Introduction to Data Analytics, Big Data, Hadoop and Spark

Lesson 3

Disclaimer

These study notes summarize key concepts from Cloud Computing, Theory and Practice, Third Edition by ***Dan C. Marinescu, Department of Computer Science, University of Central Florida, Orlando, FL, United States****. They are for educational and informational use only and do not replace or reproduce the original book.*

No copyright infringement is intended, and these notes are created under the principles of fair use. The author is not affiliated with or endorsed by the original authors or publishers. If there are any concerns regarding this content, please contact me for resolution at ***vivek.bhadra@gmail.com***

Understanding Analytics

- Analytics involves the extensive use of data, statistical models, and quantitative analysis to drive decisions and actions.
- It includes explanatory and predictive modeling techniques to uncover insights that improve business strategies.

Purpose of Analytics

Analytics helps organizations by:

✔ **Uncovering Hidden Patterns**

- Identifying trends and anomalies in data
- Example: *2 out of 100 stores had no sales for a promotion item because it was not on the right shelf*

Understanding Analytics

✔ **Deciphering Unknown Correlations**

- Revealing unexpected relationships between variables
- Example: *The famous "Beer and Diapers" story – retailers discovered a purchasing pattern where men buying diapers often also bought beer*

✔ **Understanding Trends**

- Analyzing customer preferences and market behaviors
- Example: *What do users like about a popular product that has growing sales?*

✔ **Extracting Business Intelligence**

- Identifying key drivers of sales and profitability
- Example: *Popular items during specific holiday sales*

Benefits of Analytics in Business

Analytics enhances various aspects of business operations:

✔ **Effective Marketing**

- Targeted advertising and personalized recommendations

✔ **Better Customer Service & Satisfaction**

- Predicting customer needs and improving engagement

✔ **Improved Operational Efficiency**

- Optimizing inventory management and reducing waste

✔ **Competitive Advantage**

- Staying ahead of competitors by making data-driven decisions

"Beer and Diapers" story

One evening in the 1990s, a supermarket’s data analysts stumbled upon something unexpected. As they combed through sales data, they noticed a strange pattern—men, particularly young fathers, were buying **diapers and beer together**.

At first, it made no sense. Why would two completely unrelated products appear in the same shopping cart so often? But then it clicked. **These were new dads, sent out on late-night diaper runs, and while they were at it, they’d grab a pack of beer for themselves**.

The store managers saw an opportunity. Instead of keeping beer and diapers in their usual separate aisles, they placed them closer together. The result? Sales for both shot up. More dads spotted the beer while grabbing diapers and made an impulse purchase.

This became one of the most famous stories in retail analytics, proving that **data doesn’t just confirm what you already know—it reveals what you never even thought to ask**.

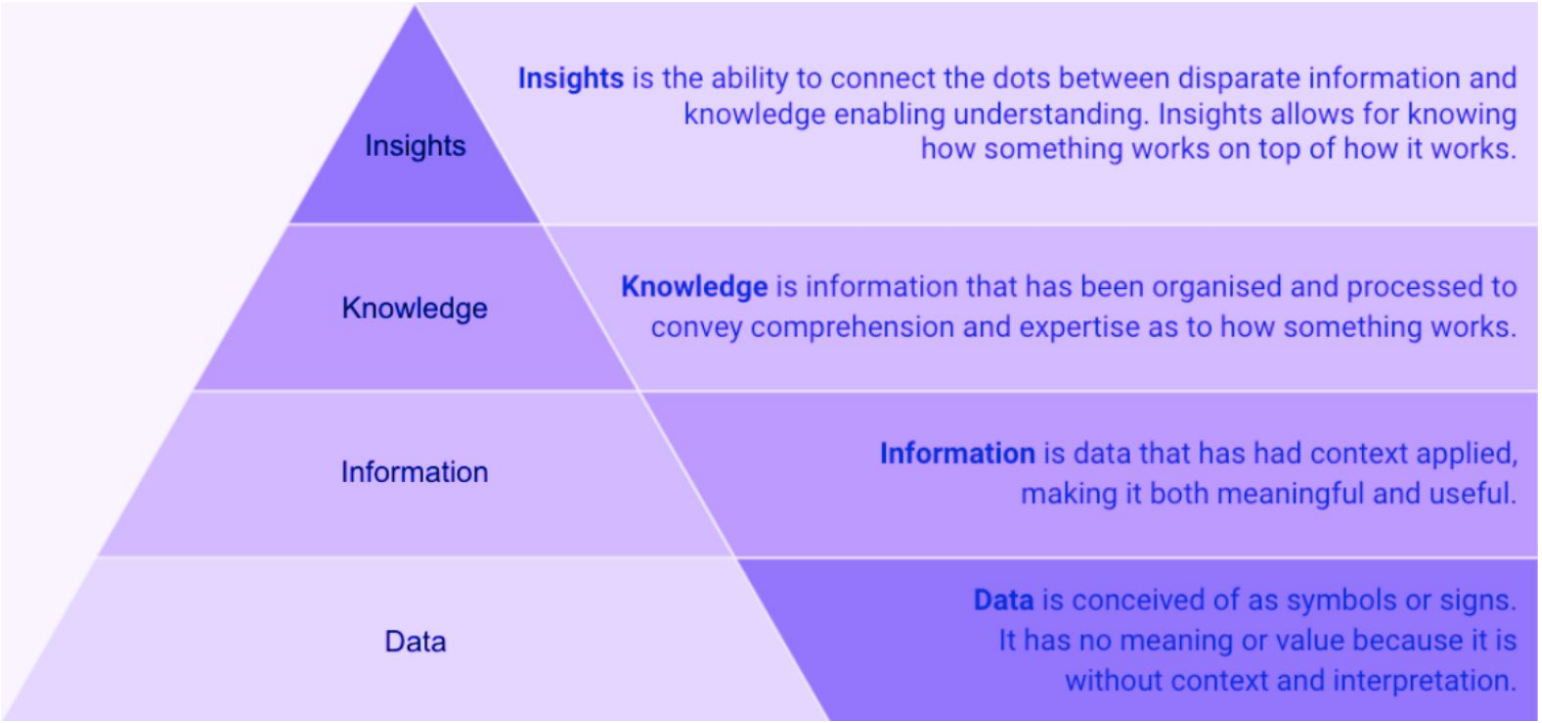
Process of Analysis: Understanding Data Transformation

Data → Information → Knowledge → Wisdom (DIKW)

Data analysis involves a structured process where raw data is gradually refined into meaningful insights that drive business decisions. The process follows these key stages:

1. **Raw Data**
   - The foundation of any analysis. Includes **sales transactions, customer interactions etc.**
2. **Information**
   - Information is data that has been processed and given context, making it understandable.
   - Example: **Aggregated weekly sales data** instead of individual transactions
3. **Knowledge**
   - Knowledge is information that has been further processed and analyzed to explain how something works.
   - Example: Adding **promotional events, pricing changes, social media trends** to explain sales variations
4. **Actionable Insights (Wisdom)**
   - Final refined stage where knowledge is synthesized into strategic decisions
   - Example: **Identifying which promotions work best or predicting future demand**

Process of Analysis: Understanding Data Transformation



Key Stages of Data Transformation

1. **Collecting** → First, raw data is gathered from various sources.
2. **Organizing** → Next, the data is structured and formatted for efficient access.
3. **Summarizing** → Key trends and metrics are identified from the structured data.
4. **Analyzing** → Patterns and relationships are discovered using statistical or computational techniques.
5. **Synthesizing** → Finally, insights from the analysis are integrated for informed decision-making.

Importance of Context & Metadata

Raw data alone lacks meaning and can lead to misinterpretation without **context**. **Metadata** helps explain influencing factors such as **time periods, locations, market conditions, and customer segments**, making analysis more accurate.

✔ **Context Avoids Misinterpretation:** A sudden drop in sales might seem alarming, but metadata (e.g., a seasonal dip or new competitor entry) provides clarity.
✔ **Metadata Enhances Analysis:** Adding details like time, location, or market conditions improves data accuracy and relevance.
✔ **Informed Decision-Making:** Businesses use metadata to ensure insights are correctly interpreted, avoiding misleading conclusions.

Example: A winter clothing brand sees a drop in sales in June. Without **seasonality as context**, the decline might be misread as a loss of popularity.

Context and metadata turn raw data into actionable insights, ensuring data-driven decisions are accurate and meaningful.

Analytics Maturity Model

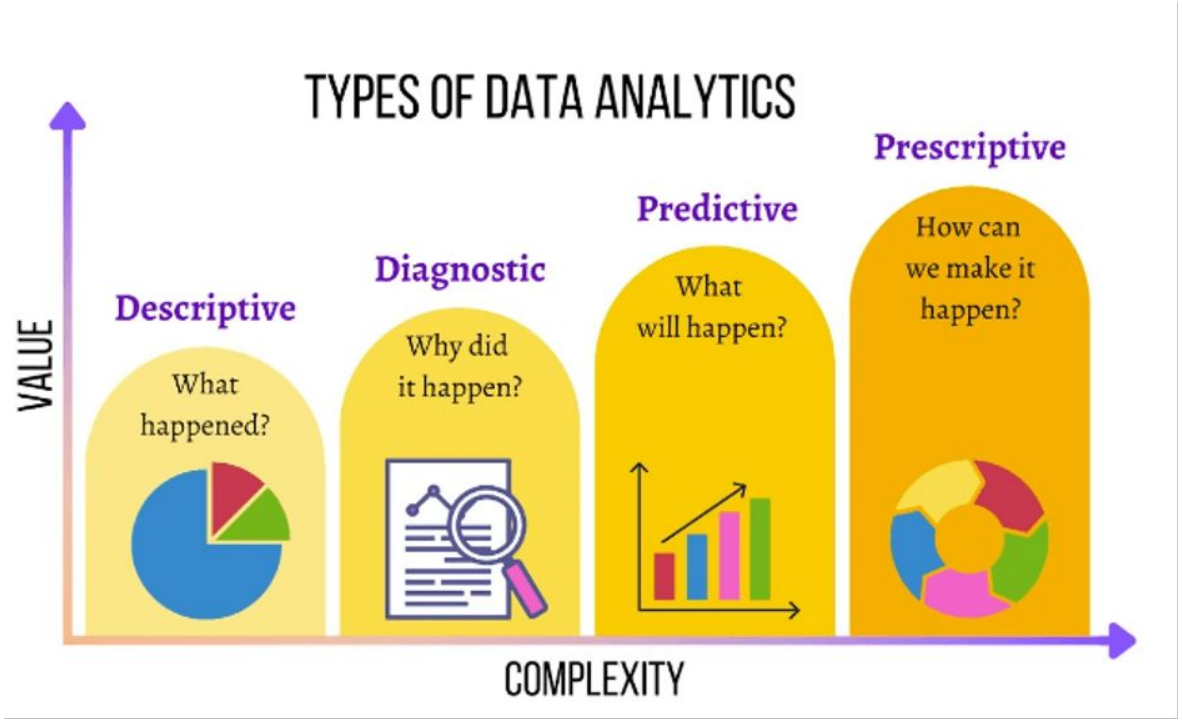
Data	Inconsistent, poor quality	Usable in functional silos	Beginning to create centralized repositories	Integrated data warehouse	Relentless search for new data & metrics
Enterprise	NA	Siloes of data, tech. and talent	Early stages of enterprise-wide approach	Key data, tech. and analysts are centralized	All key analytical resources centrally managed
Leadership	No awareness or interest	Only at functional or process level	Begins to recognize importance of analytics	Support for analytical component	Strong passion for analytics
Targets	NA	disconnected	Efforts aligned to small set of targets	Centered on few key domains	Supports overall strategic objectives
Analysts	Few skills in specific functions	Few isolated analysts	Influx of analysts in key target areas	Specialized analysts in central organization	Mix of analytics experts and amateurs

source: Competing on Analytics, Thomas Davenport

Types of Analytics

- **Descriptive** (*What we have done in the past?*)
  - Descriptive analytics focuses on summarizing historical data to understand past events. It provides insights into trends, patterns, and key business metrics.
- **Diagnostic** (*Why we see past results?*)
  - Diagnostic analysis is a type of **descriptive analytics** that focuses on understanding **why** past events occurred by identifying patterns, correlations, and relationships in data.
- **Predictive** (*Where we are going and when?*)
  - Predictive analytics uses historical data, statistical models, and machine learning techniques to forecast future trends and events.
- **Prescriptive** (*What Actions Should Be Taken?*)
  - Prescriptive analytics goes beyond predicting outcomes—it suggests the best actions to take based on predictions and optimization models.

Types of Analytics



Introduction to Hadoop & Big Data

What is Big Data?

Big Data refers to **large, complex datasets** that traditional data processing systems cannot efficiently handle. These datasets exhibit **high volume, velocity, and variety**.

Why Hadoop?

- Distributed systems have gained popularity for their efficiency in processing big data. Several approaches exist for processing large volumes of data by leveraging a compute cluster.
- Hadoop uses a divide-and-conquer paradigm.
- It breaks big problems into smaller ones, distributes the tasks across nodes, and later aggregates individual results to form the final output.

Big Data Analytics

Big Data Analytics is about handling and making sense of **large, complex, and fast-moving data** that traditional databases can't process efficiently. It helps businesses **analyze trends, improve operations, and make better decisions** using advanced computing techniques.

Characteristics

- **Works with Large and Fast Data** – Businesses generate massive data every second (e.g., social media, transactions, sensors). Big Data tools handle this scale efficiently.
- **Goes Beyond Traditional Databases** – Standard databases (RDBMS) struggle with huge unstructured datasets, but technologies like **Hadoop, Spark, and cloud computing** solve this challenge.
- **Speeds Up Decision-Making** – Instead of waiting for reports, companies can analyze **real-time data** and react quickly (e.g., optimizing supply chains).

Big Data Analytics

Characteristics

- **Moves Computation Closer to Data** – The **principle of locality** ensures that analysis happens where the data is stored, avoiding slow data transfers.
- **Handles Both Past and Live Data** – It supports **batch processing** (analyzing past records) and **stream processing** (real-time data insights), depending on business needs.
- **Bridges IT and Business Teams** – Data scientists, IT teams, and business leaders work together to extract meaningful insights and drive strategy.
- **Gives a Competitive Edge** – Companies that use Big Data **better understand customer behavior, predict trends, and optimize performance** ahead of competitors.

Challenges in Adopting Big Data in Organizations

1. **Lack of Executive Support**

- Securing funding and leadership backing for Big Data initiatives is a major hurdle.
- Many executives are hesitant due to **unclear ROI (Return on Investment)** and high implementation costs.

2. Data Silos and Resistance to Sharing

- Different departments often **store and control their own data**, making collaboration difficult.
- Business units may **hesitate to share data** due to security concerns or lack of trust.

3. Shortage of Skilled Professionals

- Finding **qualified data analysts, data scientists, and engineers** who can handle massive datasets is challenging.
- The demand for skilled professionals often exceeds supply, leading to **high hiring costs**.

Challenges in Adopting Big Data in Organizations

4. Scalability and Storage Issues

- As data grows in **volume, velocity, and variety**, organizations struggle to scale storage and processing efficiently.
- Choosing between **on-premise vs. cloud storage** and **real-time vs. batch processing** can be complex.

5. Structured vs. Unstructured Data

- Businesses must decide how to **combine structured data (databases, spreadsheets) with unstructured data (social media, images, emails)** for analysis.
- **Internal vs. external data** sources add another layer of complexity.

Challenges in Adopting Big Data in Organizations

6. Effective Reporting and Visualization

- Collecting data is not enough—organizations must determine the **best way to present and interpret findings**.
- Poor data visualization can make insights difficult for decision-makers to understand.

7. Translating Insights into Action

- Even when data-driven insights are available, companies often struggle to **convert them into concrete business strategies**.
- Without a clear action plan, Big Data investments may not deliver expected results.

Introduction to Hadoop

Overview of Hadoop

- Hadoop is an **open-source, Java-based software framework** for distributed storage and processing of large data sets.
- It is widely used in **marketing analytics, machine learning, image processing, and web crawling**.
- Hadoop enables **scalable, fault-tolerant data processing** by distributing workloads across multiple machines.

Introduction to Hadoop

Major Users of Hadoop

- **IT Companies:** Apple, IBM, HP, Microsoft, Yahoo, Amazon.
- **Media Companies:** *New York Times*, Fox.
- **Social Networks:** Twitter, Facebook, LinkedIn.
- **Government Agencies:** Federal Reserve.

Real-World Use Case:

- In 2012, Facebook's Hadoop cluster had **100 petabytes of data** and was growing at **0.5 petabytes/day**.
- By 2013, over **half of Fortune 500 companies** were using Hadoop.

Introduction to Hadoop

What does Hadoop offer?

Hadoop was developed to **solve Big Data challenges** by providing:

- **Distributed Storage** Hadoop Distributed File System (**HDFS**), which is designed to store large datasets across multiple machines in a fault-tolerant and scalable manner.
- **Parallel Processing** via MapReduce, a distributed computing model that processes large datasets by dividing tasks into smaller sub-tasks executed across multiple nodes in parallel.
- **Scalability** which ensures that a system can efficiently handle increased workloads by adding more resources without performance degradation
- **Fault Tolerance** which enables a system to continue functioning correctly even when one or more components fail.

Hadoop Architecture

HDFS (Hadoop Distributed File System)

- **Stores data across multiple nodes**
- **Replication Factor** ensures fault tolerance
- **Blocks**: Large files are divided into **64MB/128MB blocks**

YARN (Yet Another Resource Negotiator)

- **Job Scheduling** and **Resource Allocation**
- Enables multiple applications to share cluster resources efficiently

MapReduce Processing Framework

- **Map Phase**: Processes input data into key-value pairs
- **Shuffle & Sort Phase**: Organizes data for processing
- **Reduce Phase**: Aggregates results

Hadoop Architecture

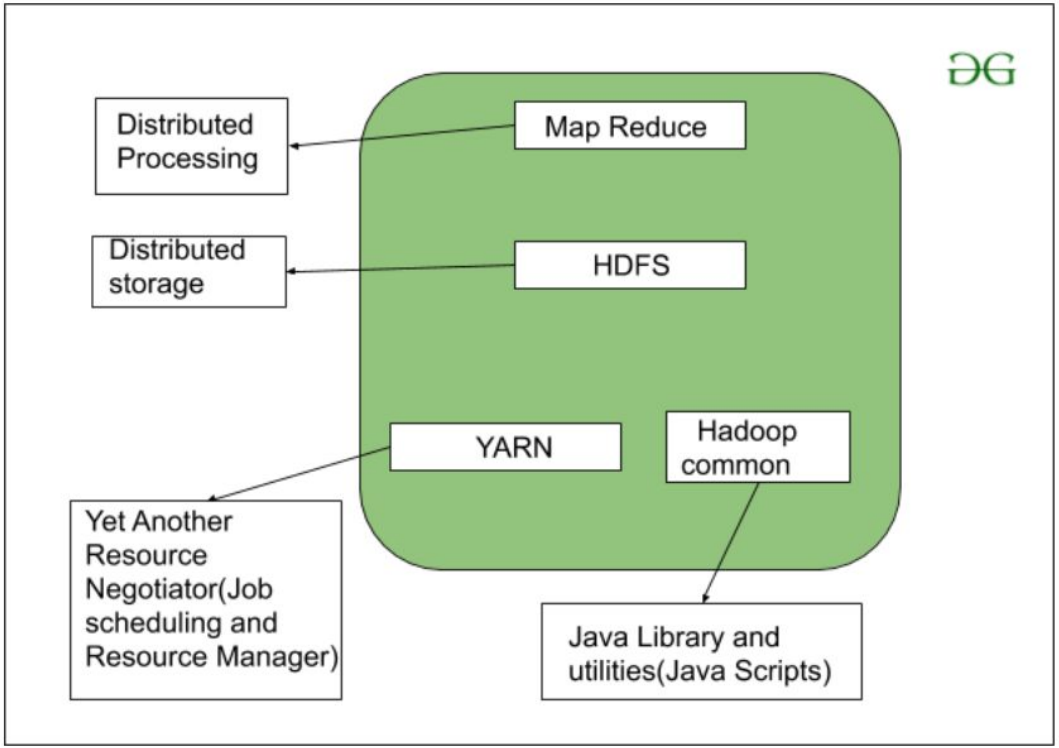


Image Source: GeeksforGeeks, *Hadoop Architecture*, retrieved from GeeksforGeeks

HDFS (Hadoop Distributed File System)

Distributed Storage:

- Large files are divided into **blocks** (default size: 64MB or 128MB).
- These blocks are stored **across multiple nodes** in a Hadoop cluster.

Replication for Fault Tolerance:

- Each block is **replicated across multiple nodes** (default: 3 copies).
- If one node fails, data is still accessible from other replicas.

HDFS (Hadoop Distributed File System)

Scalability:

- New nodes can be **added dynamically** to handle more data.
- No need to modify existing configurations.

Write-Once, Read-Many Model:

- Files in HDFS are typically **written once and read multiple times**, making it ideal for analytical workloads.

NameNode & DataNodes Architecture:

- **NameNode**: Stores metadata (file locations, block mappings).
- **DataNodes**: Store actual file blocks and handle read/write operations.

How data is processed in Hadoop?

- A Hadoop system has two components, a *MapReduce* engine and a database (Hadoop Distributed File System or HDFS)
- HDFS is a highly performant distributed file system written in Java.
- The Hadoop engine on the master of a multi-node cluster consists of a *job tracker* and a *task tracker*
- The Hadoop engine on a slave has **only a task tracker**.
- The **job tracker** on the master node receives a MapReduce job from a client (user of the Hadoop System) and dispatches the work to the task trackers (slave) running on the nodes of a cluster.
- To increase efficiency, the job tracker attempts to dispatch the tasks to the available slaves closest to the place where the task data was stored.
- The **task tracker** supervises the execution of the work allocated to the node.
- Several scheduling algorithms have been implemented in Hadoop engines, including Facebook's fair scheduler and Yahoo's capacity schedulers.

How data is processed in Hadoop?

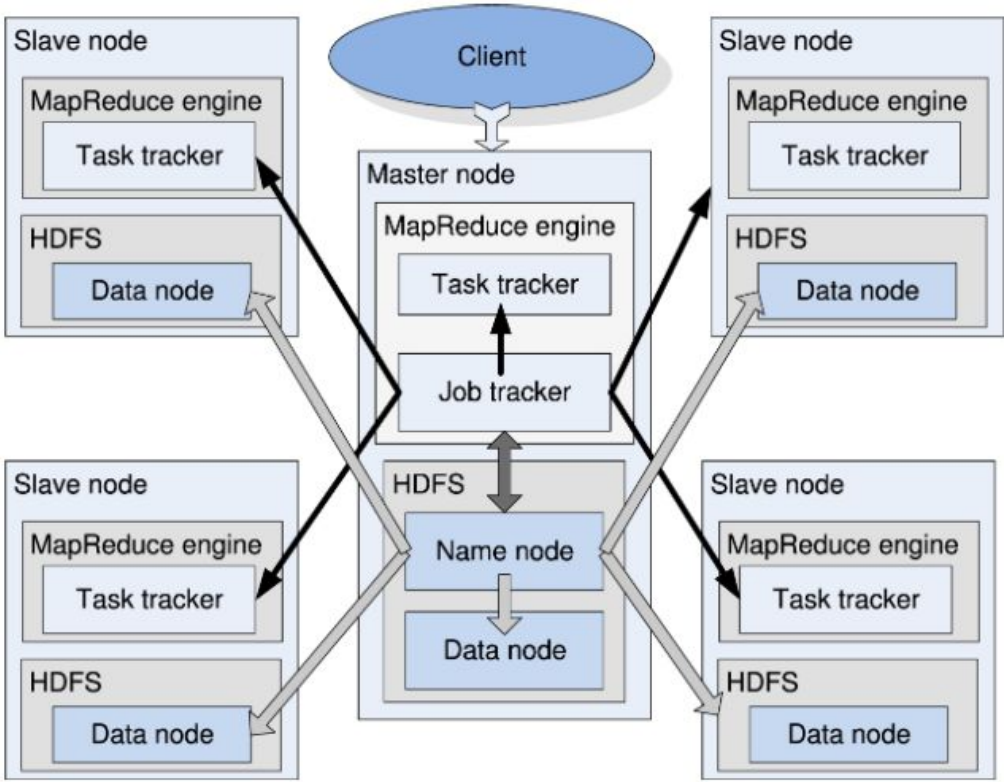
In this example, a **Hadoop cluster** using **HDFS** consists of a **master node** and **four slave nodes**.

Each node runs a **MapReduce engine** and **HDFS**.

The **job tracker** on the **master node** communicates with:

- **Task trackers** on all nodes.
- The **name node** of **HDFS**.

The **name node** shares **data placement** details with the **job tracker** to reduce communication overhead between nodes storing and processing data.



How HDFS achieves Fault Tolerance?

- HDFS achieves fault tolerance through **data replication**. When a file is stored in HDFS it is divided into **blocks** (default size: 128MB) and each block is **replicated** across multiple nodes in the cluster.
- The **replication factor** determines how many copies are maintained (default **3**).

What Happens During a Node Failure?

- If a **DataNode** (which stores file blocks) fails, the **NameNode** detects the missing blocks and automatically **recreates** them on another available node.
- This ensures that the **data remains accessible** without interruption.
- Assume a file is split into **Block A**, **Block B**, and **Block C**, with a **replication factor of 3**.
  - HDFS distributes these blocks across multiple nodes:
  - Node 1 stores: A1, B1, C1
  - Node 2 stores: A2, B2, C2
  - Node 3 stores: A3, B3, C3
- If **Node 1 fails**, the system still has **copies on Node 2 and Node 3**, ensuring no data loss.

MapReduce

MapReduce in Hadoop is a well-organized **strategy** for processing huge amounts of data in a distributed system. Hadoop follows a **Master-Slave** architecture. The **Master node** manages the job and assigns tasks to multiple **Slave nodes**. Here's how each MapReduce stage fits into this structure:

1. **Mapping Stage (Happens on the Slave Nodes)**
    - The **JobTracker (Master Node)** assigns tasks to multiple **TaskTrackers (Slave Nodes)**.
    - Each **Slave node** (TaskTracker) runs a **Map task** on a portion of the input data.
    - The data is split into chunks, and each **Mapper** processes its assigned data in parallel.
    - **Output:** Key-value pairs like `("word", 1)`, `("another", 1)`.

MapReduce

2. Sorting Stage (Happens on the Slave Nodes)

- Each **TaskTracker** locally sorts the output from the **Map stage** by key.
- All `"apple"` keys are grouped together, `"banana"` keys together, and so on.
- This step is still handled at the **Slave nodes**, preparing for the next stage.

3. Combiner Stage (Happens on the Slave Nodes, if enabled)

- If a **Combiner function** is used, it **aggregates local results** on each **Slave node** before sending data over the network.
- Example: If `"apple"` appears five times in the same block, the **Combiner** on the **Slave node** sums it up before sending (`"apple"` → 5 instead of five separate `"apple"` → `1"` entries).

MapReduce

4. Shuffling Stage (Happens Between Slave and Master Nodes)

- The sorted **key-value pairs** are sent to the appropriate **Reducer node**.
- This data **shuffling** is managed by **HDFS (Hadoop Distributed File System)** and coordinated by the **JobTracker** (Master Node).
- Data is transferred from **TaskTrackers (Slaves) to the Reducer Node**.

5. Reduce Stage (Happens on the Slave Nodes)

- The **TaskTracker** running a **Reducer** on a **Slave node** processes all data for a given key.
- It sums up, aggregates, or performs other computations.
- **Final output** is stored back into HDFS.

Map Function in MapReduce

In **Hadoop MapReduce**, the **Map function** is the first phase of processing that takes an **input dataset** and transforms it into **intermediate key-value pairs**. This phase is executed **in parallel** across multiple nodes, improving scalability and efficiency.

How the Map Function Works

1. **Reads Input Data**
   - The dataset (e.g., a large text file) is split into smaller **input splits**.
   - Each split is processed by a separate instance of the **Map function**.
2. **Processes Data**
   - The function **tokeniszs the text** (breaks it into words).
   - It **assigns a key-value pair** for each word:
     - **Key:** The word itself.
     - **Value:** The number **1** (indicating one occurrence of the word).

Example python Map function

```
#!/usr/bin/env python3

import sys

# Read input line by line

for line in sys.stdin:

    words = line.strip().split()

    for word in words:

        print(f"{word}\t1")  # Output: word 1
```

https://github.com/vivekbhadra/AWS/blob/main/BigData/mapper.py

Example python Reduce function

```
#!/usr/bin/env python3
import sys
current_word = None
current_count = 0for line in sys.stdin:
    word, count = line.strip().split("\t")
    count = int(count)
    if word == current_word:
        current_count += count
else:
        if current_word:

print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count
# Print last word count
if current_word:
    print(f"{current_word}\t{current_count}")
```

https://github.com/vivekbhadra/AWS/blob/main/BigData/reducer.py

Setup Hadoop Cluster in AWS : Step by Step Guide

https://techfortalk.co.uk/2025/02/24/set-up-a-hadoop-cluster-on-aws-emr-a-step-by-step-guide/

Hadoop Ecosystem

HDFS – stores structured and unstructured data.

YARN – Manages cluster resources and job scheduling

Data Ingestion:

- **Flume** – Handles unstructured data (e.g., social media, logs).
- **Sqoop** – Transfers structured data between Hadoop and relational databases.

Processing & Analytics:

- **MapReduce** – Processes data using different programming languages.
- **Apache Spark** – In-memory data processing for faster computation.
- **Hive & Drill** – SQL-based querying on Hadoop.
- **Pig** – Uses scripting for data transformation.

Streaming & Indexing:

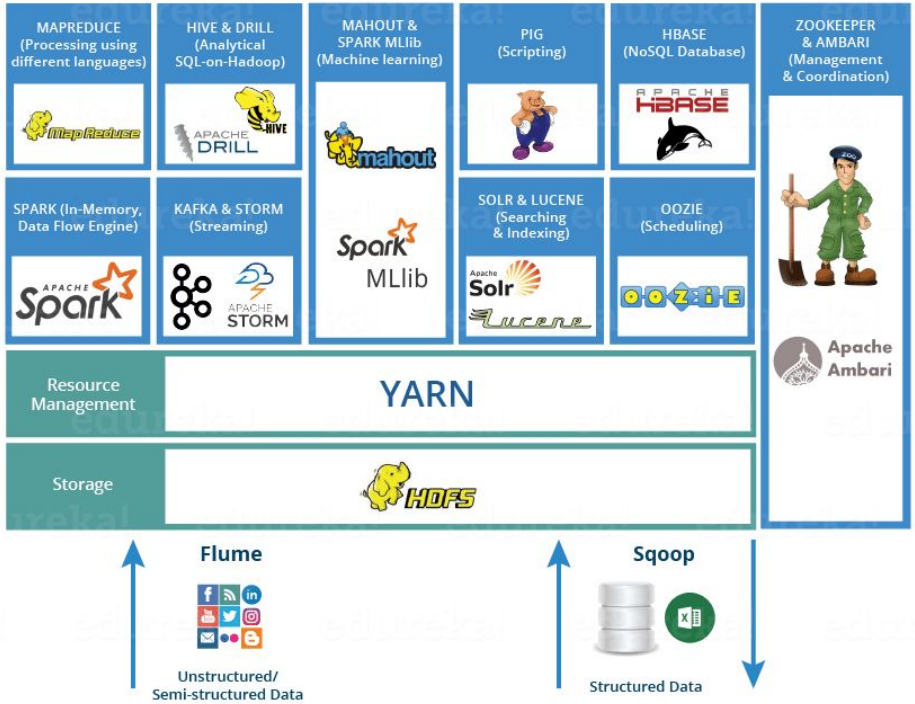
- **Kafka & Storm** – Real-time data processing and event streaming.
- **Solr & Lucene** – Search and indexing functionalities.

Database & Scheduling:

- **HBase** – NoSQL database for real-time read/write access.
- **Oozie** – Job scheduling and workflow automation.

Management & Coordination:

- **Zookeeper & Ambari** – Manage and monitor Hadoop clusters.



Issues with MapReduce on Hadoop

Slow Processing for Low-Latency Applications

- MapReduce relies on **disk-based storage**, which makes it much slower compared to in-memory frameworks like **Apache Spark**.
- It is **not suitable for real-time data processing** where quick responses are required (e.g., fraud detection, live streaming analytics).

Inefficient for Iterative Computations

- Many applications, like **machine learning and graph processing**, require multiple iterations over data.
- Since MapReduce **writes intermediate results to disk after each step**, it slows down such tasks significantly.
- Alternative frameworks (e.g., **Apache Spark**) keep data in **memory**, reducing delays.

Issues with MapReduce on Hadoop

One-Pass Computation Model

- MapReduce processes data in a **fixed sequence (Map → Shuffle → Reduce)**, making it inflexible for complex analytical workflows.
- Advanced use cases often require **multiple stages of computation**, making MapReduce inefficient.

High Development Complexity

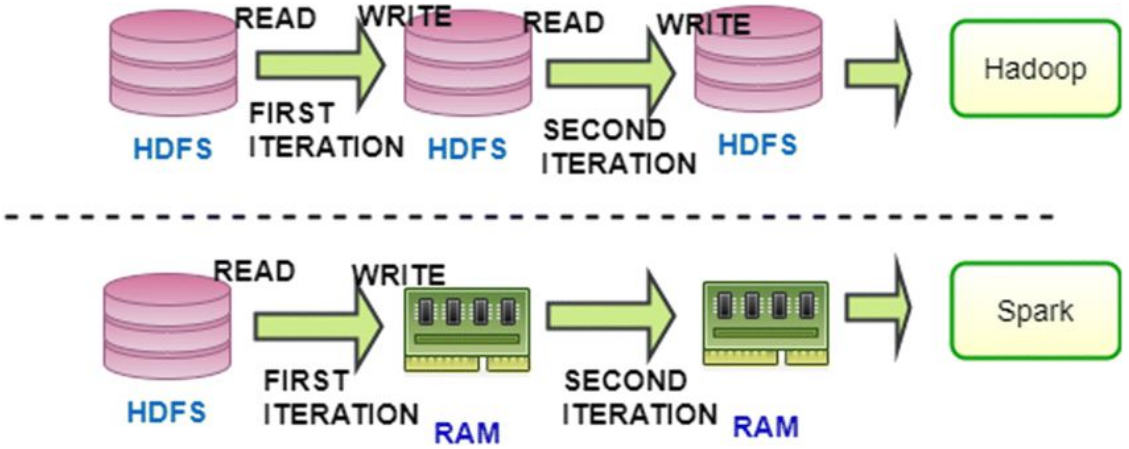
- Writing efficient MapReduce programs requires **manual coding and optimization**, which can be complex.
- SQL-based alternatives like **Hive** simplify querying but still suffer from the same performance bottlenecks.

In-Memory Computing – A Faster Approach

What is In-Memory Computing?

In-memory computing is a method of processing data where **datasets are stored in RAM (Random Access Memory) instead of traditional disk storage**.

This allows for significantly faster computations by enabling **parallel processing across multiple computers** in a cluster.



Fast and Efficient Big Data Processing with Apache Spark

Apache Spark is a powerful **big data processing framework** designed for **speed, scalability, and flexibility**. Unlike traditional MapReduce, Spark performs computations in **memory**, avoiding repeated disk read/write operations, which significantly enhances performance.

Core Operators in Spark's Programming Model:

- **Mappers (Transformations on Data Elements)**
  - Mappers apply a **function to each element** in a dataset and return a new transformed dataset.
  - Used for **data cleaning, conversion, or feature extraction** in machine learning.
  - Example: Converting a dataset of product prices from USD to EUR by applying a function to each price.

Fast and Efficient Big Data Processing with Apache Spark

Core Operators in Spark's Programming Model:

- **Reducers (Aggregation and Summarization)**
  - Reducers combine data **based on a key** and apply an aggregation function (sum, count, max, etc.).
  - Commonly used for computing **totals, averages, and summaries** across large datasets.
  - Example: Counting the number of sales for each product category.

Fast and Efficient Big Data Processing with Apache Spark

Core Operators in Spark's Programming Model:

- **Joins (Combining Datasets on Common Keys)**
  - Joins bring together **multiple datasets** by linking them on a **shared key** (similar to SQL joins).
  - Helps in integrating data from **different sources**, such as customer data and transaction logs.
  - Example: Joining a **customer dataset** with a **purchase history dataset** using the customer ID as the key.

Fast and Efficient Big Data Processing with Apache Spark

Core Operators in Spark's Programming Model:

- **Group-bys (Efficient Aggregation on Groups of Data)**
  - Organizes data **into groups** based on a particular column and applies an operation to each group.
  - Used in **trend analysis, sales aggregation, and customer segmentation**.
  - Example: Grouping **sales by region** and calculating the total revenue for each region.

Fast and Efficient Big Data Processing with Apache Spark

Core Operators in Spark's Programming Model:

- **Filters (Extracting Data Based on Conditions)**
  - Filters **remove unwanted data** by applying a condition.
  - Essential for **data preprocessing, refining datasets, and removing anomalies**.
  - Example: Filtering out transactions **below $10** to analyze only high-value purchases.

Why is Spark Faster Than Hadoop?

In-Memory Computation

- Unlike **Hadoop MapReduce**, which writes intermediate results to disk, Spark keeps data in **RAM**, drastically reducing I/O overhead.
- This makes Spark up to **100 times faster** for certain workloads.

Resilient Distributed Datasets (RDDs)

- Spark introduces **RDDs**, a fault-tolerant abstraction that efficiently stores intermediate computation results.
- This allows developers to **cache datasets** and reuse them across multiple transformations.

Efficient DAG Execution

- Spark uses a **Directed Acyclic Graph (DAG) execution model**, optimizing the order of execution to **minimize redundant computations**.

Spark Word Count Example (PySpark Implementation)

```
from pyspark import SparkContext

# Initialize Spark Context

sc = SparkContext("local", "WordCount")

# Read input file from HDFS or local file system

text_file = sc.textFile("hdfs://path/to/input.txt")

# Step 1: Split lines into words

words = text_file.flatMap(lambda line: line.split(" "))

# Step 2: Convert words into (word, 1) key-value pairs

word_pairs = words.map(lambda word: (word, 1))

# Step 3: Reduce by key to count word occurrences

word_counts = word_pairs.reduceByKey(lambda a, b: a
+ b)

# Step 4: Save the result to HDFS or local storage

word_counts.saveAsTextFile("hdfs://path/to/output")

# Stop Spark Context

sc.stop()
```

Ideal Apache Spark Applications

Low-Latency Computations: Real-Time Processing at Memory Speed

- One of Spark's biggest advantages is its ability to **cache datasets in memory**, eliminating repeated disk reads and writes. This allows it to perform computations at near **memory speeds**, making it highly suitable for applications that require **real-time or near-real-time data analysis**.
- For example, in a **big data analytics system**, multiple users might need to query the same dataset repeatedly. If this dataset is cached in memory, Spark can **serve multiple queries instantly** without having to reload the data from storage each time. This drastically reduces query response times

Ideal Apache Spark Applications

Spark is an excellent choice for:

- **Real-time financial fraud detection** – Analyzing millions of transactions per second to detect anomalies.
- **Stock market analysis** – Running queries on live stock data to make high-frequency trading decisions.
- **Streaming analytics** – Processing live data from IoT sensors, social media feeds, and network logs.

Ideal Apache Spark Applications

Efficient Iterative Algorithms: Essential for Machine Learning & AI

- Many machine learning and AI algorithms require **multiple passes over the same dataset**. Traditional systems like Hadoop **reload data from disk for each iteration**, making them incredibly slow for such workloads. Spark, however, **stores intermediate results in memory**, enabling efficient iterative processing.
- For example, in **training a machine learning model**, the algorithm may need to repeatedly refine its parameters based on the data. If Spark **keeps the dataset in memory**, it avoids unnecessary disk I/O, making iterations significantly faster.

Ideal Apache Spark Applications

This feature makes Spark suitable for:

- **Deep Learning & AI** – Training large-scale neural networks efficiently.
- **Graph Processing (PageRank, Social Network Analysis)** – Analyzing relationships in networks like LinkedIn and Twitter.
- **Recommendation Systems** – Iteratively improving product recommendations based on user behavior.

With Spark, each **subsequent iteration** shares data through memory rather than reloading it from disk, making it orders of magnitude faster for iterative tasks.

Why Spark Excels in These Applications

Unlike traditional big data tools, Spark is built around **Resilient Distributed Datasets (RDDs)**, which allow it to:

1. **Cache frequently accessed data** in memory for repeated use.
2. **Distribute processing across multiple nodes**, ensuring **parallel execution**.
3. **Optimize execution through DAG (Directed Acyclic Graph) scheduling**, reducing unnecessary computations.

Cloud Computing Basics

These have been covered in the Introduction to CC slides (link below):

- CC Definition
- 3-4-5 rules
- IaaS, PaasS, SaaS

https://techfortalk.co.uk/2025/02/21/introduction-to-cloud-computing/