**Module 1 – Introduction to big data**

**MU** \*\***Q. Describe five characteristics of Big data (5V’s):** (5-Marks)

**Ans: Big Data** refers to extremely large and complex datasets that are difficult to process using traditional methods. It involves high volumes of data generated rapidly from various sources and in different formats such as structured, semi-structured and unstructured. Advanced technologies and techniques are used to store, analyze, and extract meaningful insights from it.

The Five Main Characteristics of Big data, also known as 5 V’s of Big Data:

* **Volume:** Refers to the massive amount of data generated every second from sources like social media, sensors, transactions, and more. Traditional databases struggle to handle such large-scale data, requiring specialized storage and processing solutions.
* **Velocity:** Describes the speed at which data is generated, processed, and analyzed. Many industries require real-time or near real-time data processing, such as financial transactions, social media updates, and IoT devices.
* **Variety:** Data comes in different formats, including structured (databases, spreadsheets), semi-structured (JSON, XML), and unstructured (images, videos, emails, social media posts). Handling diverse data types requires advanced analytical techniques.
* **Veracity:** Refers to the quality and reliability of data. Since big data comes from multiple sources, it may contain inconsistencies and errors. Ensuring data accuracy and trustworthiness is crucial for meaningful analysis.
* **Value:** The ultimate goal of big data is to derive useful insights that create business value. Raw data itself has little meaning, but when processed and analyzed effectively, it can drive better decision-making, innovation, and efficiency.

**Q. Write three important characteristics of big data and explain any one with real life example [May 2024]**

**Ans:**  
The three most important characteristics of Big Data are known as the **Three Vs**:

1. **Volume:**  
   Volume refers to the huge amount of data generated continuously from various sources. It can range from terabytes to petabytes or even exabytes. Big data technologies are built to store and process such large datasets efficiently.
   * **Example:** Instagram generates data in the range of **petabytes** every day through user activity including images, videos, reels, and interactions.
2. **Velocity:**  
   Velocity refers to the speed at which data is created, transmitted, and processed. In many fields, real-time or near real-time processing is essential to take action based on the incoming data.
   * **Example:** Stock market trading systems process thousands of transactions per second. Real-time analysis is necessary to make timely buying and selling decisions.
3. **Variety:**  
   Variety means the data comes in different formats such as text, images, audio, video, or logs, and from multiple sources. This makes data handling more complex.
   * **Example 1:** In healthcare, a hospital may handle structured patient records, semi-structured data from wearable health trackers, and unstructured MRI scans or doctor's audio notes.
   * **Example 2:** In daily life, smartphones collect varied data such as typed messages, photos, voice commands, app logs, and GPS — all in different formats.

**MU \*\*Q. BDA real time examples/applications:** (5-Marks)

**Ans:**

**1. Healthcare & Medical Research**

* **Predictive Analytics:** Hospitals use big data to predict disease outbreaks and patient readmission rates.
* **Medical Imaging Analysis:** AI-powered systems analyze X-rays, MRIs, and CT scans to detect diseases.
* **Personalized Medicine:** Genetic data analysis helps in tailoring treatments for individual patients.
* **COVID-19 Tracking:** Real-time analytics helped monitor virus spread and vaccine distribution.

**2. Finance & Banking**

* **Fraud Detection:** Banks use real-time analytics to identify unusual transaction patterns and prevent fraud.
* **Risk Assessment:** Credit scoring models analyze customer data to determine loan eligibility.
* **Algorithmic Trading:** Stock market firms use BDA to execute high-frequency trading.
* **Customer Insights:** Personalized banking recommendations based on transaction history.

**3. E-Commerce & Retail**

* **Recommendation Engines:** Platforms like Amazon and Flipkart analyze user behavior to suggest products.
* **Dynamic Pricing:** Prices change based on demand, competitor pricing, and user activity.
* **Inventory Management:** Predicting demand to optimize stock levels and prevent shortages.
* **Customer Sentiment Analysis:** Analyzing reviews and social media to improve product offerings.

**4. Social Media & Digital Marketing**

* **Sentiment Analysis:** Companies analyze tweets, posts, and comments to understand public opinion.
* **Targeted Advertising:** Ads are personalized based on user behavior and browsing history.
* **Fake News Detection:** Platforms use AI to detect and flag misleading information.

**5. Transportation & Logistics**

* **Route Optimization:** Delivery services use real-time traffic data to find the fastest routes.
* **Fleet Management:** GPS and IoT sensors monitor vehicle health and driver performance.
* **Predictive Maintenance:** Airlines use sensor data to predict aircraft failures before they occur.

**6. Smart Cities & IoT**

* **Traffic Management:** Analyzing real-time traffic data to reduce congestion and optimize signals.
* **Energy Efficiency:** Smart grids analyze power consumption patterns for efficient energy distribution.
* **Public Safety:** CCTV and AI analytics help detect crimes in real time.

**7. Entertainment & Streaming Services**

* **Content Recommendation:** Platforms like Netflix and Spotify suggest movies and songs based on user preferences.
* **Audience Analytics:** Movie studios analyze social media trends to predict box office success.

**9. Education & E-Learning**

1. **Personalized Learning:** AI adapts course content based on student performance.
2. **Cheating Detection:** Exam proctoring software analyzes student behavior in real-time.

**10. Cybersecurity & Threat Detection**

* **Real-Time Intrusion Detection:** Analyzing network traffic to detect cyber threats.
* **User Behavior Analytics:** Identifying unusual login patterns to prevent data breaches.

**MU Q. what are three Vs of Big data? Give two examples of big data case studies. Indicate which Vs are satisfied by these case studies**  (5-Marks)

Ans:

The Three Main Characteristics of Big data:

* **Volume:** Refers to the massive amount of data generated every second from sources like social media, sensors, transactions, and more. Traditional databases struggle to handle such large-scale data, requiring specialized storage and processing solutions.
* **Velocity:** Describes the speed at which data is generated, processed, and analyzed. Many industries require real-time or near real-time data processing, such as financial transactions, social media updates, and IoT devices.
* **Variety:** Data comes in different formats, including structured (databases, spreadsheets), semi-structured (JSON, XML), and unstructured (images, videos, emails, social media posts). Handling diverse data types requires advanced analytical techniques.

Two Big Data Case Studies:

* 1. **Amazon – Customer Recommendation System:**

Amazon collects data from users’ browsing history, previous purchases, reviews, and clicks.

It uses this data to recommend products in real-time using machine learning algorithms.

Vs satisfied:

* **Volume:** Large amounts of customer and product data.
* **Velocity:** Real-time recommendation updates.
* **Variety:** Data includes text (reviews), images, and click behavior.
  1. **Healthcare – Disease Prediction using Patient Data:**

Hospitals collect data from electronic health records, lab tests, wearable devices, and medical images.

This data helps predict diseases like diabetes or heart conditions using predictive models.

Vs satisfied:

* **Volume:** Massive patient data across years.
* **Variety:** Includes numerical values, scans, and doctor notes.
* **Velocity:** Fast processing is needed for real-time alerts and diagnosis.

**MU Q. Explain what characteristics of social media make it suitable for BIG DATA** (5-Marks)

**Ans:** Social media platforms like Facebook, Twitter, Instagram, and YouTube generate and handle huge amounts of user-generated content every second.

The following characteristics make social media suitable for Big Data:

**1. High Volume of Data:** Millions of users post photos, videos, comments, likes, and shares daily, creating large-scale data continuously.

**2. High Velocity of Data Generation:** Data is produced in real time. Every second, new posts, tweets, and messages are shared, requiring fast data processing and analysis.

**3. Variety of Data Types:** Social media contains different forms of data such as text (comments), images, videos, live streams, and reactions (likes/emojis).

**4. Unstructured and Semi-Structured Data:** Most social media data is unstructured (like captions and images) or semi-structured (like hashtags and metadata), which fits Big Data technologies that handle such formats.

**5. User-Generated and Crowdsourced Content:** Content is created by users worldwide, contributing to a diverse and dynamic data source useful for trends, opinions, and behavior analysis.

**6. Real-Time Analytics:** Social media platforms use Big Data tools to monitor trends, breaking news and public sentiments in real-time.

**7. Scalability Needs:** As the number of users increases, social media platforms need scalable storage and processing systems, which are key features of Big Data infrastructure.

**MU Q. What are the challenges of Big data?** (5-Marks)

**Ans:** Big Data presents several challenges due to its sheer size, complexity, and speed of generation. The main challenges include:

**1. Data Storage & Management**

* Storing massive volumes of structured, semi-structured, and unstructured data requires scalable and cost-effective solutions.
* Traditional relational databases struggle to handle such large datasets, leading to the need for NoSQL databases, data lakes, and cloud storage.

**2. Data Processing & Analysis**

* Extracting meaningful insights from unstructured and semi-structured data is complex.
* Processing real-time data streams (e.g., social media feeds, IoT sensor data) requires powerful frameworks like Apache Spark and Hadoop.
* Ensuring data consistency and accuracy while processing massive datasets is challenging.

**3. Data Quality & Veracity**

* Large datasets often contain **incomplete, duplicate, inconsistent, or inaccurate data**.
* Cleaning and validating data for meaningful analysis is time-consuming and resource-intensive.

**4. Data Security & Privacy**

* Storing and processing sensitive information (personal, financial, healthcare data) increases risks of data breaches and cyberattacks.
* Compliance with regulations like **GDPR, HIPAA, and CCPA** adds complexity.

**5. Scalability & Infrastructure Costs**

* Big Data solutions require scalable infrastructure to handle growing data volumes.
* Cloud computing offers flexibility but can become expensive if not optimized properly.

**6. Real-Time Data Processing**

* Businesses need **real-time insights** (e.g., fraud detection, stock market analysis), which demand low-latency processing.
* Ensuring fast data ingestion and analysis without bottlenecks is a significant challenge.

**7. Integration of Multiple Data Sources**

* Big Data comes from **diverse sources** (social media, IoT devices, enterprise systems, etc.), making integration complex.
* Ensuring **compatibility** between structured, semi-structured, and unstructured data formats is difficult.

**8. Skilled Workforce Shortage**

* Handling Big Data requires expertise in **data engineering, data science, cloud computing, machine learning, and cybersecurity**.
* The demand for skilled professionals exceeds supply, creating a skills gap.

**9. Ethical & Compliance Issues**

* Misuse of data (e.g., biased AI models, unethical surveillance) raises ethical concerns.
* Companies must ensure responsible data usage while maintaining transparency and fairness.

**10. High Cost of Big Data Solutions**

* Implementing Big Data technologies involves significant investment in **hardware, software, cloud services, and skilled professionals**.
* Managing infrastructure efficiently while keeping costs under control is a challenge.

**Q. Compare Big Data analytics with traditional database**

**Ans:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Big Data Analytics** | **Traditional Database** |
| **Data type** | Handls structured, semi-structured and unstructured data. | Primarily structured data in rows and columns |
| **Data volume** | Large volume, Can process petabytes and exabytes of data. | Small volume, Limited to gigabytes or terabytes. |
| **Processing Approach** | Uses distributed computing and parallel processing (Hadoop, Spark). | Uses centralized processing (SQL-based). |
| **Storage System** | NoSQL databases, data lakes, cloud storage, distributed file systems. | Relational databases (SQL, MySQL, Oracle, PostgreSQL). |
| **Scalability** | Highly scalable (horizontal scaling with distributed clusters). | Limited scalability (vertical scaling by upgrading hardware). |
| **Speed & Performance** | Faster for massive datasets due to distributed processing | Slower when dealing with large datasets due to single-node processing. |
| **Real-Time Processing** | Supports real-time analytics (Kafka, Spark Streaming). | Limited real-time capabilities. |
| **Query Language** | Uses SQL, NoSQL (MongoDB, Cassandra), and frameworks like Hive. | Uses SQL for structured queries. |
| **Data Consistency** | Eventual consistency (in NoSQL) but highly available. | Strong consistency due to ACID compliance. |
| **Use Cases** | IoT, machine learning, real-time analytics, fraud detection, recommendation systems. | Banking transactions, payroll systems, inventory management, ERP applications. |
| |  | | --- | | **Cost Efficiency** |  |  | | --- | |  | | Cost-effective for large-scale data, often using open-source solutions. | Expensive when scaling up hardware for large databases. |

**Q. Identify the main sources of Big data in the real world**

**Ans:** Big Data is generated from various sources across different industries. The major sources include:

**1. Social Media Data**

* Platforms like **Facebook, Twitter, Instagram, LinkedIn, YouTube** generate vast amounts of unstructured data.
* Includes **posts, likes, shares, comments, videos, and user interactions**.
* Used for **sentiment analysis, targeted advertising, trend prediction**.
* **Example:** Twitter generates terabytes of tweets daily, which are analyzed for trends.

#### ****2. Search Engines****

* Search engines like **Google, Bing, and Yahoo** process **trillions of search queries** daily, storing keyword trends, user location, and browsing behaviour.
* **Example:** When users search for **“Who is the most favourite football player in the world?”** search engines analyze global data to rank the most searched players like **Ronaldo or Messi**.

**2. Internet of Things (IoT) & Sensor Data**

* Devices like **smartwatches, smart home systems, industrial sensors, connected vehicles** collect continuous real-time data.
* Used in **predictive maintenance, smart cities, healthcare monitoring, and automation**.
* **Example:** Smart meters track electricity usage in real time to optimize power distribution.

**3. Transactional Data**

* Includes financial transactions from **banks, e-commerce platforms, payment gateways, and stock markets**.
* Structured data from **purchase records, ATM transactions, credit card payments, invoices**.
* Used for **fraud detection, customer behavior analysis, risk management**.
* **Example:** Visa processes thousands of transactions per second and analyzes them for fraud detection.

**4. Machine-Generated Data**

* Generated by **servers, computer logs, GPS systems, network devices, and industrial machines**.
* Includes **log files, error reports, security logs, and activity tracking**.
* Used for **cybersecurity, system monitoring, and troubleshooting**.
* **Example:** Google Data Centers monitor server logs to detect failures.

**5. Healthcare & Medical Data**

* Data from **electronic health records (EHRs), medical imaging (MRIs, X-rays), wearable health devices**.
* Includes **patient history, drug prescriptions, genome sequencing, disease patterns**.
* Used for **diagnostics, personalized medicine, outbreak predictions**.
* **Example:** Hospitals analyze real-time patient vitals for early disease detection.

**6. E-commerce & Retail Data**

* Data from **online shopping platforms, customer reviews, purchase history, product searches**.
* Includes **clickstream data, abandoned carts, and loyalty programs**.
* Used for **recommendation engines, personalized marketing, and inventory management**.
* **Example:** Amazon tracks user browsing and purchase history to suggest products.

**7. Government & Public Sector Data**

* Includes **census records, tax filings, crime statistics, weather data, satellite imagery**.
* Used for **urban planning, disaster management, national security**.
* **Example:** NASA collects satellite data for climate change analysis.

**8. Media & Entertainment Data**

* Data from **streaming services (Netflix, Spotify, YouTube), gaming platforms, digital ads**.
* Includes **user preferences, watch history, engagement metrics**.
* Used for **content recommendation, ad targeting, audience insights**.
* **Example:** Netflix uses viewing history to recommend personalized content.

**9. Scientific Research Data**

* Data from **particle physics experiments (CERN), space research, genomics, drug discovery**.
* Requires **high-performance computing for simulation and analysis**.
* **Example:** The Large Hadron Collider generates petabytes of data from particle collisions.

**10. Telecommunications & Call Data Records (CDR)**

* Data from **mobile networks, call logs, SMS records, internet usage statistics**.
* Used for **network optimization, customer retention, fraud detection**.
* **Example:** Telecom providers analyze call drop rates to improve service quality.

**MU Q. Explain types of big data:** (5-Marks)

**Ans:** big data is classified into 3 types based on its format:

1. **Structured Data:**

Structured data is well-organized, follows a fixed schema, and is stored in a tabular format.

It is easy to process using traditional databases because each data point has a defined type and relationship.

It is stored in relational databases (SQL-based)

Easily searchable and analyzed using Structured Query Language (SQL).

**Example:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Class | Section | Roll No | Grade |
| Atul | 11 | A | 1 | A |
| Kartik | 11 | A | 2 | B |
| Aishwarya | 11 | A | 3 | A |

1. **Unstructured Data:**

Unstructured data lacks clear structure.

It does not follow a fixed format or predefined schema, making it difficult to process using traditional relational databases.

Comes in various formats such as text, images, videos, social media posts and tweets, etc.

Cannot be easily analyzed using SQL-based tools.

Requires specialized tools like Hadoop, Spark, or AI-based analysis.

It is usually stored in data lakes, NoSQL databases, or cloud storage systems.

1. **Semi-structured Data:**

Semi-structured data is a hybrid form that contains both structured and unstructured elements.

It does not follow a strict tabular format but includes tags, markers, or metadata that provide some organization.

More flexible than structured data but easier to analyze than unstructured data.

Commonly found in web applications and APIs.

Eg; JSON, XML, CSV files and Email metadata.

**MU Q**. **Distance measure for Big** **Data** (5-Marks)

**Ans:** Distance measures are mathematical formulas used to calculate the similarity or dissimilarity between data points. In Big Data, distance measures are important for tasks like clustering, classification, and recommendation systems.

Common Distance Measures Used in Big Data:

1. **Euclidean Distance:**
   * It is the most common distance measure.
   * It calculates the straight-line distance between two points in multi-dimensional space.
   * Formula:  
     
   * Used in K-means clustering and other machine learning algorithms.
2. **Manhattan Distance (L1 Distance):**
   * It calculates the distance by adding the absolute differences of the coordinates.
   * Formula:  
     
   * Useful when the data has high dimensionality.
3. **Cosine Similarity:**
   * It measures the cosine of the angle between two vectors.
   * It is useful for text mining and document similarity.
   * Value ranges from -1 to 1.
   * Higher cosine value means more similarity.
4. **Jaccard Similarity:**
   * It is used for comparing the similarity between two sets.
   * Formula:  
     
   * Useful in recommendation systems and user-behavior analysis.

**Q. Big data applications:**

**Ans:**

1. **Stock Market**  
   Big data is used to analyze stock prices, market trends, and trading volumes to predict future market movements. Real-time data helps traders make informed decisions and automate trading systems. It also helps in risk assessment and fraud detection.
2. **Education**  
   In education, big data helps track student performance, analyze learning behaviors, and provide personalized learning experiences. By analyzing test results, assignments, and class participation, educators can identify struggling students and offer targeted interventions.
3. **E-commerce**  
   E-commerce companies like Amazon and Flipkart use big data to track customer preferences, buying habits, and browsing history. This information is used to recommend personalized products, optimize pricing strategies, and manage inventory in real-time to meet demand.
4. **Social Media**  
   Social media platforms like Instagram, Facebook, and Twitter generate vast amounts of data from user interactions, posts, and trends. Big data helps these platforms analyze user behavior, target advertisements, and detect patterns or emerging trends. It also aids in content recommendations and user engagement strategies.
5. **Healthcare**  
   Big data in healthcare involves the analysis of patient records, clinical data, and medical research. This helps in early disease detection, predicting patient outcomes, improving treatments, and enhancing operational efficiencies in hospitals. Additionally, it supports personalized medicine by analyzing genetic and lifestyle data.
6. **Energy Sector**  
   In the energy industry, big data is used for monitoring energy consumption patterns, managing electricity grids, and predicting demand. It helps in optimizing the distribution of energy resources and is essential for renewable energy sources like wind and solar, where forecasting demand and production is complex.
7. **Transportation and Smart Cities**  
   Big data is crucial for managing traffic, optimizing routes for public transportation, and improving traffic flow. In smart cities, big data from sensors, IoT devices, and GPS helps in better urban planning, environmental monitoring, and optimizing infrastructure for reduced congestion and improved services.

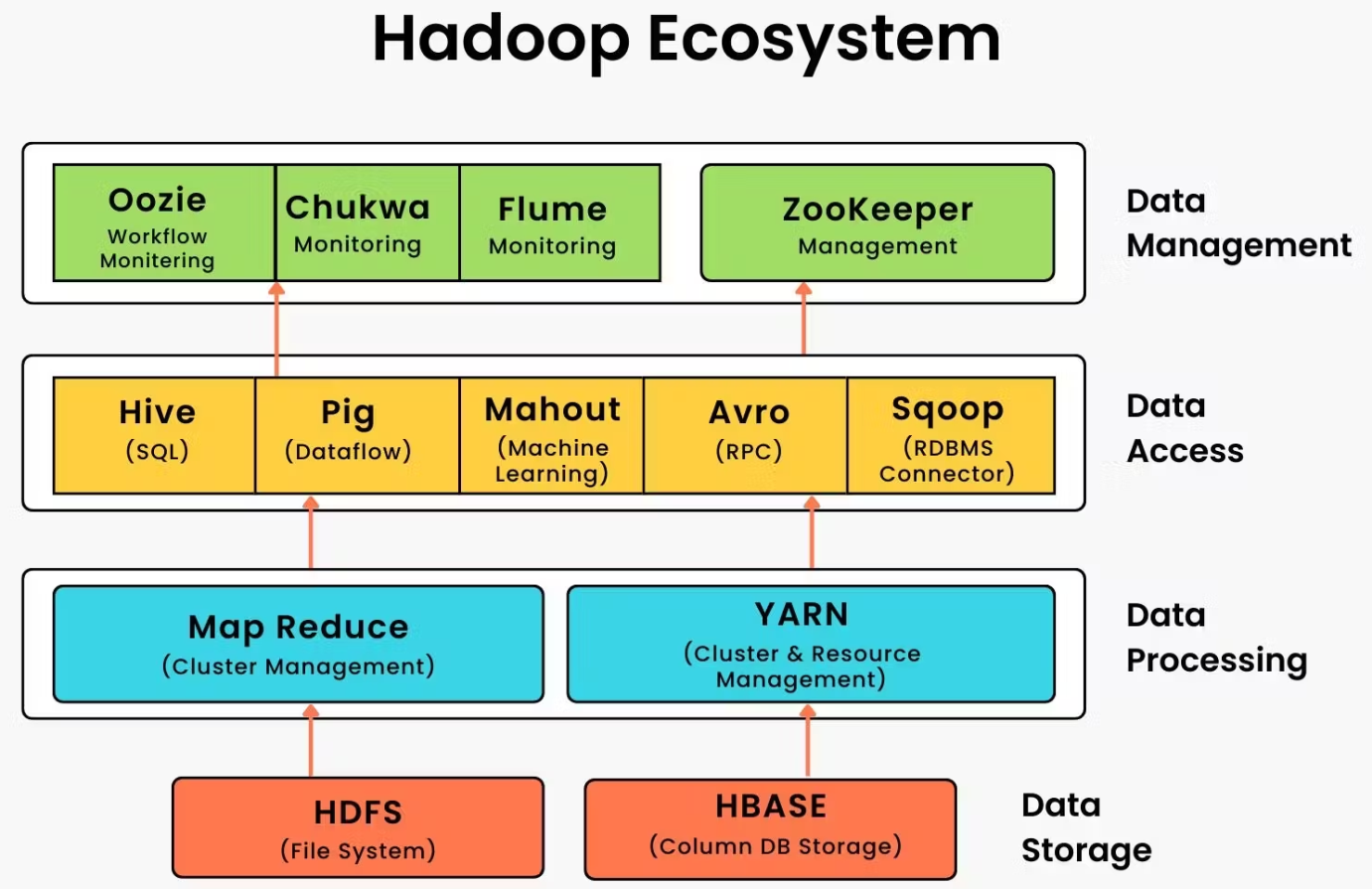
-------------------------------------------------------------------------------------------------

**Module 2 – Introduction to Big data Frameworks: Hadoop, NoSQL**

**MU\*\* Q. Hadoop Ecosystem (5 marks)**

**Explain Hadoop Ecosystem comoponents – Hive and Pig [May 2024]**

**Ans:**

****

Hadoop is an open-source software framework developed by the Apache Software Foundation. It is designed for the distributed storage and processing of large datasets across clusters of commodity hardware. Hadoop supports parallel processing and provides a highly scalable, reliable, and fault-tolerant architecture. It can handle structured, semi-structured, and unstructured data, making it ideal for big data analytics and real-time data management tasks. The core idea behind Hadoop is to bring computation to the data, rather than moving the data to a central server.

The Hadoop Ecosystem consists of various tools and technologies built around the core Hadoop framework. These components function collectively to enable large-scale data storage, computation, access, governance, and monitoring. The ecosystem is divided into multiple functional layers, with each layer responsible for a distinct set of tasks. These layers include data storage, data processing, data access, and data management.

1. Data Storage Layer:

This layer is responsible for storing massive volumes of data across multiple machines in a distributed and fault-tolerant manner. It comprises components that manage both persistent storage and real-time access.

a. HDFS (Hadoop Distributed File System):

* HDFS is the foundational storage system in Hadoop and is responsible for managing data storage across multiple nodes in a cluster.
* It stores large files by splitting them into smaller blocks and replicating these blocks across multiple machines for fault tolerance and reliability.
* It follows a write-once, read-many model, which simplifies consistency management and improves performance in batch processing scenarios.

• NameNode:

* + The NameNode serves as the master of the file system and manages the metadata, including information such as file names, permissions, and block locations.
  + It does not store actual data but plays a central role in client interactions and system coordination.

• DataNode:

* + DataNodes are the worker nodes that store and retrieve actual blocks of data upon instruction from the NameNode.
  + Each DataNode regularly sends heartbeat signals to the NameNode to confirm that it is active and functioning.

• Secondary NameNode:

* + It assists the primary NameNode by periodically checkpointing the filesystem's metadata.
  + This process helps in managing the growing size of log files and reduces recovery time during NameNode restarts.

• Structured and Unstructured Data:

* + HDFS can store all types of data formats, including structured (like CSV), semi-structured (like JSON or XML), and unstructured data (like videos, images, and logs).
  + It is optimized for storing large volumes of unstructured data, which is commonly generated in big data applications.

b. HBase:

* HBase is a NoSQL database that runs on top of HDFS and is designed to provide real-time access to large-scale structured data.
* It is ideal for sparse datasets with billions of rows and millions of columns.

• Column-Oriented Storage:

* + Data in HBase is stored in tables with rows and columns, but the storage model is based on column families rather than traditional row-based storage.
  + This allows for faster retrieval of data when queries are limited to certain columns.

• Random Read/Write:

* + HBase supports low-latency random read and write operations, making it suitable for real-time analytics and applications that need fast access to specific records.

• Integration with MapReduce:

* + HBase integrates with MapReduce for bulk processing of data and supports parallel processing of data stored in its tables.

1. Data Processing Layer:

This layer handles the computation aspect of Hadoop. It is responsible for executing large-scale data processing tasks across multiple nodes using parallel processing frameworks.

a. MapReduce:

* MapReduce is the original computation engine of Hadoop that supports distributed data processing using a simple programming model.
* It operates in two stages: the Map stage processes input data into key-value pairs, and the Reduce stage aggregates those intermediate results.

• Parallel Execution:

* + MapReduce automatically distributes tasks across multiple machines in the cluster, allowing for simultaneous execution and faster job completion.

• Fault Tolerance:

* + If a task fails on one machine, it is automatically retried on another machine, ensuring job reliability even in the event of hardware failures.

• Batch-Oriented:

* + It is best suited for batch processing tasks where large volumes of data need to be processed sequentially.

b. YARN (Yet Another Resource Negotiator):

* YARN is the resource management framework of Hadoop that allows different processing engines to run simultaneously on a single cluster.

• ResourceManager:

* + It manages and allocates computing resources to various applications running in the cluster, ensuring optimal usage of system resources.

• NodeManager:

* + Each node in the cluster runs a NodeManager, which monitors local resource usage and reports it to the ResourceManager.

• ApplicationMaster:

* + Every application has its own ApplicationMaster that handles the scheduling and coordination of tasks specific to that application.

• Support for Multiple Frameworks:

* + YARN enables the use of other data processing frameworks like Apache Spark, Apache Flink, and Tez alongside MapReduce.

1. Data Access Layer:

This layer provides users with tools to interact with and analyze data stored in the Hadoop cluster. It supports SQL-like languages and scripting interfaces.

a. Hive:

* Hive is a data warehouse infrastructure built on top of Hadoop for querying and analyzing large datasets using a SQL-like language called HiveQL.

• SQL Interface:

* + Hive allows users to write queries in HiveQL, which are automatically converted into MapReduce jobs by the system.

• Schema and Metadata Management:

* + Hive maintains a Metastore that stores metadata about tables, partitions, and their schemas, enabling structured query processing.

• Analytics at Scale:

* + It is optimized for read-heavy workloads and batch analytics on massive datasets.

b. Pig:

* Pig is a high-level scripting language platform that simplifies the processing of large datasets using a language called Pig Latin.

• Easy-to-Use Scripts:

* + Pig Latin scripts are more concise and easier to understand than equivalent Java-based MapReduce programs.

• UDF Support:

* + Pig allows the creation of custom User Defined Functions (UDFs) for advanced data processing tasks.

c. Sqoop:

* Sqoop is used to transfer data between Hadoop and traditional relational databases like MySQL or Oracle.

• Bi-Directional Data Transfer:

* + It supports importing data from RDBMS to HDFS as well as exporting processed data from HDFS back to relational databases.

• Integration with Hive:

* + Sqoop can directly import data into Hive tables for easier query execution.

d. Mahout:

* Mahout is a scalable machine learning library built for Hadoop that provides algorithms for clustering, classification, and recommendation systems.

• Pre-Built Algorithms:

* + Mahout includes a wide range of ready-to-use machine learning algorithms that are designed to run in parallel.

• Support for Spark:

* + It has evolved to support more advanced engines like Apache Spark for better speed and scalability.

e. Avro:

* Avro is a data serialization framework that supports schema-based serialization and communication between distributed systems.

• Cross-Language Compatibility:

* + Data serialized using Avro can be easily transmitted and interpreted across programs written in different languages.

• Lightweight Storage:

* + Avro files are compact, which saves space and improves read/write efficiency.

1. Data Management and Coordination Layer:

This layer manages job coordination, system configuration, performance monitoring, and service orchestration.

a. Oozie:

* Oozie is a workflow scheduler system designed to manage and coordinate Hadoop jobs such as MapReduce, Pig, and Hive.

• Workflow Definition:

* + Users can define a series of dependent jobs that run in a specified sequence based on time or data availability.

• Error Handling:

* + It includes retry policies and failure-handling mechanisms to maintain job consistency.

b. ZooKeeper:

* ZooKeeper is a centralized service for configuration management, distributed synchronization, and naming.

• Coordination Support:

* + It enables distributed systems to coordinate tasks like leader election, configuration updates, and distributed locks.

• Used in Other Tools:

* + ZooKeeper is often used by HBase, Kafka, and other distributed tools for maintaining system integrity.

c. Flume:

* Flume is a distributed service used for collecting and transporting large volumes of event and log data into HDFS.

• Real-Time Data Ingestion:

* + Flume is suitable for streaming logs and other real-time data sources directly into Hadoop.

• Source-to-Sink Model:

* + Data flows through channels from source agents (e.g., web servers) to storage sinks (e.g., HDFS).

d. Chukwa:

* Chukwa is used for monitoring and analyzing log data in Hadoop clusters.

• Built on Hadoop:

* + It utilizes HDFS for storage and MapReduce for processing collected metrics and logs.

• System Monitoring:

* + Chukwa helps administrators gain insight into system performance and behavior.

e. Ambari:

* Ambari is a management and monitoring tool for provisioning and maintaining Hadoop clusters.

• User Interface:

* + It offers a web-based dashboard that displays the health status of cluster nodes and services.

• Service Lifecycle Management:

* + Administrators can install, configure, start, and stop services from a central interface.

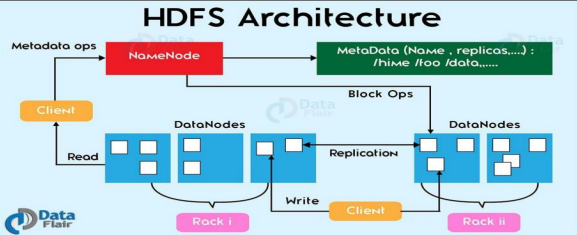
• Alerts and Security:

* + Ambari provides alerts and log tracking, making it easier to troubleshoot and secure the cluster.

Let me know if you’d like me to convert this into a Word or PDF format, or if you want a similar format for other topics.

**MU \*\*Q. describe structure of HDFS in a Hadoop ecosystem using a diagram./HDFS architecture in detail:**

**Ans:**



Hadoop comes with a distributed file system called HDFS (HADOOP Distributed File Systems) HADOOP based applications make use of HDFS.

• HDFS is designed for storing very large data files, running on clusters of commodity hardware. It is fault tolerant, scalable, and extremely simple to expand.

• Hadoop HDFS has a Master/Slave architecture in which Master is NameNode and Slave is DataNode(worker node).

• HDFS Architecture consists of single NameNode and all the other nodes are DataNodes

Divide file into segments called blocks.

This block is mapped by namenode in to one of the datanode.

Blocks are not divided randomly, 64mb is default size for dividing the blocks.

Components of hdfs:

**A. NameNode**

It is also known as Master node. HDFS Namenode stores meta-data like file size, type, name, location, number of data blocks, replicas and other details. It doesn’t store actual data. This meta-data is available in memory in the master for faster retrieval of data. NameNode maintains and manages the data nodes, and assigns tasks to them.

It performs operations such as opening, closing, and renaming files and directories.

If datanode capacity get fulled, namenode instruct datanode to delete the particular blocks which are not required, can instruct to replicate blocks which are more important so incase of datanode failure , client can get access to replicated one.

Files present in the NameNode metadata are as follows

**FsImage** – It is an “Image file”. FsImage contains the entire filesystem namespace and stored as a file in the namenode’s local file system.It also contains a serialized form of all the directories and file inodes in the filesystem.Each inode is an internal representation of file or directory’s metadata.

**EditLogs** – It contains all the recent modifications made to the file system on the most recent FsImage. Namenode receives a create/update/delete request from the client.After that this request is first recorded to edits file

**B. DataNode**

It is also known as Slave or worker node. DataNode stores actual data that is coming from files, in HDFS. responsible for storing and retrieving actual data blocks as instructed by the NameNode.

.DataNodes can deploy on commodity hardware.

Performing block creation, deletion, and replication upon instruction from the NameNode

It performs read and write operation as per the request of the client,

Periodically sending block reports and heartbeats to the NameNode to confirm its status.

DataNode manages data storage of the system.

DataNodes send heartbeat to the NameNode to report the health of HDFS. By default, this frequency is set to 3 seconds.

**C. Secondary Namenode**

The Secondary NameNode acts as a helper to the primary NameNode, primarily responsible for merging the EditLogs with the current filesystem image (FsImage) to reduce the potential load on the NameNode. It creates checkpoints of the namespace to ensure that the filesystem metadata is up-to-date and can be recovered in case of a NameNode failure.

* Merging EditLogs with FsImage to create a new checkpoint.
* Helping to manage the NameNode's namespace metadata.

**D. HDFS Client:**

Clients- users or applications interacting with hdfs.

The HDFS client is the interface through which users and applications interact with the HDFS. It allows for file creation, deletion, reading, and writing operations. The client communicates with the NameNode to determine which DataNodes hold the blocks of a file and interacts directly with the DataNodes for actual data read/write operations.

* Facilitating interaction between the user/application and HDFS.
* Communicating with the NameNode for metadata and with DataNodes for data access.

**E. Block structure**

HDFS stores files by dividing them into large blocks, typically 128MB or 256MB in size. Each block is stored independently across multiple DataNodes, allowing for[parallel processing](https://www.geeksforgeeks.org/what-is-parallel-processing/) and fault tolerance. The NameNode keeps track of the block locations and their replicas.

* Large block size reduces the overhead of managing a large number of blocks.
* Blocks are replicated across multiple DataNodes to ensure data availability and fault tolerance.

**Rack Awareness**

In a large cluster of Hadoop, in order to improve the network traffic while reading/writing HDFS file, NameNode chooses the DataNode which is closer to the same rack or nearby rack to Read /write request. NameNode achieves rack information by maintaining the rack ids of each DataNode. Rack Awareness in Hadoop is the concept that chooses Datanodes based on the rack information.

**Q. How big data problems are handled by Hadoop ecosystem**

**Ans:** Big Data poses several challenges, such as **data storage, processing speed, scalability, fault tolerance, and variety of data types**. The Hadoop ecosystem provides an effective solution by offering **distributed storage, parallel processing, resource management, and integration tools**.

#### ****1. Handling Large Data Volumes with HDFS (Hadoop Distributed File System)****

* Big Data is often **too large to store in traditional databases**. HDFS overcomes this by **splitting large files into smaller blocks** (default: **128MB or 256MB**) and distributing them across multiple nodes.
* **Replication ensures fault tolerance** by keeping multiple copies of the data (default: three copies).
* Example: **A 1PB dataset in an e-commerce business** is distributed across multiple servers, making storage and retrieval efficient.

#### ****2. High-Speed Processing with MapReduce****

* Traditional systems process data **sequentially**, which is slow for large datasets.
* Hadoop uses **MapReduce**, which processes data in a **parallel, distributed manner** across multiple nodes, reducing processing time significantly.
* Example: A **social media company** can analyze **millions of user posts, likes, and comments** in parallel instead of processing them one by one.

#### ****3. Efficient Resource Management with YARN****

* Handling **multiple workloads** simultaneously is challenging. YARN (Yet Another Resource Negotiator) allows **efficient allocation of CPU, memory, and processing power** across different applications.
* Example: **A bank running fraud detection models and transaction analysis in parallel** can optimize resource usage with YARN.

#### ****4. Supporting Structured, Semi-Structured, and Unstructured Data****

* Traditional databases require **structured data (rows and columns)**, but Hadoop can handle **all types of data**:
  + **Structured** (e.g., customer records in MySQL)
  + **Semi-structured** (e.g., JSON, XML files from IoT devices)
  + **Unstructured** (e.g., videos, images, text from social media)
* Example: A **video streaming platform like Netflix** stores video files in HDFS while metadata (title, category, duration) is managed using tools like Hive.

#### ****5. Real-Time & Batch Processing with Hadoop Ecosystem Tools****

* **Batch Processing**: Hadoop’s **MapReduce and Hive** handle large-scale batch data processing efficiently.
* **Real-Time Processing**: Tools like **Apache Spark and Flink** process streaming data in real-time.
* Example: **Stock market applications use Spark for real-time analysis of stock price fluctuations**.

#### ****6. Scalability with Distributed Computing****

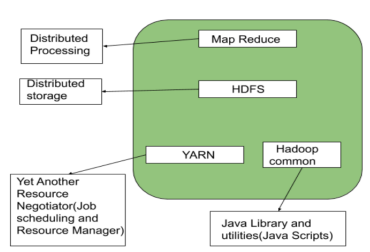
* Traditional databases require **costly hardware upgrades** to scale up, whereas Hadoop allows **horizontal scalability** by adding more commodity hardware (nodes).
* Example: A **telecom company processing call logs** can expand storage and computing capacity by adding inexpensive servers to the cluster.

#### ****7. Data Integration and Querying with Ecosystem Tools****

* **Apache Hive**: Converts SQL-like queries into MapReduce jobs, making it easy for analysts to work with Big Data.
* **Apache Pig**: A scripting language for complex data transformations.
* **Apache Sqoop**: Transfers data between **RDBMS and Hadoop**.
* **Apache Flume**: Collects and ingests **log data from multiple sources** into Hadoop.
* Example: A **banking system using Sqoop to import customer transaction data from MySQL to Hadoop** for further analysis.

**\*\*Q. Hadoop Architecture**

**Ans:** Hadoop is a framework written in Java that utilizes a large cluster of commodity hardware to maintain and store big size data. Hadoop works on MapReduce Programming Algorithm that was introduced by Google. Today lots of Big Brand Companies are using Hadoop in their Organization to deal with big data, eg. Facebook, Yahoo, Netflix, eBay, etc. The Hadoop Architecture Mainly consists of 4 components.



The Hadoop Architecture Mainly consists of 4 components.

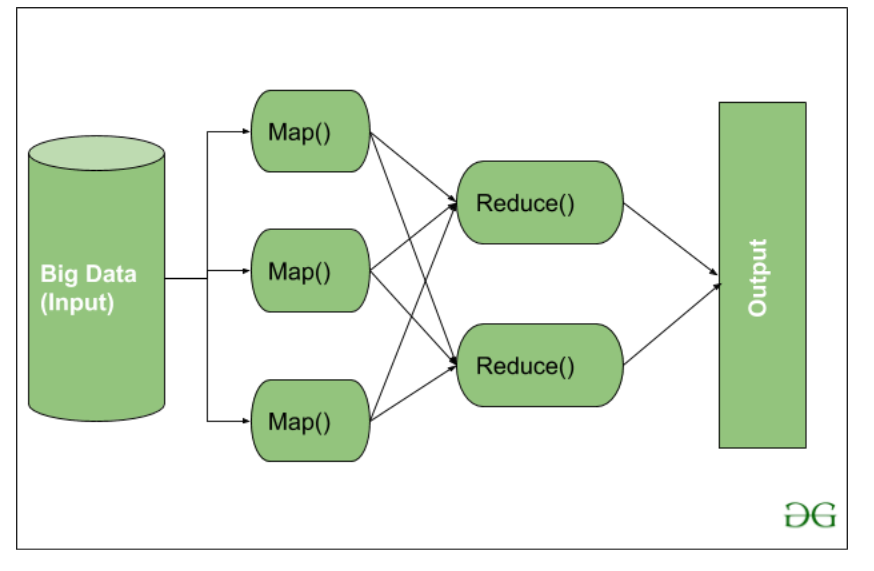
A. MapReduce

MapReduce is a programming model for **processing large-scale datasets in parallel** using the **divide-and-conquer approach**. It consists of two phases:

1. **Map Phase** – Splits the input data into key-value pairs and processes them in parallel.
2. **Reduce Phase** – Aggregates the mapped data and generates the final result.

For example, if a company wants to count the occurrence of words in a **large collection of text files**, the **Map phase** will break the text into words and assign a count of 1 to each word, while the **Reduce phase** will sum up the counts to find the total occurrences of each word.

MapReduce is **fault-tolerant** as it automatically reassigns failed tasks to other available nodes, ensuring smooth execution of data processing jobs.



**B. HDFS(Hadoop Distributed File System) :**

HDFS is responsible for storing large datasets in a distributed manner across multiple nodes. It follows a **Master-Slave Architecture**, where the **NameNode (Master)** manages metadata (file locations, permissions, and structure), and multiple **DataNodes (Slaves)** store the actual data blocks.

HDFS splits large files into smaller **blocks (default: 128MB or 256MB)** and distributes them across multiple nodes. To ensure **fault tolerance**, Hadoop maintains multiple copies of each data block (default: three copies) so that if one node fails, the data can still be retrieved from another node.

For example, if a **1TB file** is uploaded to HDFS, it is split into **128MB blocks** and stored across different nodes with replication, ensuring high availability and reliability.

**C. YARN(Yet Another Resource Negotiator)**

YARN is responsible for managing and allocating cluster resources like CPU and memory efficiently across multiple applications running in the Hadoop ecosystem. It allows multiple **Big Data applications (e.g., Spark, MapReduce, Hive, and Pig)** to run simultaneously without resource conflicts.

The key components of YARN include:

* **ResourceManager (Master Node)** – Allocates and monitors cluster resources.
* **NodeManager (Slave Nodes)** – Runs on each node and reports resource usage to the ResourceManager.
* **ApplicationMaster** – Manages the execution of individual applications within YARN.

For instance, if an organization runs both **Spark and MapReduce jobs**, YARN ensures fair resource distribution and prevents one application from monopolizing all resources.

**D. Common Utilities or Hadoop Common**

The Common Utilities in Hadoop provide essential **libraries, configuration files, and APIs** that enable seamless interaction between different components of the Hadoop ecosystem. These utilities help maintain cluster settings, enable data serialization, and support data compression techniques.

For example, Hadoop Common Utilities ensure that tools like **Hive, Pig, and Sqoop** can communicate efficiently with HDFS, YARN, and MapReduce without compatibility issues.

**Q. difference between rdbms and hadoop**

**Ans:**

|  |  |
| --- | --- |
| **RDBMS** | **Hadoop** |
| Traditional row-column based databases, basically used for data storage, manipulation and retrieval. | Open-source software used for storing data and running applications or processes concurrently. |
| Primarily, process structured data. | Process both structured and unstructured data. |
| It is best suited for [OLTP](https://www.geeksforgeeks.org/on-line-transaction-processing-oltp-system-in-dbms/) (Online transaction processing) environment. | It is best suited for BIG data. |
| It is less scalable than Hadoop. | It is highly scalable. |
| Data normalization is required in RDBMS. | Data normalization is not required in Hadoop. |
| It stores transformed and aggregated data. | It stores huge volume of data. |
| It has no latency in response. | It has some latency in response. |
| The data schema of RDBMS is static type. | The data schema of Hadoop is dynamic type. |
| High data integrity available. | Low data integrity available than RDBMS. |
| Cost is applicable for licensed software. | Free of cost, as it is an open source software. |
| Data is process using [SQL](https://www.geeksforgeeks.org/what-is-sql/). | Data is process using [Map-Reduce](https://www.geeksforgeeks.org/mapreduce-understanding-with-real-life-example/). |
| Follow ACID properties. | Does not follow [ACID properties](https://www.geeksforgeeks.org/acid-properties-in-dbms/). |

**MU Q. Differentiate between SQL and NoSQL system. (5 marks) [May 2024]**

**Ans:**

|  |  |
| --- | --- |
| **SQL System** | **NoSQL System** |
| SQL stands for Structured Query Language and is used for managing relational databases. | NoSQL stands for Not Only SQL and is used for managing non-relational or distributed databases. |
| SQL databases store data in a structured format using tables with rows and columns. | NoSQL databases store data in flexible formats like documents, key-value pairs, graphs, or wide-column stores. |
| SQL databases require a fixed and predefined schema before storing data. | NoSQL databases allow a dynamic or flexible schema, so data structure can change easily. |
| SQL systems scale vertically, which means performance increases by upgrading the server’s hardware. | NoSQL systems scale horizontally, which means performance increases by adding more servers. |
| SQL databases are best for storing structured data that fits into tables. | NoSQL databases support structured, semi-structured, and unstructured data like JSON, text, and images. |
| SQL systems use SQL as a standard query language to manage and retrieve data. | NoSQL systems use different query languages or APIs depending on the database type. |
| Examples of SQL databases are MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. | Examples of NoSQL databases are MongoDB, Cassandra, Redis, and CouchDB. |
| SQL is best for applications that need complex queries and strict data consistency, like banking systems. | NoSQL is best for applications dealing with large-scale, real-time, or unstructured data, like social media platforms. |
| SQL follows ACID properties to ensure data integrity and reliable transactions. | NoSQL follows the BASE model, which gives better performance and flexibility but may compromise consistency. |

**MU Q. What are advantages and limitations of Hadoop. (5 marks) [May 2024]**

**Ans**:

**Advantages of Hadoop:**

**1. Highly Scalable:** Hadoop can easily handle increasing volumes of data by adding more machines (nodes) to the cluster without changing the data processing logic.

**2. Cost-effective:** It uses inexpensive commodity hardware, making it an affordable solution for storing and processing massive datasets.

**3. Fault Tolerance**: Hadoop automatically creates multiple copies of data and stores them across different machines. If one node fails, data can still be accessed from another, ensuring reliability.

**4. Flexibility in Data Handling**: It can process structured, semi-structured, and unstructured data such as text, images, videos, and log files, which traditional systems cannot handle easily.

**5. Open Source:** Hadoop is open-source, so it is free to use and has a large community that continuously contributes to its improvement.

**6. Data Locality Concept:** Hadoop processes data on the same machine where it is stored, which reduces network traffic and improves processing speed.

**7. Rich Ecosystem:** It is supported by a wide range of tools like Hive, Pig, Mahout, HBase, and Oozie, which make data processing, analysis, and workflow management easier.

**Limitations of Hadoop:**

1. Not Suitable for Small Data: Hadoop is designed for large-scale data processing. Using it for small datasets or small clusters may lead to inefficient results.

2. High Energy and Maintenance Costs: Though hardware is cheap, running many machines consumes a lot of power and requires constant maintenance.

3. Data Replication Overhead: To maintain fault tolerance, Hadoop replicates data multiple times, which increases storage requirements and reduces storage efficiency.

4. Lack of Real-time Processing: Hadoop is mainly used for batch processing. It cannot handle real-time data processing effectively.

5. Complex to Learn and Manage: The Hadoop ecosystem includes many tools and frameworks, which can be overwhelming for beginners to learn and manage.

**6. Security Concerns:** By default, Hadoop has limited security features. It requires extra configuration and tools to ensure data privacy and protection.

**MU\*\*Q. Explain CAP theorem and explain how NoSQL systems guarantees BASE property. (5 marks)**

**Ans:**

**CAP Theorem:**

The CAP theorem, also known as Brewer’s theorem, is a fundamental principle in distributed database systems. It states that **a distributed system can guarantee only two out of the following three properties at a time**:

1. **Consistency (C):** All nodes in the system show the same data at the same time. That means, after an update, all users see the latest data immediately.
2. **Availability (A):** Every request receives a response, even if some nodes are not working. The system remains operational at all times.
3. **Partition Tolerance (P):** The system continues to function even if there is a communication failure between nodes in the network (i.e., if the system is divided into parts).

According to the theorem, a distributed system **must tolerate partition tolerance**, and then it has to choose between **Consistency and Availability**.

* For example, if a system prioritizes **Consistency and Partition Tolerance (CP)**, it may not always be available.
* If it chooses **Availability and Partition Tolerance (AP)**, it may allow temporary inconsistency.

**BASE Properties in NoSQL:**

NoSQL systems do not follow the traditional ACID properties (used in relational databases). Instead, they follow the **BASE** model, which is designed for **scalability and performance** in distributed environments.

The term BASE stands for:

1. **Basically Available:** The system guarantees availability, meaning it always responds to requests—even if the data is not the most recent or accurate.
2. **Soft State:** The system's state may change over time, even without input, due to eventual consistency. The state does not have to be consistent at all times.
3. **Eventually Consistent:** The system will become consistent over time, as updates propagate to all nodes. Temporary inconsistencies are acceptable for the sake of performance and availability.

**How NoSQL Guarantees BASE:**

* NoSQL systems are **highly distributed**, and to handle large-scale data with fast performance, they **sacrifice strong consistency** in favor of availability and partition tolerance.
* By using **eventual consistency**, NoSQL systems ensure that all copies of data will match eventually, even if they are temporarily out of sync.
* This approach allows them to remain **basically available** and fault-tolerant, which is crucial for applications like social media, online shopping, or real-time analytics.

**MU**\*\*\*\***Q. Explain column family store and graph store NoSQL architectural pattern with example**

**Ans:**

**Column family stores:**

Column family systems are important NoSQL data architecture patterns because they can scale to manage large volumes of data. They’re also known to be closely tied with many MapReduce systems.

A column family store organizes data into rows and columns, but unlike traditional relational databases, it does not enforce a fixed schema.

Each row can contain a different set of columns, and each column can store any type of data, such as strings, integers, floats, or even binary formats like images.

They’re sometimes referred to as data stores rather than databases, since they lack certain features may expect to find in traditional databases like RDBMS (Relational Database Management Systems)

Unlike RDBMS, They lack typed columns, secondary indexes, triggers and Advanced query languages (like SQL).

Despite this, they offer flexibility and performance advantages in use cases involving large, distributed data.

Almost all column family stores have been heavily influenced by the original Google Bigtable paper, where data is stored in column families rather than tables.

### Key Features:

* Each column family stores data with a row key and column key for easy retrieval.
* There is no restriction on data types stored in the cells.
* Data is accessed through columns rather than fixed table structures.
* Suited for sparse data and read/write-heavy applications.

Apache HBase, Hypertable, and Cassandra are good examples of systems that have Bigtable like interfaces, although how they differ in their specific implementations.

|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | C |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  | Hello World! |  |
| 4 |  |  |  |
| 5 |  |  |  |

Using a row and column to address a cell, the cell has an address of 3B and can be thought of as the lookup key in sparse matrix systems.

A cell with the address 3B indicates that the data is located in row 3, column B. This address acts as a lookup key to access or store the value in that specific cell.

**Graph store:**

A Graph Store is a type of NoSQL database that uses graph structures with nodes, edges, and properties to represent and store data. This model is especially effective for applications where relationships between data points are as important as the data itself.

In a graph database:

* Nodes represent entities (e.g., people, places, products).
* Edges represent relationships or connections between those entities (e.g., "friends with", "purchased", "connected to").
* Properties are key-value pairs that store information about nodes and edges (e.g., name, date, weight).

This structure allows the system to efficiently query and traverse relationships, which is difficult and time-consuming in relational databases.

Example: Social Network

In a social media application:

* Each user is a node.
* The "follows" or "friends with" relation is an edge between two user nodes.
* Each node can have properties like name, age, or location.
* Edges can also have properties like date followed or interaction count.

If we want to find mutual friends, shortest connection path, or recommend new connections, a graph store can process these queries much faster than traditional models.

Common Graph Store Systems:

* Neo4j
* Amazon Neptune
* OrientDB
* ArangoDB

**MU** **Q. Recall all NoSQL design patterns with example. Justify CAP property. [May 2024]**

**Ans:** There are several NoSQL design patterns that help in building scalable and efficient applications. Each pattern addresses a specific type of data structure or access requirement.

#### 1. Key-Value Store Pattern

* **Description:** Stores data as a collection of key-value pairs.
* **Example:** Redis, Amazon DynamoDB
* **Use Case:** Session storage, caching, real-time analytics.

#### 2. Column Family Store Pattern

* **Description:** Data is stored in rows and columns, but unlike RDBMS, each row can have a different set of columns.
* **Example:** Apache Cassandra, HBase
* **Use Case:** Time-series data, logs, event monitoring.

#### 3. Document Store Pattern

* **Description:** Stores data as documents (usually JSON, BSON, or XML). Each document can have a unique structure.
* **Example:** MongoDB, CouchDB
* **Use Case:** Content management systems, user profiles, product catalogs.

#### 4. Graph Store Pattern

* **Description:** Stores data in the form of nodes and edges representing entities and relationships.
* **Example:** Neo4j, Amazon Neptune
* **Use Case:** Social networks, fraud detection, recommendation systems.

#### 5. Object Store Pattern

* **Description:** Stores complex data objects directly.
* **Example:** Riak, db4o
* **Use Case:** Applications with object-oriented models that need persistent storage.

#### 6. Multi-Model Pattern

* **Description:** Supports multiple types of NoSQL models (key-value, document, graph, etc.) in a single engine.
* **Example:** ArangoDB, OrientDB
* **Use Case:** Complex applications that require diverse data models.

### Justification of CAP Theorem in NoSQL Systems

The **CAP theorem** states that in any distributed database system, it is impossible to guarantee **Consistency (C)**, **Availability (A)**, and **Partition Tolerance (P)** all at the same time. A NoSQL system must choose any two of the three.

* **Consistency (C):** All nodes see the same data at the same time.
* **Availability (A):** Every request gets a (non-error) response.
* **Partition Tolerance (P):** The system continues to function despite communication failures between parts of the system.

**MU** \*\***Q. Short note on NoSQL data stores with example.**

**Ans:**

**1. Key-Value store:**

A key-value store is a simple database that when presented with a simple string (the key) returns an arbitrary large BLOB of data (the value.)

A key-value store is like a dictionary.

|  |  |  |
| --- | --- | --- |
| **Image name 🡪**  **Web page URL 🡪**  **File path name 🡪**  **MD5 Hash 🡪**  **REST web service call🡪**  **SQL Query 🡪** | **key** | **Value** |
| Image-12345.jpg | Binary image file |
| http://www.example.com/my-web-page.html | HTML of a web page |
| N:/folder/subfolder/myfile.pdf | Pdf document |
| 80633b1a0311bcfab11e78555d915fc3 | The quicker brown fox jumps over the lazy dog |
| View-person?person-id=12345&format=xml | <Person><id>12345</id></Person> |
| SELECT PERSON FROM PEOPLE WHERE PID= “12345” | <Person><id>12345</id></Person> |

2. Graph store. (copy above)

3. Column-family (Bigtable) store(copy above)

4. Document store (copy below)

**MU** Q. Explain the types of NoSQL data stores and their typical usage.

Ans:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pattern name | Description | Typical uses |
| 1 | Key-Value store | A simple way to associate a large data file with a simple text string | Dictionary, image store, document/file store, query cache, lookup tables |
| 2 | Graph Store | A way to store nodes and arcs of a graph | Social network queries, friend-of-friends queries, inference, rules system, and pattern matching. |
| 3 | Column-family (Bigtable) store | A way to store sparse matrix data using a row and column as the key | Web crawling, large sparsely populated tables, highly-adaptable systems, systems that have high variance. |
| 4 | Document store | A way to store tree-structured hierarchical information in a single unit. | Any data that has a natural container structure including office documents, sales orders, invoices, product descriptions, forms, and web pages; popular in publishing, document exchange, and document search. |

**Module 3 – MapReduce Paradigm**

**Q. Write a map reduce pseudo code to multiply two matrices. Illustrate with an example showing all the steps**

**Ans:**

**MU Q. Explain the Map and Reduce Tasks (5 marks)**

**Ans:**

In the Hadoop framework, **Map and Reduce** are two key functions used in the **MapReduce programming model**. They help in processing and analyzing large-scale data in a distributed environment.

**1. Map Task:**

* The **Map task** is the first phase in the MapReduce model.
* It takes the input data, processes it, and produces a set of intermediate **key-value pairs**.
* The main job of the map function is to **filter, sort, and organize** the data.
* For example, in a word count problem, the map function takes each line of text as input and produces output in the form of (word, 1).

**Example:**  
Input: "Big data is big"  
Output from Map: (Big,1), (data,1), (is,1), (big,1)

Note: Hadoop converts all inputs into key-value format internally, even if it is a plain text file.

**2. Shuffle and Sort Phase (Intermediate Step):**

* After the map phase, the intermediate key-value pairs are **shuffled and sorted** automatically.
* Pairs with the same key are grouped together and passed to the reduce function.

**3. Reduce Task:**

* The **Reduce task** is the second and final phase of MapReduce.
* It **takes the grouped key and a list of values**, and then performs **aggregation or summarization**.
* In the word count example, the reduce task will add up all the values for the same key.

**Example:**  
Input to Reduce: (Big, [1,1])  
Output: (Big, 2)

* So, the final output will show how many times each word appeared in the input data.

**MU Q. Explain MapReduce programming model in detail (5 marks) [May 2024]**

**Ans:**

The **MapReduce programming model** is a core part of the Hadoop framework. It is used to process and analyze **large volumes of data in a distributed and parallel manner** across multiple nodes in a Hadoop cluster.

MapReduce works in **two main phases**: the **Map phase** and the **Reduce phase**. Between them, there is an intermediate step called **Shuffle and Sort**.

**1. Map Phase:**

* The Map phase is the **first step** of the model.
* It takes input data and processes it into smaller chunks in the form of **key-value pairs**.
* This phase is responsible for **filtering, grouping, or transforming** raw data.
* Each map task runs independently on a separate block of data.

**Example:**  
For a word count program, the map function reads text lines and produces output like:  
(word, 1) for each word found.

**2. Shuffle and Sort Phase (Intermediate Step):**

* This step happens automatically between map and reduce phases.
* The **Shuffle** step transfers the output of map tasks to the correct reducer based on the key.
* The **Sort** step groups all values associated with the same key together.
* It ensures that each reducer gets all the values for one key.

**Example:**  
From mapper output like (word, 1), the shuffle and sort phase will group it as:  
(word, [1, 1, 1])

**3. Reduce Phase:**

* The Reduce phase is the **final step** of the MapReduce model.
* It takes each key and the list of values grouped with it, and **applies aggregation or computation**.
* This phase produces the final output of the job.

**Example:**  
Input: (word, [1,1,1]) → Reducer calculates the sum = 3  
Output: (word, 3)

**Key Features of MapReduce Model:**

* **Parallel Processing:** Each task (map or reduce) runs on different nodes in parallel, increasing speed and efficiency.
* **Fault Tolerance:** If a node fails, the task can be re-assigned to another node.
* **Scalability:** It can handle petabytes of data by adding more machines.
* **Data Locality:** It processes data on the same machine where it is stored, reducing network usage.

**Simple Real-life Analogy:**

Think of MapReduce like organizing an exam paper-checking system:

* **Map phase:** Each teacher checks a batch of answer sheets and notes down marks of each student (key: student name, value: marks).
* **Shuffle and Sort:** All marks of a student from different teachers are grouped together.
* **Reduce phase:** A final checker adds up the total marks of each student and creates the final result sheet.

**Q. Explain Grouping and Aggregation algorithm using MapReduce. Support your answer with a suitable example.**

**Ans:**

Here's a clear and simple answer suitable for writing directly in your exam:

### ****Grouping and Aggregation Algorithm using MapReduce****

**Grouping and Aggregation** are key operations in Big Data processing. Grouping involves organizing data into sets based on a key, while aggregation computes a summary statistic (like count, sum, average) for each group. **MapReduce** efficiently handles both through its Map() and Reduce() phases.

### ****How It Works:****

#### ****1. Map Phase:****

* The **Map function** reads the input data and emits key-value pairs.
* The **key** represents the group identifier.
* The **value** is the data to be aggregated.

#### ****2. Shuffle and Sort Phase (Automatic):****

* Hadoop automatically groups all values that share the same key.
* This phase prepares grouped data for the Reducer.

#### ****3. Reduce Phase:****

* The **Reduce function** receives a key and a list of values.
* It performs aggregation (like counting, summing) and emits the final output.

### ****Example: Word Count (Classic Aggregation Example)****

Suppose we have the following input:

Input:

"A fox jumps"

"A dog jumps"

"A cat sleeps"

#### ****Map Function Output:****

(fox, 1)

(jumps, 1)

(dog, 1)

(jumps, 1)

(cat, 1)

(sleeps, 1)

#### ****Shuffle and Sort:****

(cat, [1])

(dog, [1])

(fox, [1])

(jumps, [1, 1])

(sleeps, [1])

#### ****Reduce Function Output (Grouped + Aggregated):****

(cat, 1)

(dog, 1)

(fox, 1)

(jumps, 2)

(sleeps, 1)

**Module 4**

**MU Q. Elaborate issues of stream processing (5 marks)**

**Ans:**

**Stream processing** is the technique of continuously processing data that is generated in real time. It is widely used in applications like fraud detection, sensor data analysis, online recommendations, and social media feeds.

Although stream processing offers high-speed and real-time capabilities, it comes with several **challenges and issues**:

**1. Handling High Velocity and Volume of Data**

* Stream processing systems must deal with a **continuous and fast inflow of data**.
* Managing large volumes at high speed requires a highly scalable and efficient architecture.
* If the system cannot keep up, it may **drop data or cause delays**.

**2. Fault Tolerance**

* Stream processing systems are expected to run **24/7 without interruption**.
* Any failure in nodes or network can result in **data loss** or **incomplete results**.
* Ensuring **fault-tolerant systems** that can recover without data corruption is difficult.

**3. Data Ordering**

* In a distributed environment, **data may arrive out of order**.
* Maintaining the correct sequence of events is important for accuracy, especially in financial or time-sensitive systems.
* Handling **out-of-order events** is a major challenge.

**4. State Management**

* Many stream applications require **maintaining state** (like user session data or previous counts).
* Managing state across nodes is difficult and needs **efficient storage and retrieval mechanisms**.
* It also requires periodic backups or checkpoints to avoid loss of state during failure.

**5. Latency Constraints**

* Stream processing is expected to deliver **results in near real-time**.
* Achieving **low latency** while ensuring accuracy and reliability can be challenging, especially when data is large or complex.

**6. System Complexity**

* Designing and maintaining a **distributed stream processing system** requires deep technical knowledge.
* Developers must handle everything from **data ingestion, windowing, and checkpointing** to **error handling and deployment**.

**7. Integration with External Systems**

* Stream processing systems often need to **connect with databases, message queues, or dashboards**.
* Ensuring smooth integration without affecting performance is another complex issue.

**8. Testing and Debugging**

* Since data flows continuously and cannot be paused, **testing and debugging** are harder than in batch systems.
* Developers must simulate real-time data to check for errors, which is time-consuming and sometimes inaccurate.

**MU Q. Explain issues in Data stream query processing**

**Ans:**

Here is a simple and clear explanation of the **issues in data stream query processing**, written in full sentences and suitable for exams:

### ****Issues in Data Stream Query Processing****

Data stream query processing is the technique of performing computations on continuous and high-speed data streams. Unlike traditional databases, data streams are unbounded, arrive rapidly, and cannot be stored entirely. Therefore, several issues arise:

#### ****1. Infinite Data Size****

* Data streams are continuous and potentially infinite, which makes it impossible to store all the data.
* Queries must be processed on-the-fly using limited memory and time.

#### ****2. Limited Memory****

* Due to the continuous nature of streams, only a small portion of the data can be stored in memory.
* Algorithms must be memory-efficient and work on summaries or samples of data.

#### ****3. Real-time Requirements****

* Stream data often needs real-time or near real-time processing.
* Query processing must be quick and delay-sensitive, especially in applications like fraud detection or traffic monitoring.

#### ****4. Data Arrival Order****

* Data may not always arrive in order. Out-of-order data can affect the correctness of results.
* Handling late-arriving data and ensuring correct results becomes a challenge.

#### ****5. Multiple Continuous Queries****

* Many applications require running multiple continuous queries simultaneously.
* Managing query efficiency and system resources becomes difficult when queries compete for memory and processing power.

#### ****6. Variable Data Rates****

* The speed at which data arrives can change over time (called burstiness).
* The system must be able to handle both slow and extremely fast data rates efficiently.

#### ****7. Approximate Query Processing****

* Due to time and memory constraints, exact answers may not always be possible.
* Stream systems often return approximate results using algorithms like sampling, sketching, or windowing.

#### ****8. Fault Tolerance and Reliability****

* Since data is transient, any system failure can cause data loss.
* Ensuring reliability and recovery from failure is a major concern in stream processing.

**MU Q. Explain DGIM algorithm**

**Ans:**

Here is a clear and simple explanation of the **DGIM algorithm**, suitable for writing in your exam in full sentences:

**DGIM Algorithm (Datar–Gionis–Indyk–Motwani Algorithm)**

The **DGIM algorithm** is a space-efficient method used for **counting the number of 1’s** in the **most recent N bits** of a **data stream**. It is useful when we cannot store the entire stream due to memory constraints and need approximate answers using less space.

**Purpose of DGIM:**

* To estimate how many 1’s occurred in the last **k bits** of a binary stream (like 0, 1, 0, 1, 1, 0...).
* Uses logarithmic space instead of storing the entire stream.
* Allows approximate counting with **guaranteed error bounds**.

**How the DGIM Algorithm Works:**

1. **Input Stream:**
   * The algorithm works on a binary stream (only 0s and 1s).
   * For example: ... 0, 1, 1, 0, 1, 0, 0, 1
2. **Buckets:**
   * A **bucket** is a group of 1’s that have occurred in the stream.
   * Each bucket has two values:
     + The **timestamp** of the last bit in the bucket.
     + The **size** of the bucket (number of 1’s it represents).
   * Sizes of buckets are powers of two: 1, 2, 4, 8, etc.
3. **Rules for Buckets:**
   * At most **two buckets** of the same size are allowed at any time.
   * When a third bucket of the same size appears, two oldest buckets are **merged** to form one new bucket of **double the size**.
4. **Estimation:**
   * To estimate the number of 1’s in the last **k bits**, count:
     + The **sizes of all buckets** that are **completely within** the last k bits.
     + Add **half the size** of the oldest bucket (partially within the window) as an estimate.

**Example:**

Suppose you want to count how many 1’s were seen in the last 10 bits.

* Buckets:
  + Size 1 → appears at positions 9 and 7
  + Size 2 → appears at position 5
  + Size 4 → appears at position 2

You count:

* Size 1 (position 9): 1
* Size 1 (position 7): 1
* Size 2 (position 5): 2
* Size 4 (position 2): only **half counted** (as it may not be fully inside window): 2

Estimated total: **1 + 1 + 2 + 2 = 6 ones** in last 10 bits.

**Advantages of DGIM:**

* Uses **logarithmic space** to represent stream history.
* Provides **fast, approximate** answers.
* Good for data streams where exact storage is not possible.

**MU Q. Explain the DGIM algorithm. State the rules used in DGIM that must be followed**

**Ans**: The **DGIM (Datar-Gionis-Indyk-Motwani)** algorithm is a streaming algorithm used to **approximate the number of 1’s** in the **last *k* bits** of a **binary data stream**. It is designed to solve the problem of **space-efficient counting** in situations where data arrives continuously, and it is not possible to store the entire stream.

Instead of storing all bits, DGIM represents recent parts of the stream using **"buckets"**, which help in approximating the count with **limited memory** and **controlled error**.

**Main Idea of DGIM:**

* DGIM groups recent 1’s into **buckets** of various sizes, all in **powers of 2**.
* Each bucket stores:
  1. The **size** of the bucket (how many 1’s it represents).
  2. The **timestamp** of the **last bit** in that bucket.

**Example of Bucket Sizes:**

* Size 1 (represents 1 one)
* Size 2 (represents 2 ones)
* Size 4 (represents 4 ones), and so on.

**Rules of DGIM That Must Be Followed:**

1. **Bucket Size Rule:**
   * Each bucket must contain only **1’s**, and its size must always be a **power of 2**.
2. **Ordering Rule:**
   * Buckets are **ordered** by the time they appeared in the stream (oldest to newest).
   * This allows the algorithm to keep track of how recent the data is.
3. **Bucket Merge Rule:**
   * For any given size, there can be **at most two buckets** of that size at any time.
   * If a **third bucket** of the same size is created, the **two oldest buckets** are **merged** into a single new bucket of **double the size**.
   * The timestamp of the merged bucket is set to the **latest** of the two merged ones.
4. **Estimation Rule:**
   * To **estimate** the number of 1’s in the **last *k* bits**:
     + Count the sizes of all buckets **fully within** the *k* window.
     + Add **half the size** of the **oldest partially included** bucket to account for estimation.

**Advantages of DGIM:**

* Only uses **logarithmic space** (O(log² N)), where N is the length of the stream.
* Guarantees **approximation within a small error margin**.
* Works very well for large or unbounded binary streams.

**MU Q. Explain DGIM algorithm for counting ones in a stream with example. [May 2024]**

**Ans:** The **DGIM algorithm**, introduced by Datar, Gionis, Indyk, and Motwani, is a space-efficient technique used to **estimate the number of 1’s in the most recent *k* bits** of a **binary data stream**. It is particularly useful in **streaming environments**, where it is not feasible to store the entire stream in memory.

**Objective:**

To **approximate the count of 1’s** in the last *k* bits of a binary stream, using **limited space** while maintaining an **acceptable error margin**.

**Basic Concept:**

* The algorithm **groups 1’s into buckets** of size that are **powers of 2** (1, 2, 4, 8...).
* Each **bucket** represents a group of **consecutive 1’s** and stores:
  1. The **size** (number of 1’s).
  2. The **timestamp** of the **last 1** in that bucket.
* The number of buckets is kept **bounded** using a **merging rule**, which helps reduce memory usage.

**Rules of DGIM Algorithm:**

1. **Each bucket size is a power of 2.**
2. **At most two buckets** of the same size are allowed at any time.
3. When a third bucket of the same size is created, the **two oldest buckets** are **merged** into a new bucket of **double the size**.
4. Buckets are stored in **reverse chronological order** (most recent to oldest).

**Example:**

Let’s say we have the following binary stream of the last 16 bits (the newest bit is on the right):

Stream: 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1

Bit Position: 1 (oldest) --------> 16 (newest)

We apply DGIM to count the number of **1’s** in the **last *k* = 10 bits** (positions 7 to 16).

1. **Step 1:** Go from right to left, and form buckets for each 1:
   * Start placing 1’s into size 1 buckets.
   * Whenever there are 3 buckets of the same size, merge the **two oldest**.
2. **Resulting Buckets (for last 10 bits):**
   * Size 4 (ending at position 13)
   * Size 2 (ending at position 15)
   * Size 1 (ending at position 16)
3. **Estimation:**
   * Count **full buckets**: 4 (size 4), 2 (size 2) → Total = 6
   * **Oldest bucket (size 1)** is only **partially** inside the last 10 bits.  
     So, we add **half of it** = 0.5
   * **Estimated count of 1’s = 6 + 0.5 = 6.5**

**Advantages:**

* **Memory-efficient**: Requires only **O(log² N)** space for a stream of length N.
* Allows **approximate counting** with **small and controlled error**.
* Suitable for **real-time analytics**, such as in **network monitoring** or **sensor data**.

**MU Q. Summarize Bloom’s filter with example and its applications**

**Ans:**

A **Bloom filter** is a **probabilistic data structure** used to **test whether an element is a member of a set**. It is highly **space-efficient** and allows **fast lookups**, but it may produce **false positives**, meaning it might tell us an element is present even when it is not. However, **false negatives are not possible**, so if the filter says an element is not present, that answer is always correct.

**Working of Bloom Filter:**

1. A Bloom filter uses a **bit array of size *m***, all initialized to 0.
2. It also uses ***k* independent hash functions**.
3. When an element is added to the set:
   * It is passed through all *k* hash functions.
   * Each hash function maps the element to a position in the bit array, and those positions are set to 1.
4. To check if an element is in the set:
   * The element is hashed using the same *k* hash functions.
   * If all the corresponding positions in the bit array are 1, the element is **possibly present**.
   * If **even one bit** is 0, the element is **definitely not present**.

**Example:**

Let’s say we want to store the set {cat, dog, cow} in a Bloom filter using 3 hash functions and a bit array of size 10.

1. Add “cat”: Assume it sets bits at positions 1, 4, and 7.
2. Add “dog”: Sets bits at positions 2, 4, and 9.
3. Add “cow”: Sets bits at positions 1, 3, and 8.

Now the bit array might look like this:

Index: 0 1 2 3 4 5 6 7 8 9

Bit: 0 1 1 1 1 0 0 1 1 1

To check if "rat" is in the set:

* It is hashed using the same 3 hash functions.
* If any of the 3 corresponding bit positions is 0, "rat" is **definitely not** in the set.
* If all are 1, we say "rat" is **possibly** in the set.

**Applications of Bloom Filter:**

1. **Databases and Caches:**
   * Used in systems like **Cassandra** and **Redis** to avoid unnecessary disk lookups.
   * Helps in quickly checking if a value might exist before actually searching the database.
2. **Spell Checkers:**
   * Used to check whether a word exists in the dictionary.
3. **Web Crawlers:**
   * Used to detect whether a URL has been visited before or not.
4. **Network Security:**
   * Applied in intrusion detection systems to detect malicious addresses.
5. **Distributed Systems:**
   * Used in systems like **Apache HBase** and **Google Bigtable** for fast lookup operations.

**MU Q. Give two applications for counting the number of 1’s in a long stream of binary values. Using a stream of binary digits, illustrate how DGIM will find the number of 1’s.**

**Ans:**

### ****Applications:****

1. **Network Monitoring:**  
   In internet traffic analysis, each bit in the stream can represent whether a specific type of event (like a packet drop or an error) occurred or not. Counting the number of 1’s helps track how many such events occurred recently.
2. **Clickstream Analysis in Web Applications:**  
   Each bit may represent whether a user clicked on an ad (1) or not (0). Counting 1’s in recent history gives an estimate of recent user activity or engagement.

### ****DGIM Algorithm: (Datar-Gionis-Indyk-Motwani Algorithm)****

The **DGIM algorithm** is designed to **estimate the number of 1's** in the **last k bits** of a **binary stream** using **very little memory**.

Instead of storing every bit, it stores **buckets** containing information about the number of 1’s and their **approximate position** in the stream.

### ****Rules of DGIM:****

1. Each bucket represents **2ⁿ number of 1’s** (e.g., 1, 2, 4, 8...).
2. Each bucket stores:
   * **Size** (number of 1’s)
   * **Timestamp** (the position of the last 1 in that bucket)
3. **No more than two buckets** of the same size are allowed.
4. The algorithm merges buckets when there are more than two of the same size.

### ****Example:****

Let’s consider a binary stream coming in like this (rightmost is most recent):

... 1 0 1 1 0 1 0 1 1 1

We are interested in counting the **number of 1’s in the last 10 bits**.

#### ****Step-by-step DGIM Bucket Formation:****

We read bits from right to left (latest to oldest):

* Bit 10 (latest): 1 → Bucket of size 1, time 10
* Bit 9: 1 → Bucket of size 1, time 9
* Now we have two buckets of size 1 → ok
* Bit 8: 1 → Bucket of size 1, time 8
  + Now three buckets of size 1 → merge two oldest → one bucket of size 2, latest time 9
* Bit 7: 0 → ignore
* Bit 6: 1 → Bucket of size 1, time 6
* Bit 5: 0 → ignore
* Bit 4: 1 → Bucket of size 1, time 4
* Bit 3: 1 → Bucket of size 1, time 3
  + Three of size 1 → merge two oldest → bucket of size 2, time 3
* Bit 2: 0 → ignore
* Bit 1: 1 → Bucket of size 1, time 1

#### ****Final Buckets (after processing 10 bits):****

* Size 1, time 10
* Size 2, time 9
* Size 1, time 6
* Size 2, time 3
* Size 1, time 1

Now, to estimate the number of 1’s in the last 10 bits:

* Add all bucket sizes except the **oldest one** fully.
* For the **oldest bucket**, we **add only half** of its size (to avoid overestimation).

**Sum = 1 + 2 + 1 + 2 + (½ × 1) = 6.5 → approx 7 ones**

**Q. Flajolet-Martin Algorithm: [May 2024]**

**Ans:**

**Flajolet-Martin Algorithm**  
The Flajolet-Martin (FM) algorithm is a **probabilistic algorithm** used to **estimate the number of distinct elements (cardinality)** in a data stream. This method is very useful in **big data scenarios** where storing and comparing every item is computationally expensive or impossible.

**1. Purpose**

* The FM algorithm is used to estimate the **number of unique elements** in a stream without storing all elements.

**2. Working Principle**

a. **Hashing elements**:

* Each element in the stream is passed through a **hash function** that converts it into a binary string.

b. **Tracking trailing zeros**:

* The binary hash is observed for the **position of the rightmost 1-bit** or **number of trailing zeros**.
* The idea is that **more trailing zeros indicate a rarer event**, which likely comes from a larger dataset.

c. **Recording maximum trailing zeros**:

* The algorithm keeps track of the **maximum number of trailing zeros** observed in the hash values.

d. **Estimate**:

* If the maximum number of trailing zeros is R, then the **estimated number of distinct elements** is approximately:

2R2^R

**3. Example**

Suppose a stream of elements: A, B, C, A, C, D

* Hash values (binary assumed for simplicity):
  + A → 0100 → 2 trailing zeros
  + B → 1000 → 3 trailing zeros
  + C → 0010 → 1 trailing zero
  + D → 0001 → 0 trailing zeros
* Maximum number of trailing zeros = 3 (from B)
* Estimate = 23=82^3 = 8

Actual number of distinct elements = 4 (A, B, C, D)  
Estimate is not exact but **close**. To improve accuracy, **multiple hash functions** and **averaging techniques** can be used.

**4. Advantages**

* Uses **very little memory**
* Suitable for **real-time streaming**
* **Fast computation**

**5. Limitations**

* It is **probabilistic**, not exact.
* Accuracy depends on the **quality of the hash function** and use of **multiple hash functions**.

**6. Applications**

* Network traffic analysis
* Search engines to count distinct queries
* Online advertising platforms to count unique visitors
* Big data systems to analyze unique users or events in logs.

**Module 5**

**Q. K-Nearest Neighbors (KNN) Algorithm [May 2024]**

**Ans:**

K-Nearest Neighbors (KNN) is a **supervised machine learning algorithm** used for **classification and regression** tasks. It predicts the value or class of a data point by looking at the **‘k’ nearest data points** in the training set.

**2. Working Principle**

a. **Choose the value of K** (number of neighbors to consider).  
b. **Calculate the distance** between the query point and all points in the training data using a distance metric (like **Euclidean, Manhattan**, etc.).  
c. **Select the K nearest neighbors** to the query point based on the shortest distance.  
d. **Voting or Averaging**:

* **For classification**: Take the **majority class** among the K neighbors.
* **For regression**: Take the **average** of the K neighbors' values.

**3. Distance Metrics**

* **Euclidean Distance**:



* **Manhattan Distance**:



**4. Example (Classification)**

Suppose you have data about animals based on weight and height, and you want to classify a new animal as a dog or cat.  
You choose k = 3.

* If out of the 3 nearest animals, 2 are dogs and 1 is a cat → the algorithm classifies the new animal as a **dog**.

**5. Advantages**

* **Simple and easy to implement**
* **No training phase** (lazy learner)
* **Works well with smaller datasets**

**6. Limitations**

* **Computationally expensive** on large datasets
* **Sensitive to irrelevant or scaled features**
* **Does not work well with high-dimensional data (curse of dimensionality)**

**7. Applications**

* Recommendation systems
* Handwriting recognition
* Fraud detection
* Medical diagnosis
* Pattern recognition systems

**MU \*\*Q. Explain PCY algorithm and its 2 types with neat labelled diagram or Q. Explain clearly with diagrams the PCY method of finding frequent itemsets (pairs) in a large data set. (see chatgpt)**

**Ans:**

The **PCY (Park–Chen–Yu) algorithm** is used for finding **frequent item pairs** in large datasets.

It is an improvement over the **Apriori algorithm**, designed to **reduce memory usage and overall computational cost** by using a hash-based technique to identify frequent pairs.

The main goal of the PCY algorithm is to **minimize the number of candidate pairs** generated in the second pass by using **bitmaps and hashing**.

It is commonly used in data mining and association rule learning.

Applications of the PCY algorithm include market basket analysis, recommendation system, and web log analysis.

### ****Working of the PCY Algorithm (Two Passes)****

#### ****Pass 1:****

* Count the **support of individual items** (like Apriori).
* Also, for each pair of items in a basket, compute a **hash value** and count the number of pairs that hash to each bucket.
* Create a **bitmap** where a bit is set to 1 if the bucket count is ≥ support threshold.

#### ****Pass 2:****

* Generate **candidate pairs** only if:
  + Both items are individually frequent.
  + The hash bucket they map to in the bitmap is marked as frequent.
* This reduces the number of pairs we need to count.

### ****Types of PCY Algorithm****

#### ****Type 1: Basic PCY Algorithm****

* Uses **only one hash table and one bitmap** in the first pass.
* Efficient for moderately sized datasets.
* Works well when memory can hold a single bitmap.

#### ****Type 2: Multi-Stage PCY Algorithm****

* Extends basic PCY by using **multiple hash tables and bitmaps**.
* After the first bitmap is created, a second hash table is created using **only frequent pairs** from the first one.
* This further **refines candidate selection** and improves accuracy for large datasets.

**Algorithm:**

**Pass 1:**

**FOR (each basket):**

**FOR (each item in the basket):**

**Add 1 to item’s count**

**FOR (each pair of items):**

**Hash the pair to a bucket**

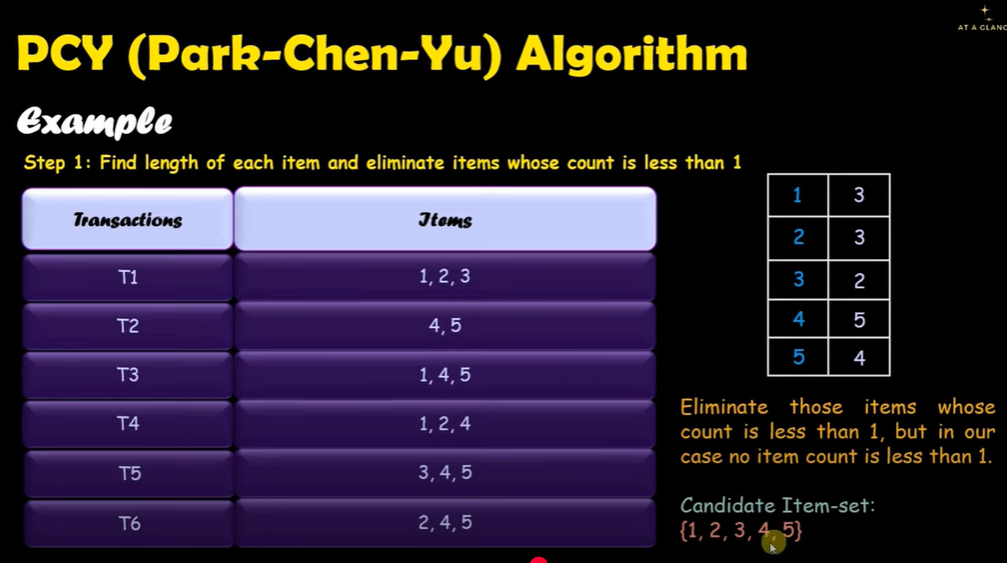
**Add 1 to the count for that bucket**

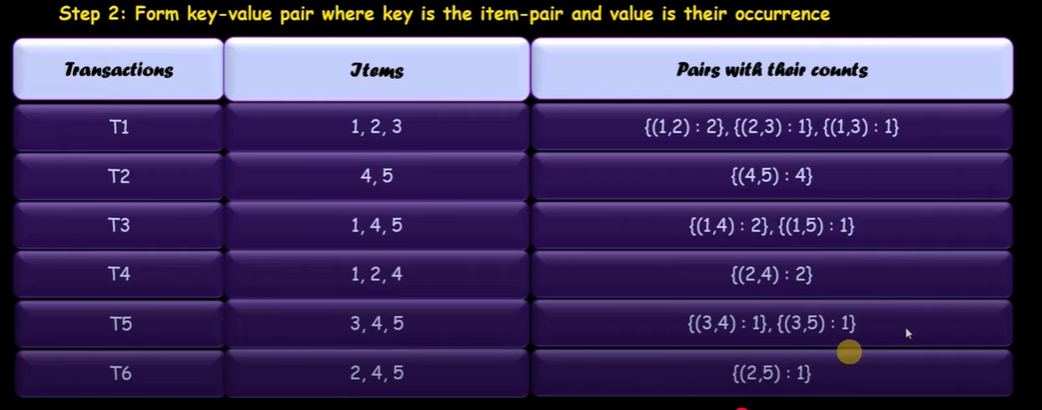
**Pass 2:**

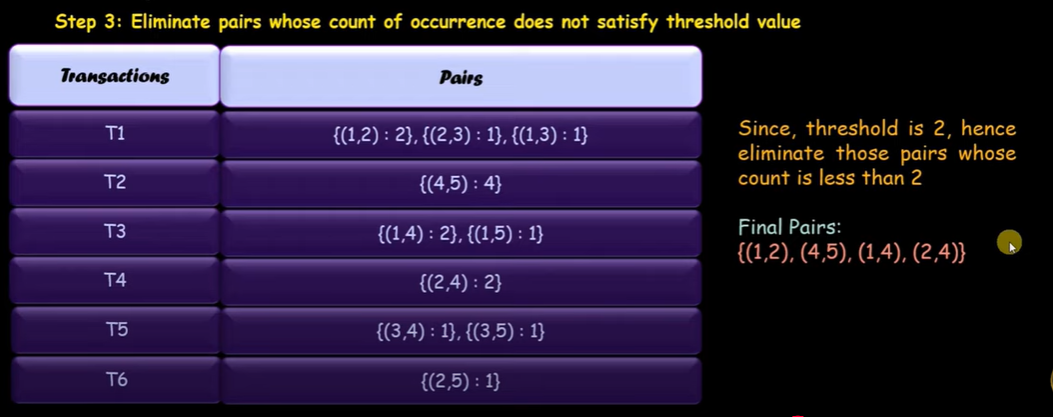
**Count all pairs {i, j} that meet the conditions for being a candidate pair:**

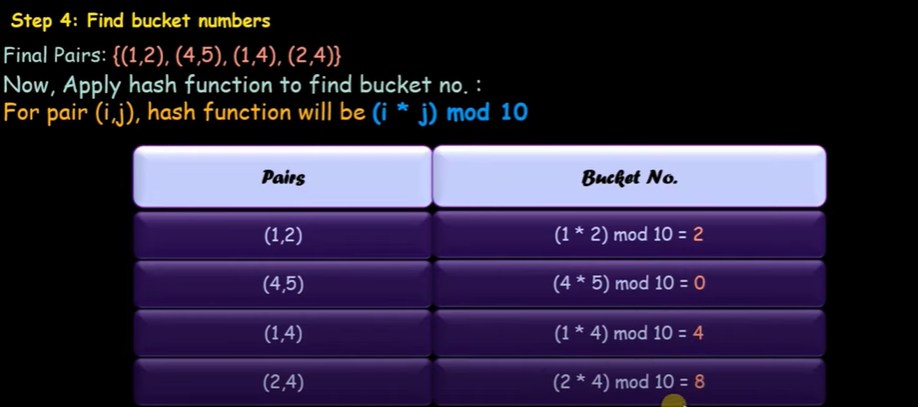
* 1. **Both I and j are frequent items**
  2. **The pair {i, j} hashes to a bucket whose bit in the bit vector is 1**

**Example:**









**MU \*\*Q. Explain CURE algorithm. [May 2024]**

**Ans:**

The **CURE (Clustering Using Representatives)** algorithm is a hierarchical clustering algorithm designed to handle large datasets effectively. It is particularly useful in identifying clusters in data with **non-spherical shapes** and **outliers**.

**Key Features of CURE Algorithm:**

1. **Robust to Outliers**: CURE is designed to handle datasets with noise and outliers effectively.
2. **Uses Representative Points**: Instead of using the centroid (like in K-means), CURE uses **multiple representative points** for each cluster to define its shape.
3. **Scalable**: CURE works well for large datasets, as it reduces the need to examine every data point repeatedly.

**Steps of the CURE Algorithm:**

1. **Choose Representative Points**:
   * For each cluster, the CURE algorithm selects a set of representative points. These points are chosen such that they are spread out and capture the shape of the cluster.
   * Typically, **k representative points** are chosen for each cluster, where **k** is a parameter set by the user.
2. **Shrink Points Toward Centroid**:
   * The representative points are **shrunk toward the centroid** of the cluster. The amount of shrinking depends on a parameter that controls the degree of shrinking.
   * This step helps in making the points closer to the center of the cluster, simplifying the cluster's representation.
3. **Cluster the Representative Points**:
   * The algorithm then **clusters** the representative points of the data using a **hierarchical clustering approach** (e.g., agglomerative clustering).
   * The distances between clusters are calculated using the distance between their representative points.
4. **Merge Clusters**:
   * The clusters are merged iteratively based on the distance between their **representative points**. Clusters with the smallest distance between their representative points are merged first.
5. **Final Clustering**:
   * The output of the algorithm is a **hierarchical tree** or **dendrogram**, where clusters are progressively merged based on the representative points.

**Advantages of CURE Algorithm:**

1. **Handles Non-Spherical Clusters**: CURE can efficiently handle clusters of various shapes, unlike algorithms such as K-means that assume spherical clusters.
2. **Robust to Outliers**: By using representative points and shrinking them toward the centroid, the algorithm reduces the impact of outliers.
3. **Scalable**: Since it works with representative points, it can handle large datasets more efficiently than algorithms like K-means, which require frequent distance computations between all data points.

**Disadvantages of CURE Algorithm:**

1. **Parameter Sensitivity**: The number of representative points and the shrinking factor must be chosen carefully. If these parameters are chosen incorrectly, it can lead to poor clustering results.
2. **Computational Complexity**: Although it is scalable, the algorithm can still be computationally expensive for very large datasets with a high number of clusters.

**Example:**

Let’s say we have a dataset of customers with various features like age and spending. Using the CURE algorithm, we can:

1. Choose representative points for each customer group (cluster).
2. Shrink those points toward the centroid of each group.
3. Cluster the groups based on their representative points and merge the clusters based on similarity.

The output would be a hierarchical structure that shows how customers can be grouped based on their similarities.

**MU Q. Explain CURE algorithm, clearly stating its advantages over traditional clustering algorithm.**

**Ans:**

The **CURE (Clustering Using Representatives)** algorithm is an advanced hierarchical clustering method designed to overcome limitations of traditional clustering algorithms, particularly in handling large datasets with complex and non-spherical cluster shapes. CURE is robust to outliers, works with clusters of different shapes, and can handle large datasets more efficiently than many traditional clustering techniques.

### ****CURE Algorithm Overview****

The CURE algorithm uses a **hierarchical clustering approach**, but it enhances it by selecting multiple **representative points** for each cluster, which allows it to capture the shape of the cluster more accurately. The key steps of the CURE algorithm are as follows:

#### ****1. Choosing Representative Points:****

* For each cluster, CURE selects **multiple representative points** that capture the cluster’s shape.
* These points are chosen from the data points within the cluster, such that they are spread out across the cluster, capturing the overall structure.

#### ****2. Shrinking the Representative Points Toward the Centroid:****

* The representative points are **shrunk toward the centroid** of the cluster. This helps to minimize the impact of extreme points (outliers) and brings the points closer to the center.
* The degree of shrinking is controlled by a parameter that specifies how much the points should move toward the centroid.

#### ****3. Clustering the Representative Points:****

* Once the representative points are selected and shrunk, they are used to perform **hierarchical clustering**. This process involves computing the distance between the representative points of different clusters and progressively merging clusters based on these distances.
* The hierarchical structure (dendrogram) is built based on the merging of clusters with the smallest distances between their representative points.

#### ****4. Merging Clusters:****

* The clusters are merged iteratively. At each step, the algorithm merges the clusters whose representative points are closest to each other.
* The merging continues until the desired number of clusters is reached.

### ****Advantages of the CURE Algorithm Over Traditional Clustering Algorithms****

#### ****1. Handles Non-Spherical Clusters:****

* Traditional clustering algorithms like **K-means** assume that clusters are **spherical** and have similar sizes. This assumption works well when clusters are round, but it fails when clusters have irregular shapes or varying densities.
* **CURE** handles **non-spherical clusters** effectively by using representative points that can capture the **shape and structure** of the clusters, even if they are elongated, curved, or have different densities.

#### ****2. Robust to Outliers:****

* In traditional clustering algorithms like **K-means**, outliers can heavily influence the centroid, distorting the cluster structure.
* **CURE** addresses this issue by selecting multiple representative points for each cluster and **shrinking them toward the centroid**, which minimizes the effect of outliers. Outliers have a much smaller effect on the overall structure because the shrinking process reduces their influence.

#### ****3. More Accurate Cluster Representation:****

* In algorithms like **K-means**, a single centroid represents each cluster, which is insufficient when the cluster is spread out or non-convex.
* **CURE** uses multiple representative points to capture the overall shape of the cluster, leading to a **more accurate representation** of the cluster's boundaries. This is particularly helpful in complex datasets where clusters may not be simple circular or spherical shapes.

#### ****4. Hierarchical Clustering Approach:****

* Traditional methods like **K-means** require specifying the number of clusters in advance. This can be challenging if the optimal number of clusters is not known.
* **CURE** uses a **hierarchical clustering approach**, which builds a **dendrogram** of clusters. This means you don’t need to specify the number of clusters upfront, and you can observe how clusters are progressively merged based on their similarity. This provides more flexibility and insight into the clustering process.

#### ****5. Scalable for Large Datasets:****

* While traditional hierarchical clustering algorithms can be **computationally expensive**, especially for large datasets, **CURE** is **more scalable** because it works with a limited number of representative points rather than all data points. By focusing on the representative points, CURE reduces the amount of data it needs to process, making it more efficient for large datasets.

#### ****6. No Assumption on Cluster Size:****

* Traditional clustering algorithms like **K-means** tend to perform poorly when clusters vary in size, as they use a fixed number of clusters (centroids).
* **CURE** does not make assumptions about the size or shape of clusters and can handle clusters with varying densities and sizes, providing better results in real-world datasets.

**Module 6**

**Q. PageRank algorithm (5 marks)**

**Ans:**

Here is a clear, full-sentence explanation of the **PageRank algorithm**, suitable for writing directly in your exam:

**Q. Explain PageRank Algorithm**

The **PageRank algorithm** is a method developed by **Larry Page and Sergey Brin**, the founders of Google, to rank web pages in search engine results. It assigns a **numerical value (score)** to each web page, which represents the importance of that page within the set of all web pages.

**Concept:**

The key idea behind PageRank is that a **web page is important if it is linked to by many other important pages**. Simply having many incoming links (in-links) increases a page’s rank, but the quality of those links also matters.

**How It Works:**

1. **Each page is given an initial PageRank** value, usually 1 or 1/N, where N is the total number of pages.
2. The PageRank of a page is **shared equally among all the pages it links to**.
3. The PageRank of each page is updated in **multiple iterations** using the following formula:

PR(A)=(1−d)+d(PR(T1)C(T1)+PR(T2)C(T2)+⋯+PR(Tn)C(Tn))PR(A) = (1 - d) + d \left( \frac{PR(T1)}{C(T1)} + \frac{PR(T2)}{C(T2)} + \cdots + \frac{PR(Tn)}{C(Tn)} \right)

Where:

* + **PR(A)** is the PageRank of page A.
  + **d** is the damping factor (usually 0.85), which accounts for the probability that a user continues clicking links.
  + **T1, T2,..., Tn** are pages that link to A.
  + **C(Ti)** is the number of links on page Ti.

1. The formula is applied **iteratively** until the PageRanks converge (i.e., stop changing significantly).

**Example:**

Suppose there are three pages: A, B, and C.

* Page A links to B
* Page B links to C
* Page C links back to A and B

The algorithm will compute the PageRank of each page based on how they link to each other. If a highly ranked page links to a low-ranked one, it increases that page's rank.

**Damping Factor:**

The **damping factor** (usually 0.85) is used to simulate the probability that a user will continue clicking on links. The remaining 0.15 (i.e., 1 - d) accounts for a random jump to any page, ensuring that even pages with no in-links receive some score.

**Importance and Applications:**

* PageRank is widely used in **search engines** to determine the **relevance** and **authority** of web pages.
* It is also used in **academic citation networks**, **social media influence analysis**, and **biological networks**.

**MU Q. Modified PageRank algorithm (5 marks)**

**Ans:**

The **PageRank algorithm** was originally developed by Google to rank web pages based on their importance. It assigns a numerical score to each page depending on how many other pages link to it and how important those linking pages are.

However, in real-world applications, the **basic PageRank** may not always perform well in specialized scenarios like social networks, academic citations, or personalized search. Therefore, **Modified PageRank algorithms** were introduced to improve the performance by considering additional factors.

**Key Modifications in the Modified PageRank Algorithm:**

1. **Topic-Sensitive PageRank:**
   * In this version, PageRank is computed with respect to a specific **topic or category**.
   * It uses a set of **topic-specific "teleportation" pages** instead of jumping to random pages.
   * It is more relevant in personalized or filtered searches where results are related to a specific interest.
2. **Personalized PageRank:**
   * It modifies the teleportation factor to always jump to a **personalized set of pages** based on the user's history or preferences.
   * This makes the search results more tailored and meaningful to each individual user.
3. **Weighted PageRank:**
   * In this model, **not all links are treated equally**.
   * A weight is assigned to each incoming or outgoing link based on certain criteria like click frequency or trust level.
   * The more important or trusted the link is, the more it contributes to the page’s final score.
4. **Time-Aware PageRank:**
   * This modification gives **higher importance to recent links** rather than older ones.
   * It is useful in dynamic environments like news websites or social media, where recent content is more valuable.
5. **TrustRank:**
   * This version helps fight **web spam**.
   * A small set of **trusted pages** is chosen manually, and the PageRank algorithm is biased towards these pages.
   * It spreads "trust" through the link structure, and spam pages receive low scores.

**Applications of Modified PageRank:**

* Search engines for **more relevant and trustworthy results**.
* Social network analysis for identifying **influential users**.
* Recommendation systems for **personalized content**.
* Academic citations to measure **research paper impact**.

**MU Q. Explain the collaborative filtering based recommendation system: (5 marks)**

**Ans:**

**Collaborative filtering** is a popular technique used in recommendation systems to suggest items (like products, movies, books, etc.) to users based on their past behavior and the preferences of other similar users.

It works on the principle that **users with similar interests in the past will likely prefer similar items in the future**.

**How Collaborative Filtering Works:**

1. **User behavior data is collected** – such as ratings, clicks, purchases, or likes.
2. The system identifies **patterns or similarities** among users or items.
3. Recommendations are made based on what **similar users liked or rated highly**, even if the user has not interacted with those items yet.

**Types of Collaborative Filtering:**

1. **User-Based Collaborative Filtering:**
   * This method finds users who are similar to the target user based on past activities.
   * It then recommends items that those similar users liked.
   * **Example:** If User A and User B both liked the same books, and User A liked a new book, then that book may be recommended to User B.
2. **Item-Based Collaborative Filtering:**
   * This method finds items that are similar to items the user has liked before.
   * It recommends items that are often liked together by other users.
   * **Example:** If many users who liked "Movie X" also liked "Movie Y", then a user who watched "Movie X" may be recommended "Movie Y".

**Advantages:**

* Does not require detailed information about the items.
* Can discover complex patterns in user preferences.
* Learns from real user interactions rather than static rules.

**Limitations:**

* Suffers from the **cold start problem** (when new users or items have little data).
* Can become **less accurate** if the dataset is sparse (few users or ratings).
* May struggle with **scalability** when user numbers are very high.

**MU \*\*Q. Explain collaborative filtering system. How is it different from content based system.**

**Ans:**

### ****Collaborative Filtering System:****

* A **Collaborative Filtering system** is a type of recommendation system that suggests items to a user based on the preferences and actions of **other users**.
* It assumes that if two users liked the same items in the past, they are likely to like similar items in the future.
* The system relies on the concept of **user-user similarity** or **item-item similarity** without needing to understand the content of the items.

#### ****Types of Collaborative Filtering:****

1. **User-Based Collaborative Filtering:**
   * Recommends items by finding users with similar preferences.
   * Example: If User A and User B liked the same products before, products liked by User A but not yet seen by User B will be recommended to User B.
2. **Item-Based Collaborative Filtering:**
   * Recommends items based on similarity between items.
   * Example: If most users who watched Movie X also watched Movie Y, then a user who watched Movie X will be recommended Movie Y.

### ****Content-Based Filtering System:****

* A **Content-Based Filtering system** recommends items by analyzing the **features of the items** and the **user’s previous choices**.
* It does not consider other users but focuses on the **attributes** of items (like genre, price, category, etc.) that the user has liked in the past.
* The system builds a profile for each user and recommends items that match the user's profile.

#### ****Example:****

If a user has liked action movies in the past, the system will recommend more action movies, even if no other users have watched them.

**MU Q. Explain HITS algorithm**

**Ans:**

**HITS** stands for **Hyperlink-Induced Topic Search**. It is a link analysis algorithm used to rank web pages based on their relevance and connection to a particular search query. The HITS algorithm was introduced by **Jon Kleinberg** in 1999.

**Basic Idea:**

* The HITS algorithm works on the concept of identifying **two types of pages** in a web network:
  1. **Authorities** – These are pages that contain valuable content related to a topic.
  2. **Hubs** – These are pages that link to multiple authority pages.
* A good **hub** page points to many **authorities**, and a good **authority** is linked by many good **hubs**.

**How the HITS Algorithm Works:**

1. **Input:** A query is entered, and a small web subgraph related to the query is created.
2. **Initialization:** Each page in the subgraph is assigned two scores:
   * An **authority score** (initially 1)
   * A **hub score** (initially 1)
3. **Iteration:** The algorithm updates these scores as follows:
   * A page's **authority score** becomes the sum of the **hub scores** of all pages linking to it.
   * A page's **hub score** becomes the sum of the **authority scores** of all pages it links to.
4. **Normalization:** After each update, scores are normalized to keep them from growing indefinitely.
5. **Convergence:** This process repeats until the scores stabilize.

**Mathematical Representation (Simple Explanation):**

* Let A(p) be the authority score of page p, and H(p) be the hub score.
* Update rules:
  + **A(p) = sum of H(q)** for all pages q that link to p
  + **H(p) = sum of A(r)** for all pages r that p links to

**Example:**

* Suppose there is a page that lists the best research articles (a hub).
* It links to actual research articles (authorities).
* The HITS algorithm increases the hub score of the list page and the authority score of the research articles.

**Applications of HITS:**

* Ranking results in search engines (less commonly now).
* Identifying important web pages on a specific topic.
* Social network and citation network analysis.

**Limitations of HITS:**

* Sensitive to **spam links**, as false hubs and authorities can be created.
* Requires creating a **query-dependent subgraph**, which can be computationally expensive.
* Not as scalable as PageRank for very large datasets.

**MU Q. Comment on usefulness of different types of recommendation system in real life with example. [May 2024]**

**Ans:**

Recommendation systems play an important role in guiding users to relevant content, products, or services. They help in improving user experience, increasing sales, and saving time by filtering out unnecessary information.

**1. Collaborative Filtering:**

* **Usefulness:**  
  This system recommends items to a user based on the preferences of other users with similar interests.
* **Example:**  
  On **Netflix**, if you watch crime thrillers and another user with a similar history liked a specific show, that show will be recommended to you.
* **Real-Life Impact:**  
  It helps in discovering items that a user may not find on their own, based purely on shared user behavior.

**2. Content-Based Filtering:**

* **Usefulness:**  
  This system recommends items similar to what the user has already liked or viewed, based on item attributes.
* **Example:**  
  On **Spotify**, if you listen to slow piano music, it recommends other songs with the same instruments or mood.
* **Real-Life Impact:**  
  It is highly personalized and useful for people with niche or unique tastes.

**3. Hybrid Recommendation System:**

* **Usefulness:**  
  This system combines collaborative and content-based filtering to give more accurate and balanced suggestions.
* **Example:**  
  On **Amazon**, it suggests products based on both your browsing/purchase history (content-based) and what others bought (collaborative).
* **Real-Life Impact:**  
  It overcomes the limitations of using only one method and improves recommendation quality.

**4. Demographic-Based Recommendation:**

* **Usefulness:**  
  This system recommends items based on user’s age, gender, location, and other demographic details.
* **Example:**  
  An **e-learning platform** might recommend coding courses to college students and business strategy courses to working professionals.
* **Real-Life Impact:**  
  It allows services to target user groups more effectively, especially for marketing and onboarding.

**5. Knowledge-Based Recommendation:**

* **Usefulness:**  
  This system uses explicit knowledge about user preferences and item features to make recommendations.
* **Example:**  
  A **travel website** may suggest a honeymoon package to a user who selects "romantic" and "beach" as preferences.
* **Real-Life Impact:**  
  It is useful when historical data is not available or when user needs are very specific.

**MU Q. Structure of the Web.**

Ans:

The **structure of the web** refers to how web pages are organized, linked, and interconnected across the internet. It plays an important role in web navigation, search engine indexing, and the flow of information.

**1. Web as a Directed Graph:**

* The structure of the web can be visualized as a **directed graph** where:
  + **Nodes** represent web pages.
  + **Edges** represent hyperlinks from one page to another.
* These links help search engines crawl pages and create rankings.

**2. Components of Web Structure:**

1. **Strongly Connected Component (SCC):**
   * A group of web pages where each page is reachable from every other page.
   * This is the **core** of the web, where intense linking happens.
2. **IN Component:**
   * Contains pages that can reach the SCC but cannot be reached from it.
   * These are often **new or entry** pages.
3. **OUT Component:**
   * Contains pages that are reachable from the SCC but do not link back to it.
   * Often contains **exit or final** pages.
4. **Tendrils:**
   * Pages that are connected to IN or OUT components but not directly to the SCC.
   * These are like **branches or isolated paths**.
5. **Disconnected Components:**
   * Pages or websites that are **completely isolated**, having no links to or from the main web graph.

**3. Real-Life Example:**

* A website like Wikipedia (core part of the SCC) links to many pages and is linked by others.
* A new blog with no backlinks is in the IN component.
* A government form download page may lie in the OUT component—it is linked from many sources but doesn't link out much.

**4. Importance of Web Structure:**

* Helps **search engines** like Google understand the importance of pages.
* Influences **PageRank** and **HITS** algorithms.
* Determines how quickly and efficiently content can be discovered and indexed.

**Q. Illustrate differet relational algebra operations using mapreduce. [May 2024]**

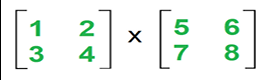
**Ans:**

**Relational algebra operations using MapReduce:**

1. **Selection (σ):**
   * Selection filters records based on a condition.
   * **Map step:** Each record is checked. If it satisfies the condition, it is emitted.
   * **Reduce step:** Identity function or may simply output the values.
   * **Example:** Select students with marks > 60
     + Map(key, value): if value.marks > 60 emit(key, value)
     + Reduce(key, values): emit(key, values)
2. **Projection (π):**
   * Projection retrieves specific columns from records.
   * **Map step:** Emit only the required fields.
   * **Reduce step:** Removes duplicates.
   * **Example:** Project student names
     + Map(key, value): emit(value.name, null)
     + Reduce(name, list): emit(name, null)
3. **Union (R ∪ S):**
   * Combines tuples from two relations without duplicates.
   * **Map step:** Emit all tuples from both datasets with a common key.
   * **Reduce step:** Eliminate duplicates.
   * **Example:**
     + Map(key, value): emit(value, null)
     + Reduce(value, list): emit(value, null)
4. **Intersection (R ∩ S):**
   * Returns common tuples from two relations.
   * **Map step:** Emit tuples from both with a label indicating origin.
   * **Reduce step:** Emit only tuples that occur in both datasets.
   * **Example:**
     + Map(key, value): emit(value, “R”) or emit(value, “S”)
     + Reduce(value, labels): if labels contains both “R” and “S” then emit(value)
5. **Difference (R − S):**
   * Returns tuples from R that are not in S.
   * **Map step:** Emit tuples with source identifier.
   * **Reduce step:** Keep tuples that occur only from R.
   * **Example:**
     + Map(key, value): emit(value, "R" or "S")
     + Reduce(value, list): if "R" in list and "S" not in list then emit(value)
6. **Cartesian Product (R × S):**
   * Pairs each tuple of R with every tuple of S.
   * **Map step:** Emit R tuples with tag “R” and S tuples with tag “S”.
   * **Reduce step:** Combine all R and S records in the same group.
   * **Example:**
     + Map(key, value): emit(“key”, (value, tag))
     + Reduce(key, list): for each r in R and s in S, emit(r, s)
7. **Join (⨝):**
   * Combines tuples from two relations based on a common attribute.
   * **Map step:** Emit join key as key and tuple with tag as value.
   * **Reduce step:** Match tuples with the same join key.
   * **Example:**
     + Map(key, value): emit(value.id, (value, “R” or “S”))
     + Reduce(id, list): emit all matching pairs between R and S

**Numericals:**

Q. Apply Map-reduce algorithm to multiply two matrices.



Q. Apply Map-reduce algorithm for selection relational algebra operation

Scenario:

Select all the rows where value of B is less than or equal to 4 for given dataset.

