**Chapter 1: Introduction to Big Data and Hadoop**

**1.1 Big Data Characteristics (The 5 V's)**

**Definition:** The key characteristics that define Big Data challenges.

* **Volume:** The enormous *size* of data. (Example: Petabytes of data from social media posts).
* **Velocity:** The high *speed* at which data is generated and processed. (Example: Real-time data from IoT sensors).
* **Variety:** The different *forms* of data. (Example: Structured tables, semi-structured JSON logs, and unstructured videos/images).
* **Veracity:** The *uncertainty* and quality of data. (Example: Inconsistent or fake user reviews).
* **Value:** The process of extracting *useful insights* from data. (Example: Using customer data for targeted advertising).

**1.2 Traditional vs. Big Data Business Approach**

* **Traditional Approach:** Uses relational databases (SQL) on a single server. Good for structured data and complex transactions. Struggles with Volume, Velocity, and Variety.
* **Big Data Approach:** Uses distributed systems (like Hadoop) and NoSQL databases. Designed to handle the 5 V's by scaling out across many servers.

**1.3 Concept of Hadoop & its Ecosystem**

* **Hadoop:** An open-source framework that allows for the distributed processing of large data sets across clusters of computers.
* **Core Components:**
  + **HDFS (Hadoop Distributed File System):** For storage. It splits big files into blocks and distributes them across a cluster.
  + **MapReduce:** For processing. A programming model for processing large datasets in parallel.
* **Hadoop Ecosystem:** Tools built on top of Hadoop (e.g., Hive for SQL-like queries, Pig for data flow, HBase for NoSQL database).

**Chapter 2: Hadoop HDFS and MapReduce**

**2.1 Hadoop Distributed File System (HDFS)**

* **Concept:** A distributed file system designed to store very large files reliably across multiple machines.
* **Key Components:**
  + **NameNode:** The "master" that manages the file system metadata (like the directory tree).
  + **DataNode:** The "slaves" that store the actual data blocks.
* **How it works:** A 1 GB file is broken into 128 MB blocks, and each block is replicated across multiple DataNodes for fault tolerance.

**2.2 MapReduce**

**Definition:** A programming model for processing large datasets in parallel by dividing work into two phases.

* **Map Phase:** Filters and sorts the data. (Input: Key-Value Pair → Output: Intermediate Key-Value Pairs).
  + *Example:* In a word count, the Map task reads a line of text and outputs (word, 1) for each word.
* **Reduce Phase:** Aggregates the results. (Input: Intermediate Key + List of Values → Output: Final Key-Value Pair).
  + *Example:* The Reduce task takes (word, [1,1,1,...]) and outputs (word, 3).
* **Combiners:** A mini-reducer that runs on the same node as the Mapper to reduce the amount of data sent over the network.

**2.3 Algorithms Using MapReduce**

* **Matrix-Vector Multiplication:** Map tasks compute portions of the product, Reduce tasks sum the results.
* **Relational Algebra Operations:**
  + **Selection (σ):** The Map function outputs a tuple if it satisfies the condition; Reduce is often just an identity function.
  + **Projection (π):** The Map function outputs only the required columns.
  + **Union, Intersection, Difference:** Map tasks label tuples based on which relation they come from. Reduce tasks perform the set operation (e.g., for Union, output a tuple if it appears at least once).

**2.4 Hadoop Limitations**

* **Batch Processing:** MapReduce is not designed for real-time stream processing.
* **Latency:** High latency for small jobs due to setup overhead.
* **Complex Programming:** Writing low-level MapReduce code can be complex compared to higher-level tools like Spark.

**Chapter 3: NoSQL**

**3.1 Introduction to NoSQL**

* **Definition:** "Not Only SQL" - databases designed for large-scale data storage and for handling data that doesn't fit neatly into tables.
* **Business Drivers:** Need for agility, scalability to handle massive traffic, and ability to handle unstructured data.

**3.2 NoSQL Data Architecture Patterns**

* **Key-Value Store:** Data is a collection of key-value pairs.
  + *Example:* user\_id\_123 → { "name": "Alice" }. (DB: Redis, DynamoDB).
* **Document Store:** Data is stored as documents (e.g., JSON, XML) with flexible schemas.
  + *Example:* A product document with id, name, price, tags. (DB: MongoDB, CouchDB).
* **Column-Family Store:** Data is stored in columns rather than rows, optimized for queries over large datasets.
  + *Example:* Storing all 'names' together, all 'cities' together. (DB: Cassandra, HBase).
* **Graph Store:** Data is stored as nodes and edges to represent relationships.
  + *Example:* User(Alice) -[LIKES]-> Product(Phone). (DB: Neo4j).

**3.3 NoSQL for Big Data Problems**

* **Shared-Nothing Architecture:** Each node is independent (own CPU, RAM, disk). This allows for easy horizontal scaling by just adding more nodes. (Contrast with traditional shared-disk architectures).
* **Distribution Models:**
  + **Master-Slave:** One master handles writes, slaves handle reads. (Simple, but master is a bottleneck). (Example: MongoDB).
  + **Peer-to-Peer:** All nodes are equal. (Highly available and fault-tolerant). (Example: Cassandra).

**Chapter 4: Mining Data Streams**

**4.1 The Stream Data Model**

* **Concept:** Data that arrives continuously and sequentially, like a never-ending flow.
* **DSMS (Data Stream Management System):** A system for managing and querying continuous data streams, unlike a DBMS which manages static data.
* **Stream Sources:** Examples include sensor data, social media feeds, stock tickers, and server logs.
* **Key Issue:** Cannot store the entire stream, so processing must be done on the fly using limited memory.

**4.2 Sampling Data in a Stream**

* **Why?** To create a manageable, representative subset of the infinite stream.
* **Fixed Sample Size:** Keep only the first n elements. (Risk: may miss later trends).
* **Fixed Proportion Sampling:** Keep every k-th element (e.g., 1% of the stream).

**4.3 Filtering Streams: Bloom Filter**

* **Definition:** A memory-efficient, probabilistic data structure used to test whether an element is a member of a set.
* **Key Property:** Can have **false positives** (says "maybe yes" when the answer is no) but **no false negatives** (if it says "no", it's definitely not in the set).
* **Example:** A web crawler uses a Bloom Filter to check if a URL has already been visited, without storing every single URL.

**4.4 Counting Distinct Elements (Flajolet-Martin Algorithm)**

* **Problem:** Estimate the number of *unique* elements in a massive stream (e.g., unique visitors to a website).
* **Flajolet-Martin Algorithm:** Uses hash functions and the pattern of trailing zeros in binary hashes to estimate distinct count. It's an approximation but uses very little memory.

**4.5 Counting Ones in a Window & DGIM Algorithm**

* **Problem:** Count how many 1s are in the last N bits of a stream (e.g., number of users who clicked an ad in the last hour).
* **DGIM Algorithm:** A smart method to *estimate* the count of 1s in a sliding window using "buckets". It uses logarithmic memory and provides an answer with a guaranteed error bound.

**4.6 Decaying Windows**

* **Concept:** A model where recent data is more important than older data, but the past is not completely forgotten. Older data's contribution "decays" exponentially over time.
* **Example:** Calculating a "trending" topic score, where a tweet from 1 minute ago has a much higher weight than a tweet from 1 hour ago.

**Chapter 5: Real-Time Big Data Models**

**5.1 Recommendation Systems**

* **Content-Based Filtering:** Recommends items similar to what a user liked in the past, based on item features.
  + *Example:* If you liked the movie "Inception", you might be recommended "The Matrix" because both are sci-fi/action.
* **Collaborative Filtering:** Recommends items based on the opinions of other like-minded users.
  + **User-User:** Finds users similar to you and recommends what they liked.
  + **Item-Item:** Finds items similar to the ones you've liked and recommends them.
  + *Example:* On Amazon, "Customers who bought this item also bought...".

**5.2 Social Networks as Graphs**

* **Representation:** A social network is a **graph**.
  + **Nodes/Vertices:** Represent people or entities.
  + **Edges/Links:** Represent relationships (friendship, follows).
* **Clustering:** The process of finding "communities" or groups of nodes that are more densely connected to each other than to the rest of the network. (Example: Finding your close-knit friend circle on Facebook).

**5.3 Community Detection (Direct Discovery)**

* **Girvan-Newman Algorithm:** Finds communities by repeatedly removing edges with the highest **"betweenness centrality"** (edges that act as bridges between communities).
* **Clique Percolation Method (CPM):** Finds communities by looking for overlapping cliques (groups where every member is connected to every other member). Allows users to be in multiple communities.

**Chapter 6: Data Analytics with R**

**6.1 Basic Features of R**

* **R:** A programming language and environment for statistical computing and graphics.
* **RGUI vs. RStudio:** RGUI is the basic interface. RStudio is a powerful IDE (Integrated Development Environment) with a script editor, console, environment pane, and plots pane, making coding much easier.
* **Vectors:** The fundamental data structure. A sequence of elements of the same type. Created with c(). (Example: ages <- c(21, 23, 25)).
* **Objects:** Everything in R is an object (variables, data frames, functions).

**6.2 Data Handling & Functions**

* **Reading Data:** read.csv("file.csv") to read a CSV file into a **data frame**.
* **Exporting Data:** write.csv(data, "file.csv").
* **Functions:** Reusable blocks of code.

r

function\_name <- function(arg1, arg2) {

# Code to execute

return(result)

}

* **Built-in Functions:** R has many, like mean(), sum(), plot().

**6.3 Data Visualization**

* **Purpose:** To represent data graphically to uncover patterns, trends, and insights.
* **Types & Applications:**
  + **Bar Chart:** Compare categories (e.g., sales of different products). barplot()
  + **Histogram:** Show distribution of a numeric variable (e.g., distribution of exam scores). hist()
  + **Scatter Plot:** Show relationship between two numeric variables (e.g., height vs. weight). plot()
  + **Line Chart:** Show trends over time (e.g., stock price over days). plot(type="l")
  + **Boxplot:** Show summary of data (median, quartiles, outliers). boxplot()