Double	Very precise decimals
Chararray	Text strings
Bytearray	Raw bytes
Datetime	Datetime
Boolean	true or false

5.8 Running Pig

You can run Pig in two ways:

1. Interactive Mode.

2. Batch Mode.

5.8.1 Interactive Mode

You can run Pig in interactive mode by invoking the grunt shell. Type pig to get the grunt shell as shown below.

Table 5.4 Complex Data Types in Pig

Name	*Description*
Tuple	An ordered set of fields. Example: (2,3)
Bag	A collection of tuples. Example: {(2,3),(7,5)}
Map	key, value pair

5.8.2 Batch Mode

Create a "Pig Script" to run Pig in batch mode. Write Pig Latin statements in a file and save it with a .pig extension.

5.9 Execution Modes of Pig

You can execute Pig in two modes:

1. Local Mode.
2. MapReduce Mode.

5.9.1 Local Mode

To run Pig in local mode, you need to have your files in the local file system.

Syntax:

pig –x local filename

5.9.2 MapReduce Mode

To run Pig in MapReduce mode, you need to have access to a Hadoop Cluster to read/write files. This is the default mode of Pig.

Syntax:

pig filename

5.10 HDFS Commands

It is possible to work with all HDFS commands in the Grunt shell. For example, you can create a directory as shown below.

5.11 Relational Operators

5.11.1 FILTER

The FILTER operator is used to select tuples from a relation based on specified conditions.

Objective: Find the tuples of those students where the GPA is greater than 4.0.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = filter A by gpa > 4.0;
DUMP B;
```

5.11.2 FOREACH

Use FOREACH when you want to do data transformation based on columns of data.

Objective: Display the name of all students in uppercase.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = foreach A generate UPPER(name);
DUMP B;
```

5.11.3 GROUP

The GROUP operator is used to group data.

Objective: Group tuples of students based on their GPA.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = GROUP A BY gpa;
DUMP B;
```

5.11.4 DISTINCT

The DISTINCT operator is used to remove duplicate tuples. In Pig, the DISTINCT operator works on the entire tuple and NOT on individual fields.

Objective: To remove duplicate tuples of students.

Input:

Student (rollno:int, name:chararray, gpa:float)

Example Data:

```
1001 John  3.0
1002 Jack  4.0
1003 Smith 4.5
1004 Scott 4.2
1005 Joshi 3.5
1006 Alex  4.5
1007 David 4.2
1008 James 4.0
1001 John  3.0
1005 Joshi 3.5
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = DISTINCT A;
DUMP B;
```

5.11.5 LIMIT

The LIMIT operator is used to limit the number of output tuples.

Objective: Display the first 3 tuples from the “student” relation.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = LIMIT A 3;
DUMP B;
```

5.11.6 ORDER BY

The ORDER BY operator is used to sort a relation based on a specific value.

Objective: Display the names of the students in ascending order.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = ORDER A BY name;
DUMP B;
```

5.11.7 JOIN

The JOIN operator is used to join two or more relations based on values in the common field. It always performs an inner join.

Objective: To join two relations, namely, "student" and "department", based on the values contained in the "rollno" column.

Input:

Student (rollno:int, name:chararray, gpa:float)
Department (rollno:int, deptno:int, deptname:chararray)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = load '/pigdemo/department.tsv' as (rollno:int, deptno:int, deptname:chararray);
C = JOIN A BY rollno, B BY rollno;
DUMP C;
DUMP B;
```

5.11.8 UNION

The UNION operator is used to merge the contents of two relations.

Objective: To merge the contents of two relations “student” and “department”.

Input:

Student (rollno:int, name:chararray, gpa:float)
Department (rollno:int, deptno:int, deptname:chararray)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno, name, gpa);
B = load '/pigdemo/department.tsv' as (rollno, deptno, deptname);
C = UNION A, B;
STORE C INTO '/pigdemo/uniondemo';
DUMP B;
```

5.11.9 SPLIT

It is used to partition a relation into two or more relations.

Objective: To partition a relation based on the GPAs acquired by the students.

- If GPA = 4.0, place it into relation X.
- If GPA is < 4.0, place it into relation Y.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
SPLIT A INTO X IF gpa == 4.0, Y IF gpa <= 4.0;
DUMP X;
```

5.11.10 SAMPLE

It is used to select a random sample of data based on the specified sample size.

Objective: To depict the use of SAMPLE.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = SAMPLE A 0.01;
DUMP B;
```

5.12 Eval Function

5.12.1 AVG: AVG is used to compute the average of numeric values in a single column bag.

Objective: To calculate the average marks for each student.

Input:

Student (studname:chararray, marks:int)

Action:

```
A = load '/pigdemo/student.csv' USING PigStorage(',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE group AS studname, AVG(A.marks);
DUMP C;
```

5.12.2 MAX: MAX is used to compute the maximum marks for each student.

Objective: To calculate the maximum marks for each student.

Input:

Student (studname:chararray, marks:int)

Action:

```
A = load '/pigdemo/student.csv' USING PigStorage(',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE group AS studname, MAX(A.marks);
DUMP C;
```

5.12.3 COUNT:COUNT is used to count the number of elements in a bag.

Objective: To count the number of tuples in a bag.

Input:

Student (studname:chararray, marks:int)

Action:

A = load '/pigdemo/student.csv' USING PigStorage(',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE group AS studname, COUNT(A);
DUMP C;

5.13 Complex Data Types

5.13.1 TUPLE

A TUPLE is an ordered collection of fields.

Objective: To use the complex data type "Tuple" to load data.

Input:

(John,12)
(Jack,13)
(James,7)
(Joseph,5)
(Smith,8)
(Scott,12)

Action:

```
A = LOAD '/root/pigdemos/studentdata.tsv' AS (t1:tuple(t1a:chararray, t1b:int), t2:tuple(t2a:chararray, t2b:int));
B = FOREACH A GENERATE t1.t1a, t1.t1b, t2.$0, t2.$1;
DUMP B;
```

5.13.2 MAP

MAP represents a key/value pair.

Objective: To depict the complex data type "map".

Input:

```
John [city#Bangalore]
Jack [city#Pune]
James [city#Chennai]
```

Action:

```
A = load '/root/pigdemos/studentcity.tsv' Using PigStorage as (studname:chararray, m:map[chararray]);
B = foreach A generate m#'city' as CityName:chararray;
DUMP B
```

5.14 Piggy Bank

Pig users can use Piggy Bank functions in Pig Latin script, and they can also share their functions in Piggy Bank.

Objective: To use Piggy Bank string UPPER function.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
register '/root/pigdemos/piggybank-0.12.0.jar';
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
upper = foreach A generate org.apache.pig.piggybank.evaluation.string.UPPER(name);
DUMP upper;
```

5.15 User-Defined Functions (UDF)

Pig allows you to create your own function for complex analysis.

Objective: To depict a user-defined function.

Java Code to convert name into uppercase:

```
java
Copy code
package myudfs;

import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;

public class UPPER extends EvalFunc<String> {
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try {
            String str = (String)input.get(0);
            return str.toUpperCase();
```

```
        } catch(Exception e) {
            throw WrappedIOException.wrap("Caught exception processing input row", e);
        }
    }
}
```

Note: Convert the above Java class into a JAR file to include this function in your code.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
register /root/pigdemos/myudfs.jar;
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = FOREACH A GENERATE myudfs.UPPER(name);
DUMP B;
```

5.16 Parameter Substitution

Pig allows you to pass parameters at runtime.

Objective: To depict parameter substitution.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

```
A = load '$student' as (rollno:int, name:chararray, gpa:float);
DUMP A;
```

Execute:

pig –param student=/pigdemo/student.tsv parameterdemo.pig

5.17 Diagnostic Operator

It returns the schema of a relation.

Objective: To depict the use of DESCRIBE.

Input:

Student (rollno:int, name:chararray, gpa:float)

Action:

A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
DESCRIBE A;

5.18 Word Count Example using Pig

Objective: To count the occurrence of similar words in a file.

Input:

Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session

Action:

lines = LOAD '/root/pigdemos/lines.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped

Output:

(Introduction,1)
(Hadoop,2)
(Hive,2)
(Pig,1)
(Session,2)
(to,2)
(Welcome,1)
(Introducing,1)

5.19 When to use Pig? Pig can be used in the following situations:

1. When your data loads are time sensitive.
2. When you want to process various data sources.
3. When you want to get analytical insights through sampling.

5.20 When NOT to use Pig? Pig should not be used in the following situations:

1. When your data is completely in the unstructured form such as video, text, and audio.
2. When there is a time constraint because Pig is slower than MapReduce jobs.

5.21 Pig at Yahoo! Yahoo uses Pig for two things:

1. In Pipelines, to fetch log data from its web servers and to perform cleansing to remove interval views and clicks.
2. In Research, script is used to test a theory. Pig provides a facility to integrate Perl or Python script which can be executed on a huge dataset.

5.22 Pig versus Hive Features

Features	Pig	Hive
Used By	Programmers and Researchers	Analyst
Used For	Programming	Reporting
Language	Procedural data flow language	SQL Like
Suitable For	Semi-Structured	Structured
Schema/Types	Explicit	Implicit
UDF Support	YES	YES
Join/Order/Sort	YES	YES
DFS Direct Access	YES (Implicit)	YES (Explicit)
Web Interface	YES	NO
Partitions	YES	NO
Shell	YES	YES

This table provides a comparison between Pig and Hive based on the features such as usage, language, suitability, schema/types, support for UDFs, operations like join/order/sort, direct access to distributed file system (DFS), availability of web interface, support for partitions, and shell interface.

Lets sum up

Pig is a high-level platform for processing large datasets in Hadoop, known for its scripting language Pig Latin. Pig's architecture includes components like the Pig Latin interpreter and execution environment, and it boasts features such as simplicity, extensibility, and ease of use. The philosophy of Pig centers on handling semi-structured data and performing data transformations, with use cases spanning ETL processes and data analysis. Pig Latin provides a rich set of primitive data types and supports complex data types. It operates in local and MapReduce modes and utilizes HDFS commands. Pig includes relational operators and eval functions for data manipulation, and the Piggy Bank library extends functionality with user-defined

functions (UDFs). Parameter substitution and diagnostic operators enhance script flexibility and debugging. A classic example of Pig's application is the word count problem. Compared to Hive, Pig is more procedural, offering a different approach to big data processing and analysis.

UNIT V : SUMMARY

- **Introduction:**
  - Pig is a high-level scripting language used for analyzing large datasets.
  - It simplifies complex data transformations and processing on Hadoop.
- **Anatomy:**
  - Pig scripts consist of operations represented as data flow sequences.
  - Scripts are executed using Pig Latin, a data flow language.
- **Features:**
  - Supports complex data types and operations like filtering, grouping, and joining datasets.
  - Provides extensibility through User-Defined Functions (UDFs) and libraries like Piggy Bank.
- **Philosophy:**
  - Designed for ease of use, productivity, and extensibility in data processing tasks.
  - Focuses on data flow and transformation rather than low-level programming.
- **Use Case for Pig:**
  - Ideal for ETL (Extract, Transform, Load) processes, data pipelines, and ad-hoc data exploration tasks.
- **Pig Latin Overview:**
  - Declarative language similar to SQL but tailored for Hadoop environments.
  - Allows users to specify data transformations using operations like LOAD, FILTER, GROUP, and STORE.
- **Pig Primitive Data Types:**
  - Includes basic types such as int, long, float, double, chararray, bytearray, and boolean.
- **Running Pig:**

- Pig scripts are executed using the Pig Latin interpreter, which compiles scripts into MapReduce jobs.

- **Execution Modes of Pig:**
  - Supports local mode for testing and MapReduce mode for distributed execution on Hadoop clusters.
- **HDFS Commands:**
  - Pig interacts with HDFS (Hadoop Distributed File System) for data storage and retrieval.
- **Relational Operators:**
  - Includes operators like JOIN, FILTER, DISTINCT, ORDER BY, and FOREACH for data manipulation.
- **Eval Function:**
  - Used in Pig Latin to apply expressions and functions to data fields.
- **Complex Data Types:**
  - Supports tuples, bags (unordered collections), and maps for handling nested and complex data structures.
- **Piggy Bank:**
  - Library of reusable Pig scripts and UDFs contributed by the community.
- **User-Defined Functions (UDFs):**
  - Custom functions written in Java or other languages to extend Pig's functionality.
- **Parameter Substitution:**
  - Allows users to parameterize Pig scripts for flexibility and reuse.
- **Diagnostic Operator:**
  - Used for debugging by displaying intermediate results during script execution.
- **Word Count Example using Pig:**
  - Classic example demonstrating Pig's capability to count words in a dataset using simple operations.
- **Pig at Yahoo!:**

- Yahoo! extensively used Pig for various data processing tasks, showcasing its scalability and effectiveness.

- **Pig Versus Hive:**
  - Comparison between Pig and Hive, highlighting differences in their query languages, data processing paradigms, and use cases.

Glossary

- **Apache Pig**: A platform for analyzing large data sets that provides a high-level language called Pig Latin for expressing data analysis programs, which are compiled into sequences of MapReduce programs, reducing the complexities of writing raw MapReduce code.
- **Pig Latin**: A high-level scripting language used with Apache Pig that allows users to write complex data transformations without having to use Java. Pig Latin includes many operators for data operations like join, filter, and sort.
- **Grunt**: The interactive shell for running Pig scripts and commands. It allows users to enter Pig Latin statements and immediately see the results.

- **Pig Primitive Data Types**: Basic data types in Pig Latin including int, long, float, double, chararray (string), and bytearray (binary data).
- **Pig Complex Data Types**: Data types that can hold multiple values or nested values, such as tuple, bag, and map.
- **Relational Operators**: Operators in Pig Latin used to manipulate and transform data. Examples include LOAD, STORE, FILTER, FOREACH, JOIN, GROUP, and ORDER BY.
- **Eval Functions**: Functions in Pig Latin used to perform operations on data fields, such as mathematical computations or string manipulations.
- **Piggy Bank**: A collection of user-contributed functions for Pig that extend its capabilities with additional data processing operations not included in the core Pig distribution.
- **User-Defined Functions (UDFs)**: Custom functions written by users in Java, Python, or other supported languages to extend Pig's functionality by adding new operations or transformations.
- **Parameter Substitution**: A feature in Pig that allows users to pass parameters to Pig scripts at runtime, enabling dynamic script configuration.
- **Diagnostic Operators**: Tools in Pig used for debugging and troubleshooting Pig scripts, such as DESCRIBE, EXPLAIN, and ILLUSTRATE.

- **ETL (Extract, Transform, Load)**: A common use case for Pig, involving the extraction of data from various sources, transformation of the data using Pig operations, and loading the processed data into a data warehouse or other storage systems.

Checkup Your Progress

EXERCISE 1: Fill in each gap with the right word from the list

1. Pig is a high-level scripting language used for analyzing large datasets on _____ (Hadoop, Spark, MongoDB).
2. Pig scripts are executed using _____ (Pig Latin, SQL, Java), a data flow language.
3. Pig supports complex data types such as tuples, bags, and _____ (arrays, maps, sets).
4. Pig scripts can be run in _____ (local mode, standalone mode, cluster mode) for testing purposes.
5. Pig provides a variety of relational operators such as _____ (JOIN, UPDATE, DELETE) for data manipulation.

EXERCISE 2: Read the questions below and circle if the answer is True

1. Pig is a low-level programming language used for analyzing large datasets - True/False
2. Pig Latin is a procedural language used to define data flows in Pig scripts - True/False
3. Pig supports only primitive data types such as int, float, and string - True/False
4. Pig scripts can be executed in local mode for testing and debugging purposes - True/False
5. Piggy Bank is a library of reusable Pig scripts and User-Defined Functions (UDFs) contributed by the community - True/False

EXERCISE 3: Choose the correct answer

1. Which of the following is true about Pig?

   a) It is a low-level programming language.
   b) It is used for real-time data processing.

c) It simplifies complex data transformations on Hadoop.

2. Pig scripts are written in which language?

   a) Pig Latin
   b) Java
   c) SQL

3. Which data types does Pig support?

   a) Only primitive data types
   b) Primitive and complex data types
   c) Only complex data types

4. What mode can Pig scripts be executed in for testing purposes?

   a) Standalone mode
   b) Cluster mode
   c) Local mode

5. What is Piggy Bank?

   a) A data repository in Pig
   b) A visualization tool for Pig scripts
   c) A library of reusable scripts and User-Defined Functions (UDFs)

EXERCISE 4: Match the following

1. Pig Latin - A. Allows users to extend Pig's functionality by defining custom operations

2. Execution - B. Library of reusable Pig scripts and functions contributed by

	Modes of Pig		the community
3.	Piggy Bank	-	C. Language used to define data flows and transformations in Pig scripts
4.	User-Defined Functions (UDFs)	-	D. Allows for dynamic values to be used in Pig scripts
5.	Parameter Substitution	-	E. Defines how Pig scripts are run, such as local mode or cluster mode

EXERCISE 5: Self-Assessment Questions

1. What is the main purpose of Apache Pig in the Hadoop ecosystem?
2. Name two primitive data types in Pig.
3. Describe one use case where Pig is particularly advantageous.
4. What are the two execution modes of Pig, and what is the main difference between them?
5. Write a simple Pig Latin script to load data from an HDFS file.

Answers for Checkup Your Progress

EXERCISE 1:

1. Hadoop, 2. Pig Latin 3. Maps, 4. local mode, 5. JOIN

EXERCISE 2:

1. False 2. False 3. False 4. True 5. Ture

EXERCISE 3:

1. c) 2. a) 3. b) 4. c) 5. c)

EXERCISE 4:

1. C, 2.E, 3.B, 4.A, 5.D.

Open Source E-content Links:

1. https://www.geeksforgeeks.org/introduction-to-apache-pig/
2. https://www.tutorialspoint.com/apache_pig/index.htm
3. https://data-flair.training/blogs/hadoop-pig-tutorial/
4. https://www.javatpoint.com/pig-example

References:

1. Gates, A., & Natkovich, O. (2017). Apache Pig: The definitive guide. Sebastopol, CA: O'Reilly Media.
2. Apache Pig. (2023). Retrieved June 28, 2024, from https://pig.apache.org/docs/latest/
3. Sankar, K., & Gates, A. (2015). Learning Apache Pig. Birmingham, UK: Packt Publishing.
4. Hortonworks. (2023). Apache Pig Tutorial. Retrieved June 28, 2024, from https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/pig.html
5. Cloudera. (2023). Apache Pig Tutorial. Retrieved June 28, 2024, from https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/pig.html

Unit-wise Assignments

The following assignments are designed to help students grasp key concepts and apply their knowledge practically.

Unit 1: Big Data and Analytics

Assignment 1: Understanding Data Types

- **Objective:** To understand the differences between structured, semi-structured, and unstructured data.
- **Task:**
    - Collect samples of structured, semi-structured, and unstructured data (e.g., a relational database table, an XML/JSON file, and a collection of tweets).
    - Write a report explaining the characteristics of each type of data, including its structure and use cases.

Assignment 2: Evolution and Challenges of Big Data

- **Objective:** To explore the history and challenges of Big Data.
- **Task:**
    - Research the evolution of Big Data technologies.
    - Write an essay detailing key milestones in the development of Big Data, major technological advancements, and the challenges faced in managing and processing Big Data.

Assignment 3: Big Data vs. Traditional Business Intelligence

- **Objective:** To compare Big Data analytics with traditional business intelligence.
- **Task:**

- Create a comparison chart that highlights the differences between Big Data analytics and traditional BI in terms of data volume, variety, velocity, and tools used.
- Provide case studies or examples where Big Data analytics provided better insights compared to traditional BI.

Assignment 4: Big Data Analytics Tools

- **Objective:** To familiarize with popular Big Data analytics tools.
- **Task:**
  - Choose three popular Big Data tools (e.g., Hadoop, Spark, Flink).
  - Write a report comparing their features, strengths, and weaknesses, and suggest scenarios where each tool is most effective.

Assignment 5: Data Governance and Quality in Big Data

- **Objective:** To understand the importance of data quality and governance.
- **Task:**
  - Research and write a paper on the best practices for ensuring data quality and implementing data governance in Big Data environments.
  - Include examples of companies that have successfully implemented these practices.

Unit 2: Technology Landscape

Assignment 1: NoSQL Databases

- **Objective:** To explore the world of NoSQL databases.
- **Task:**
  - Choose two NoSQL databases (e.g., MongoDB, Cassandra) and compare their data models, query languages, and use cases.

- Implement a simple project using one of the NoSQL databases to store and retrieve data.

Assignment 2: Hadoop Ecosystem

- **Objective:** To understand the components of the Hadoop ecosystem.
- **Task:**
  - Create a detailed diagram of the Hadoop ecosystem, including HDFS, MapReduce, YARN, and other related tools.
  - Write a report explaining the function of each component and how they interact with each other.

Assignment 3: Distributed Computing Challenges

- **Objective:** To comprehend the challenges in distributed computing and how Hadoop addresses them.
- **Task:**
  - Write an essay on the challenges of distributed computing (e.g., data consistency, fault tolerance, scalability).
  - Explain how Hadoop's architecture addresses these challenges.

Assignment 4: HDFS Deep Dive

- **Objective:** To understand the Hadoop Distributed File System.
- **Task:**
  - Write a detailed report on HDFS architecture, including block storage, replication, and fault tolerance mechanisms.
  - Include a case study of a company using HDFS to handle large-scale data storage.

Assignment 5: Comparing SQL and NoSQL

- **Objective:** To compare traditional SQL databases with NoSQL databases.

- **Task:**
    - Create a comparison matrix that outlines the differences between SQL and NoSQL databases in terms of scalability, flexibility, consistency, and performance.
    - Provide examples of scenarios where NoSQL databases are preferred over SQL databases and vice versa.

Unit 3: MongoDB and MapReduce Programming

Assignment 1: MongoDB Schema Design

- **Objective:** To design a flexible schema using MongoDB.
- **Task:**
    - Choose a real-world scenario (e.g., e-commerce, social media).
    - Design a MongoDB schema for this scenario, including collections and sample documents.
    - Explain your design choices in a written report.

Assignment 2: Implementing CRUD Operations in MongoDB

- **Objective:** To practice basic operations in MongoDB.
- **Task:**
    - Create a MongoDB database and perform CRUD (Create, Read, Update, Delete) operations using the MongoDB shell or a programming language of your choice.
    - Document each operation with code snippets and explanations.

Assignment 3: MapReduce Job Implementation

- **Objective:** To implement a MapReduce job.
- **Task:**

- Write a MapReduce program to analyze a large dataset (e.g., word count, log file analysis).
- Execute the program on a Hadoop cluster or a local setup and document the process and results.

Assignment 4: Aggregation Framework in MongoDB

- **Objective:** To use MongoDB's aggregation framework.
- **Task:**
  - Create an aggregation pipeline to perform complex data analysis on a sample dataset.
  - Write a report explaining each stage of the pipeline and the results obtained.

Assignment 5: Comparing MongoDB with Traditional RDBMS

- **Objective:** To compare MongoDB with traditional relational databases.
- **Task:**
  - Write an essay comparing MongoDB with a traditional RDBMS (e.g., MySQL) in terms of schema design, scalability, performance, and use cases.
  - Include a section on the advantages and disadvantages of each approach.

Unit 4: Hive

Assignment 1: Hive Architecture Analysis

- **Objective:** To understand the architecture of Hive.
- **Task:**
  - Write a detailed report on the components of Hive's architecture, including the metastore, query processor, and execution engine.

- Explain how Hive translates SQL-like queries into MapReduce jobs.

Assignment 2: Creating and Querying Hive Tables

- **Objective:** To practice creating and querying Hive tables.
- **Task:**
  - Create Hive tables using different data types and file formats.
  - Write and execute Hive queries to perform basic data analysis tasks (e.g., filtering, grouping, joining).
  - Document the process and results.

Assignment 3: Hive Partitioning and Bucketing

- **Objective:** To implement partitioning and bucketing in Hive.
- **Task:**
  - Create a partitioned and bucketed Hive table based on a sample dataset.
  - Write queries to analyze the data and compare the performance with non-partitioned/bucketed tables.

Assignment 4: Implementing UDFs in Hive

- **Objective:** To extend Hive's functionality using user-defined functions (UDFs).
- **Task:**
  - Write a custom UDF in Java or Python to perform a specific data transformation.
  - Register and use the UDF in Hive queries.
  - Document the implementation and usage.

Assignment 5: Optimization Techniques in Hive

- **Objective:** To optimize Hive queries for performance.
- **Task:**

- Research and implement various Hive query optimization techniques (e.g., indexing, query rewriting, join optimization).
- Write a report detailing the techniques used and their impact on query performance.

Unit 5: Pig

Assignment 1: Introduction to Pig Latin

- **Objective:** To learn the basics of Pig Latin scripting language.
- **Task:**
  - Write a Pig Latin script to load, process, and store a sample dataset.
  - Document each step of the script and explain the operations performed.

Assignment 2: Relational Operators in Pig

- **Objective:** To use relational operators in Pig.
- **Task:**
  - Write a Pig script to perform common relational operations (e.g., join, filter, group, order) on a sample dataset.
  - Explain the purpose and output of each operation.

Assignment 3: User-Defined Functions in Pig

- **Objective:** To create and use UDFs in Pig.
- **Task:**
  - Implement a UDF in Java or Python to perform a specific data processing task.
  - Integrate the UDF into a Pig script and demonstrate its usage.
  - Document the implementation and results.

Assignment 4: Diagnostic Operators in Pig

- **Objective:** To use diagnostic operators for debugging Pig scripts.
- **Task:**
    - Write a Pig script that includes diagnostic operators (e.g., DESCRIBE, ILLUSTRATE, EXPLAIN).
    - Use these operators to analyze the script's execution plan and intermediate results.
    - Document the findings and any optimizations made based on the diagnostics.

Assignment 5: Complex Data Types in Pig

- **Objective:** To work with complex data types in Pig.
- **Task:**
    - Write a Pig script that uses complex data types (e.g., tuples, bags, maps) to process a sample dataset.
    - Explain the data types used and their advantages in the script.
    - Document the script and results.

Question Bank

Multiple Choice Questions

Unit 1: Big Data and Analytics

1. **Which of the following is an example of structured data?**
    - A. Audio file
    - B. XML file
    - C. SQL database
    - D. Social media post
2. **What characteristic of Big Data refers to the variety of data types and sources?**
    - A. Volume
    - B. Velocity
    - C. Variety
    - D. Veracity
3. **Which of the following is a challenge associated with Big Data?**
    - A. High storage cost
    - B. Data integration
    - C. Lack of volume
    - D. Simple analysis tools
4. **What does 'Velocity' in Big Data refer to?**
    - A. The size of the data
    - B. The speed at which data is generated
    - C. The types of data
    - D. The accuracy of the data
5. **What is the main difference between traditional business intelligence and Big Data?**
    - A. BI handles structured data, Big Data handles unstructured data
    - B. BI is faster than Big Data
    - C. BI uses more advanced analytics than Big Data

- D. BI is less scalable than Big Data

6. **Which of the following describes a data warehouse?**
   - A. A system that processes real-time data streams
   - B. A system for storing large amounts of structured data
   - C. A system that stores unstructured data
   - D. A system that provides in-memory data processing
7. **What is Hadoop primarily used for?**
   - A. Data mining
   - B. Data warehousing
   - C. Distributed storage and processing
   - D. Real-time analytics
8. **What is the role of a Data Scientist?**
   - A. To manage databases
   - B. To perform advanced analytics on data
   - C. To develop software
   - D. To ensure data security
9. **Which of the following is a common characteristic of Big Data?**
   - A. High accuracy
   - B. Low cost
   - C. High volume
   - D. Low velocity
10. **What does 'Veracity' refer to in the context of Big Data?**
    - A. Data speed
    - B. Data variety
    - C. Data accuracy and trustworthiness
    - D. Data size
11. **Which model ensures availability and eventual consistency in Big Data systems?**
    - A. ACID
    - B. BASE
    - C. CAP

- D. RDBMS

12. **Which of the following is NOT a Big Data analytics tool?**
    - A. Hadoop
    - B. Excel
    - C. Spark
    - D. Flink
13. **What is the primary purpose of Big Data analytics?**
    - A. To replace traditional databases
    - B. To analyze large and complex data sets
    - C. To store large volumes of data
    - D. To perform simple data operations
14. **In Big Data, what does the '3 Vs' model refer to?**
    - A. Volume, Variety, Velocity
    - B. Volume, Veracity, Velocity
    - C. Volume, Variety, Value
    - D. Volume, Value, Velocity
15. **Which term is used to describe the integration and analysis of large data sets?**
    - A. Data mining
    - B. Data warehousing
    - C. Big Data analytics
    - D. Business intelligence
16. **What does 'Scalability' mean in the context of Big Data?**
    - A. The ability to handle increasing data volume
    - B. The ability to analyze data in real-time
    - C. The ability to maintain data accuracy
    - D. The ability to integrate multiple data sources
17. **Which of the following is an example of unstructured data?**
    - A. A relational database table
    - B. A CSV file
    - C. A video file

- D. A JSON document

18. **What is the Hadoop Distributed File System (HDFS) designed for?**
    - A. In-memory data processing
    - B. Real-time data analysis
    - C. Distributed data storage
    - D. Data visualization
19. **Which of the following best describes 'Data Science'?**
    - A. The study of managing databases
    - B. The study of algorithms and statistics to gain insights from data
    - C. The development of software applications
    - D. The management of IT infrastructure
20. **What is the main benefit of using Hadoop for Big Data processing?**
    - A. Low cost
    - B. High performance
    - C. Scalability and fault tolerance
    - D. User-friendly interface

Unit 2: Technology Landscape

1. **What does NoSQL stand for?**
    - A. Not Only SQL
    - B. New SQL
    - C. No Standard Query Language
    - D. Network SQL
2. **Which type of database is designed to handle large volumes of unstructured data?**
    - A. RDBMS
    - B. NoSQL
    - C. OLAP
    - D. OLTP
3. **What is a key difference between SQL and NoSQL databases?**

- A. SQL databases are schema-less
- B. NoSQL databases are schema-less
- C. SQL databases are designed for scalability
- D. NoSQL databases do not support transactions

4. **Which component of Hadoop is responsible for resource management?**
   - A. HDFS
   - B. MapReduce
   - C. YARN
   - D. Hive
5. **Which of the following is NOT a challenge of distributed computing?**
   - A. Fault tolerance
   - B. Network latency
   - C. Centralized control
   - D. Data locality
6. **What is the main function of the Hadoop Distributed File System (HDFS)?**
   - A. To process large datasets
   - B. To manage cluster resources
   - C. To store large datasets across multiple nodes
   - D. To run SQL queries on large datasets
7. **Which Hadoop component is used for processing data in a distributed environment?**
   - A. HDFS
   - B. YARN
   - C. MapReduce
   - D. Oozie
8. **What does YARN stand for?**
   - A. Yet Another Resource Negotiator
   - B. Yet Another Random Number
   - C. Your Another Resource Network
   - D. Your Another Random Negotiator

9. **Which of the following is an advantage of NoSQL databases over traditional RDBMS?**
    - A. Better performance for complex queries
    - B. Better support for unstructured data
    - C. Stronger ACID compliance
    - D. More rigid schema design
10. **What is a common use case for NoSQL databases?**
    - A. Transactional applications
    - B. Analytical processing
    - C. Handling large volumes of unstructured data
    - D. Simple data retrieval
11. **Which feature is a hallmark of NoSQL databases?**
    - A. Fixed schema
    - B. Support for joins
    - C. Horizontal scalability
    - D. Strong ACID properties
12. **What is the primary purpose of Hadoop MapReduce?**
    - A. To store data in a distributed manner
    - B. To provide a programming model for processing large datasets
    - C. To manage cluster resources
    - D. To provide a user interface for Hadoop
13. **Which Hadoop component allows for running different types of distributed applications beyond MapReduce?**
    - A. HDFS
    - B. YARN
    - C. Hive
    - D. Pig
14. **Which SQL feature is typically not supported in NoSQL databases?**
    - A. Basic CRUD operations
    - B. ACID transactions
    - C. Indexing

- D. High availability

15. **What is the key benefit of using Hadoop for Big Data analytics?**
    - A. Cost efficiency
    - B. Real-time processing
    - C. Scalability and fault tolerance
    - D. Ease of use
16. **Which term describes the capability to expand storage and processing capacity by adding more nodes to a Hadoop cluster?**
    - A. Scalability
    - B. Elasticity
    - C. Reliability
    - D. Availability
17. **Which of the following is a NoSQL database?**
    - A. MySQL
    - B. PostgreSQL
    - C. MongoDB
    - D. Oracle
18. **In Hadoop, what is the role of the NameNode?**
    - A. To store data blocks
    - B. To manage metadata and file system namespace
    - C. To execute MapReduce jobs
    - D. To provide a user interface
19. **Which statement about Hadoop YARN is true?**
    - A. It is a storage component
    - B. It allows multiple data processing engines to run on the same cluster
    - C. It replaces HDFS
    - D. It only supports MapReduce
20. **What type of data processing is Hadoop MapReduce best suited for?**
    - A. Real-time data processing
    - B. Batch processing of large datasets
    - C. Small-scale data processing

- D. In-memory processing

Unit 3: MongoDB and MapReduce Programming

1. **Which of the following is a primary feature of MongoDB?**
   - A. Relational tables
   - B. Schema-less data model
   - C. ACID transactions
   - D. Fixed schema
2. **In MongoDB, a document is analogous to what structure in an RDBMS?**
   - A. Table
   - B. Row
   - C. Column
   - D. Index
3. **Which command in MongoDB is used to insert a document into a collection?**
   - A. add()
   - B. insert()
   - C. create()
   - D. put()
4. **Which data type is NOT supported by MongoDB?**
   - A. String
   - B. Integer
   - C. Binary
   - D. Boolean
5. **What is the role of a Mapper in a MapReduce job?**
   - A. To combine results
   - B. To partition data
   - C. To process input data and produce key-value pairs
   - D. To aggregate data
6. **What function does a Reducer perform in a MapReduce job?**

- A. Sorts the input data
- B. Processes intermediate key-value pairs to produce final output
- C. Partitions the data
- D. Combines the output of mappers

7. **In MapReduce, what is the purpose of a Combiner?**
   - A. To partition the data
   - B. To perform local aggregation of intermediate outputs
   - C. To distribute the data evenly across reducers
   - D. To handle data compression
8. **Which of the following is NOT a term used in MongoDB?**
   - A. Collection
   - B. Document
   - C. Table
   - D. Field
9. **What is the default storage format for MongoDB data?**
   - A. CSV
   - B. JSON
   - C. BSON
   - D. XML
10. **In MongoDB, which operator is used to query documents that match a specified condition?**
    - A. $match
    - B. $query
    - C. $find
    - D. $select
11. **Which phase in the MapReduce process sorts and shuffles the intermediate data?**
    - A. Map
    - B. Reduce
    - C. Combine
    - D. Shuffle and Sort

12. **Which of the following is a built-in data type in MongoDB?**
    - A. Date
    - B. Currency
    - C. Complex
    - D. Matrix
13. **What is a key advantage of using MongoDB over traditional RDBMS?**
    - A. Support for complex joins
    - B. Flexible schema design
    - C. Strong ACID properties
    - D. Centralized architecture
14. **What is a Partitioner in MapReduce used for?**
    - A. To map input data
    - B. To reduce intermediate outputs
    - C. To determine how the data is split among reducers
    - D. To combine map outputs
15. **Which of the following commands is used to update a document in MongoDB?**
    - A. modify()
    - B. update()
    - C. change()
    - D. replace()
16. **In MapReduce, what does the term 'shuffle' refer to?**
    - A. Moving input data to the mappers
    - B. Distributing intermediate data to reducers
    - C. Sorting the final output
    - D. Aggregating the map outputs
17. **Which command in MongoDB is used to delete a document?**
    - A. remove()
    - B. delete()
    - C. drop()
    - D. erase()

18. **In MongoDB, what is a 'Collection'?**
    - A. A set of databases
    - B. A set of documents
    - C. A set of tables
    - D. A set of fields
19. **What is the main purpose of using a Combiner in MapReduce?**
    - A. To reduce the amount of data transferred to the reducer
    - B. To combine the final outputs
    - C. To partition the data
    - D. To perform initial data mapping
20. **Which MongoDB data type is used to store binary data?**
    - A. Blob
    - B. ByteArray
    - C. Binary
    - D. Buffer

Unit 4: Hive

1. **Which of the following is a supported data type in Hive?**
    - A. Integer
    - B. Float
    - C. String
    - D. All of the above
2. **Which file format is known for its high performance in Hive?**
    - A. TextFile
    - B. ORC
    - C. JSON
    - D. CSV
3. **What does HQL stand for in the context of Hive?**
    - A. Hive Query Language
    - B. Hadoop Query Language

- C. High-Level Query Language
- D. None of the above

4. **Which Hive feature allows dividing data into more manageable parts?**
   - A. Bucketing
   - B. Indexing
   - C. Partitioning
   - D. Clustering
5. **Which statement is used to create a table in Hive?**
   - A. INSERT
   - B. SELECT
   - C. CREATE
   - D. ALTER
6. **What is the purpose of the RCFile format in Hive?**
   - A. To provide a columnar storage format
   - B. To store data in a row-wise fashion
   - C. To compress data
   - D. To improve network performance
7. **In Hive, what is a UDF?**
   - A. User-Defined Function
   - B. Unified Data Framework
   - C. Universal Data Function
   - D. Unique Data File
8. **Which Hive data type is used to store date and time information?**
   - A. DATE
   - B. TIMESTAMP
   - C. DATETIME
   - D. TIME
9. **Which of the following is NOT a complex data type in Hive?**
   - A. ARRAY
   - B. MAP
   - C. STRUCT

- D. STRING

10. **What is the main advantage of using partitions in Hive?**
    - A. To compress data
    - B. To reduce the size of data
    - C. To improve query performance
    - D. To replicate data
11. **What is bucketing in Hive used for?**
    - A. To divide data into equal parts based on hash functions
    - B. To partition data
    - C. To create indexes
    - D. To compress data
12. **Which keyword is used to create a view in Hive?**
    - A. CREATE TABLE
    - B. CREATE VIEW
    - C. CREATE INDEX
    - D. CREATE PARTITION
13. **In Hive, which file format supports columnar storage?**
    - A. TextFile
    - B. SequenceFile
    - C. ORC
    - D. JSON
14. **What does SerDe stand for in Hive?**
    - A. Serialization and Deserialization
    - B. Server Data Engine
    - C. Service Data Execution
    - D. Serialized Data Entity
15. **Which of the following is a valid Hive complex data type?**
    - A. INT
    - B. FLOAT
    - C. STRING
    - D. MAP

16. **Which clause is used to group rows that have the same values in specified columns in Hive?**
    - A. GROUP BY
    - B. ORDER BY
    - C. HAVING
    - D. PARTITION BY
17. **What is the function of the HAVING clause in Hive?**
    - A. To filter records after grouping
    - B. To sort the results
    - C. To join tables
    - D. To partition data
18. **Which statement is used to delete a table in Hive?**
    - A. DROP TABLE
    - B. DELETE TABLE
    - C. REMOVE TABLE
    - D. ERASE TABLE
19. **What is the purpose of using views in Hive?**
    - A. To store data
    - B. To create a virtual table
    - C. To perform data compression
    - D. To manage user access
20. **Which of the following is NOT a join type in Hive?**
    - A. INNER JOIN
    - B. LEFT JOIN
    - C. CROSS JOIN
    - D. VERTICAL JOIN

Unit 5: Pig

1. **What is Pig Latin?**
    - A. A high-level language for data processing

- B. A Latin dialect
- C. A scripting language for web development
- D. A database query language

2. **Which of the following is NOT a feature of Apache Pig?**
   - A. Ease of programming
   - B. Optimization opportunities
   - C. High-level abstraction
   - D. Real-time processing
3. **What is the primary use case for Apache Pig?**
   - A. Real-time analytics
   - B. Batch processing of large datasets
   - C. Transactional processing
   - D. In-memory computing
4. **What does the Pig execution mode 'Local' refer to?**
   - A. Running Pig on a Hadoop cluster
   - B. Running Pig on a single machine
   - C. Running Pig in the cloud
   - D. Running Pig in a virtual environment
5. **Which operator in Pig is used to load data from a file?**
   - A. LOAD
   - B. STORE
   - C. SELECT
   - D. FETCH
6. **What does 'Piggy Bank' refer to in Apache Pig?**
   - A. A built-in data type
   - B. A user-defined function library
   - C. A storage format
   - D. A debugging tool
7. **Which command is used to run a Pig script in MapReduce mode?**
   - A. pig -x local script.pig
   - B. pig -x mapreduce script.pig

- C. pig -mode mapreduce script.pig
- D. pig -mode local script.pig

8. **What are the two main data types in Pig?**
   - A. Primitives and Complex
   - B. Simple and Compound
   - C. Basic and Advanced
   - D. Scalar and Vector
9. **In Pig, which operator is used to group data?**
   - A. GROUP
   - B. JOIN
   - C. UNION
   - D. FILTER
10. **Which of the following is NOT a relational operator in Pig?**
    - A. LOAD
    - B. STORE
    - C. SELECT
    - D. GROUP
11. **What is the purpose of the 'FOREACH' statement in Pig?**
    - A. To iterate over each element in a dataset
    - B. To join two datasets
    - C. To filter data
    - D. To load data
12. **Which execution mode in Pig is used for development and testing on a local machine?**
    - A. MapReduce mode
    - B. Local mode
    - C. Cluster mode
    - D. Distributed mode
13. **What does 'Eval function' in Pig refer to?**
    - A. Evaluation functions that operate on data fields
    - B. Execution functions for running scripts

- C. Error-checking functions
- D. Event-handling functions

14. **Which Pig data type is used to represent a collection of tuples?**
    - A. Bag
    - B. List
    - C. Array
    - D. Map
15. **What is the role of 'UDF' in Pig?**
    - A. To define custom processing logic
    - B. To load data from external sources
    - C. To store processed data
    - D. To optimize query performance
16. **Which command in Pig is used to store the result into a file?**
    - A. LOAD
    - B. STORE
    - C. SAVE
    - D. OUTPUT
17. **In Pig, what does the 'DIFF' operator do?**
    - A. Computes the difference between two datasets
    - B. Finds the common elements in two datasets
    - C. Filters out duplicate records
    - D. Joins two datasets
18. **Which diagnostic operator is used to display data in Pig?**
    - A. DESCRIBE
    - B. ILLUSTRATE
    - C. EXPLAIN
    - D. PRINT
19. **What is a common use case for Pig at Yahoo!?**
    - A. Real-time user data analysis
    - B. Log processing and data analysis
    - C. In-memory computation

- D. Small-scale data processing

20. **What is a key difference between Pig and Hive?**
    - A. Pig is used for real-time processing, Hive for batch processing
    - B. Pig uses a procedural language, Hive uses a declarative language
    - C. Pig is a storage system, Hive is a processing engine
    - D. Pig supports SQL, Hive does not

Answers

Unit 1: Big Data and Analytics

1. A. Data stored in a fixed format
2. C. Versatility
3. B. High storage costs
4. B. A system for reporting and data analysis
5. D. Slow data generation
6. C. Data Scientist
7. A. Low data volume
8. A. Volume
9. B. Big Data Analytics important
10. D. Small-scale data processing
11. B. High storage costs
12. A. Data stored in a fixed format
13. B. A system for reporting and data analysis
14. D. Slow data generation
15. C. Slow data generation

Unit 2: Technology Landscape

1. A. Running Pig on a Hadoop cluster
2. A. LOAD
3. B. Hadoop

4. A. Primitives and Complex
5. A. GROUP
6. B. Running Pig on a single machine
7. B. pig -x mapreduce script.pig
8. C. Simple and Compound
9. A. LOAD
10. C. Basically Available Soft State Eventual Consistency
11. B. Running Pig on a single machine
12. B. Local mode
13. B. Execution functions for running scripts
14. A. To iterate over each element in a dataset
15. B. Real-time user data analysis

Unit 3: MongoDB and MapReduce Programming

1. B. Schema-less data model
2. A. Table
3. B. insert()
4. D. Boolean
5. C. To process input data and produce key-value pairs
6. B. Processes intermediate key-value pairs to produce final output
7. B. To perform local aggregation of intermediate outputs
8. C. Table
9. C. BSON
10. C. $find
11. D. Shuffle and Sort
12. A. Date
13. B. Flexible schema design
14. C. To determine how the data is split among reducers

15. B. update()
16. B. Distributing intermediate data to reducers
17. A. remove()
18. B. A set of documents
19. A. To reduce the amount of data transferred to the reducer
20. C. Binary

Unit 4: Hive

1. D. All of the above
2. B. ORC
3. A. Hive Query Language
4. C. Partitioning
5. C. CREATE
6. A. To provide a columnar storage format
7. A. User-Defined Function
8. B. TIMESTAMP
9. D. STRING
10. C. To improve query performance
11. A. To divide data into equal parts based on hash functions
12. B. CREATE VIEW
13. C. ORC
14. A. Serialization and Deserialization
15. A. INT
16. A. GROUP BY
17. A. To filter records after grouping
18. A. DROP TABLE
19. B. To create a virtual table
20. D. VERTICAL JOIN

Unit 5: Pig

1. A. A high-level language for data processing
2. D. Real-time processing
3. B. Batch processing of large datasets
4. B. Running Pig on a single machine
5. A. LOAD
6. B. A user-defined function library
7. B. pig -x mapreduce script.pig
8. A. Primitives and Complex
9. A. GROUP
10. A. LOAD
11. A. To iterate over each element in a dataset
12. B. Local mode
13. A. Evaluation functions that operate on data fields
14. B. Bag
15. A. To define custom processing logic
16. B. STORE
17. A. Computes the difference between two datasets
18. B. ILLUSTRATE
19. B. Log processing and data analysis
20. B. Pig uses a procedural language, Hive uses a declarative language

Descriptive Questions

Unit 1: Big Data and Analytics

1. What are the main characteristics of structured data, semi-structured data, and unstructured data? Provide examples of each.
2. Explain the concept of Big Data, including its characteristics, evolution, and definition.
3. What are the key challenges associated with Big Data? How do these challenges differ from traditional data processing?
4. Compare and contrast Big Data analytics with traditional business intelligence. What are the main differences between the two approaches?
5. Describe the concept of "Volume, Velocity, Variety, Veracity" (4Vs) in the context of Big Data. How do these factors influence data analysis?
6. Discuss the role of data science and data scientists in the era of Big Data. What skills and expertise are required for a career in data science?
7. Explain the concept of "Basically Available, Soft State, Eventual Consistency" (BASE) in distributed systems. How does it relate to Big Data environments?
8. What are some common terminologies used in Big Data environments, such as Hadoop, MapReduce, and NoSQL? Provide brief explanations for each term.
9. Compare and contrast Big Data analytics tools, such as Hadoop, Spark, and Flink. What are the strengths and limitations of each tool?
10. How does Big Data analytics contribute to decision-making processes in various industries, such as healthcare, finance, and retail?
11. Discuss the importance of data quality and data governance in Big Data analytics. How can organizations ensure the accuracy and reliability of their data?
12. Explain the concept of data integration in the context of Big Data. Why is it important for organizations to integrate data from multiple sources?

13. Describe the concept of data warehousing and its role in Big Data analytics. How does it differ from traditional database systems?
14. Discuss the impact of Big Data on privacy, security, and ethical considerations. What measures can organizations take to protect sensitive data?
15. How do traditional business intelligence approaches differ from advanced analytics techniques used in Big Data environments? Provide examples of each approach.

Unit 2: Technology Landscape

1. What is NoSQL, and how does it differ from traditional relational databases? Provide examples of NoSQL databases and their use cases.
2. Explain the architecture of Hadoop and its core components, including HDFS, MapReduce, and YARN.
3. Describe the concept of distributed computing and its challenges. How does Hadoop address these challenges?
4. Compare and contrast Hadoop with traditional RDBMS systems. What are the advantages and disadvantages of each approach?
5. What is HDFS, and what role does it play in the Hadoop ecosystem? How does it differ from traditional file systems?
6. Explain the role of MapReduce in processing large datasets in a distributed environment. How does it work?
7. What is YARN, and how does it improve resource management in Hadoop clusters? What are its key features?
8. Discuss the interaction between Hadoop and other Big Data technologies, such as Apache Spark, Apache Hive, and Apache Pig.
9. Describe the concept of NoSQL databases and their advantages over traditional relational databases. What are some common types of NoSQL databases?
10. Explain the concept of scalability and fault tolerance in the context of distributed systems. How does Hadoop achieve these properties?

11. Discuss the concept of distributed data processing and its advantages. How does it enable parallel computation?
12. Describe the role of data serialization and deserialization in Hadoop. What are some common serialization formats used in Hadoop?
13. Compare and contrast batch processing and real-time processing approaches in Big Data analytics. What are the use cases for each approach?
14. Discuss the challenges of managing Big Data environments, including data storage, data processing, and resource allocation.
15. Explain the concept of data locality in Hadoop. How does it impact the performance of data processing jobs?

Unit 3: MongoDB and MapReduce Programming

1. Describe the key features and advantages of MongoDB as a NoSQL database.
2. Explain the structure of a MongoDB document and provide examples of document-oriented data.
3. Discuss the differences between relational databases and MongoDB in terms of data modeling and schema design.
4. Explain the MapReduce programming model and its role in processing large datasets.
5. Describe the basic components of a MapReduce job, including Mapper, Reducer, and Partitioner.
6. Provide an overview of the MongoDB Query Language (MQL) and its syntax for querying and manipulating data.
7. Discuss the use cases for MongoDB and its suitability for various types of applications.
8. Explain the concept of sharding in MongoDB and how it helps to distribute data across multiple servers.
9. Discuss the advantages and limitations of MongoDB compared to traditional relational databases.

10. Describe the process of MapReduce job execution and how it distributes tasks across a cluster of nodes.
11. Explain the role of indexes in MongoDB and how they improve query performance.
12. Discuss the use of aggregation pipelines in MongoDB for complex data processing tasks.
13. Explain the concept of replication in MongoDB and how it ensures data availability and fault tolerance.
14. Describe the process of data backup and recovery in MongoDB.
15. Discuss best practices for designing schema and data models in MongoDB to optimize performance and scalability.

Unit 4: Hive

1. Explain the architecture of Apache Hive and its components.
2. Describe the supported data types in Hive and their usage in table definitions.
3. Discuss the different file formats supported by Hive and their advantages and disadvantages.
4. Explain the concept of partitioning in Hive and how it improves query performance.
5. Describe the role of bucketing in Hive and its use cases.
6. Provide an overview of Hive Query Language (HQL) and its syntax for querying data.
7. Discuss the implementation details of RCFile format in Hive and its benefits.
8. Explain the concept of User-Defined Functions (UDFs) in Hive and how they extend its functionality.
9. Discuss the importance of serialization and deserialization in Hive and its impact on data processing.
10. Describe the process of creating views in Hive and their use cases.
11. Explain the concept of joins in Hive and different types of join operations supported.

12. Discuss the use of aggregations in Hive and its syntax for group by and having clauses.
13. Describe the implementation details of Hive partitions and how they are managed.
14. Discuss the integration of Hive with other Hadoop ecosystem components such as HDFS and YARN.
15. Explain the process of optimizing Hive queries for performance and efficiency.

Unit 5: Pig

1. Describe the features and advantages of Apache Pig for data processing.
2. Explain the Pig Latin scripting language and its syntax for defining data processing workflows.
3. Discuss the different execution modes of Apache Pig and their use cases.
4. Describe the Pig data types and their usage in Pig Latin scripts.
5. Explain the role of relational operators in Pig and their syntax for data manipulation.
6. Discuss the use of user-defined functions (UDFs) in Pig and how they extend its functionality.
7. Describe the process of loading data into Pig from external sources and storing results back to the filesystem.
8. Explain the use of built-in functions and operators in Pig for common data processing tasks.
9. Discuss the concept of data streaming in Pig and how it enables real-time processing.
10. Describe the Piggybank library and its collection of user-contributed functions and utilities.
11. Explain the diagnostic operators in Pig and how they help in debugging and troubleshooting scripts.
12. Discuss the process of parameter substitution in Pig and its use for dynamic script configuration.

13. Describe the process of running Pig scripts in local mode and distributed mode.
14. Explain the use of Pig macros for code reuse and modularization.
15. Discuss the integration of Pig with other components of the Hadoop ecosystem such as HDFS and MapReduce.

16.