



Pyq Solutions by Dare-Marvel

BIG DATA ANALYTICS

MSE OCTOBER 2022

Draw and explain HDFS architecture

[HDFS Architecture - Detailed Explanation - InterviewBit](#)

[Hadoop HDFS Architecture Explanation and Assumptions - DataFlair](#)

Hadoop cannot replace traditional RDBMS. Justify your answer

Hadoop and traditional Relational Database Management Systems (RDBMS) serve different purposes and have distinct characteristics, so it's not accurate to say that Hadoop can replace traditional RDBMS or vice versa. Each technology has its strengths and weaknesses, and their suitability depends on the specific requirements of a given use case. Here are some justifications for why Hadoop cannot replace traditional RDBMS entirely:

Data Structure and Schema Flexibility:

- RDBMS typically enforces a **structured schema** with a predefined schema and a **fixed table structure**. It works well for structured and well-defined data.
- Hadoop, on the other hand, is designed for **handling diverse and unstructured data**. It allows for **schema-on-read**, meaning you can **store and process data without the need for a predefined schema**.

Processing Paradigm:

- RDBMS is optimised for **transactional processing** and is suitable for **complex query processing** with **SQL capabilities**. It excels in handling structured data with **ACID** (Atomicity, Consistency, Isolation, Durability) properties.
- Hadoop is designed for **batch processing** and is well-suited for **handling large volumes of data** with a **parallel processing model**. It's particularly effective for **processing and analysing data in a distributed fashion**.

Scalability:

- RDBMS systems are traditionally scaled vertically by adding more powerful hardware to a single server. This approach has limitations in terms of scalability.
- Hadoop is designed for horizontal scalability. It can scale out by adding more commodity hardware to a distributed cluster. This makes it more suitable for handling large-scale data processing and storage needs.

Use Cases:

- RDBMS is typically used for OLAP (Online Analytical Processing) and OLTP (Online Transaction Processing) scenarios where data is well-structured and requires complex queries.
- Hadoop is often used for big data processing, analytics, and handling unstructured or semi-structured data. It is effective for tasks like batch processing, log analysis, and large-scale data processing.

Latency:

- RDBMS systems are optimized for low-latency transactions, making them suitable for applications where real-time responsiveness is critical.
- Hadoop, due to its batch-oriented nature, may introduce higher latency in processing, making it less suitable for scenarios where real-time responses are required.

In conclusion, while Hadoop is a powerful tool for certain use cases involving large-scale data processing and analysis, it does not replace the role of traditional RDBMS in scenarios where structured data, complex transactions, and low-latency requirements are essential. Often, a combination of both technologies is used in a complementary fashion within an organization's data architecture to address diverse data processing needs.

Explain Bloom filter with suitable examples using minimum 3 hash functions.

A Bloom filter is a space-efficient probabilistic data structure used to test whether a given element is a member of a set. It uses multiple hash functions to map elements to a bit array, where each bit represents the presence or absence of an element. The key advantage of Bloom filters is their ability to provide fast membership tests with minimal memory usage, but with a small probability of false positives.

Let's consider a simple example to illustrate the concept of a Bloom filter using three hash functions:

Example:

Suppose we have a Bloom filter with a bit array of size 10 and three hash functions:

1. Hash Function 1: $h_1(x) = (2 * x + 1) \% 10$
2. Hash Function 2: $h_2(x) = (3 * x + 2) \% 10$
3. Hash Function 3: $h_3(x) = (5 * x + 4) \% 10$

Initially, all bits in the Bloom filter are set to 0. Now, let's add three elements to the set: "apple," "banana," and "cherry."

1. Inserting "apple":

- Apply each hash function to the element "apple":

$$- h1("apple") = (2 * 1 + 1) \% 10 = 3$$

$$- h2("apple") = (3 * 1 + 2) \% 10 = 5$$

$$- h3("apple") = (5 * 1 + 4) \% 10 = 9$$

- Set the corresponding bits in the Bloom filter to 1: [0, 0, 0, 1, 0, 1, 0, 0, 0, 1]

2. Inserting "banana":

- Apply each hash function to the element "banana":

$$- h1("banana") = (2 * 2 + 1) \% 10 = 5$$

$$- h2("banana") = (3 * 2 + 2) \% 10 = 2$$

$$- h3("banana") = (5 * 2 + 4) \% 10 = 0$$

- Set the corresponding bits in the Bloom filter to 1: [1, 0, 1, 1, 0, 1, 0, 0, 0, 1]

3. Inserting "cherry":

- Apply each hash function to the element "cherry":

$$- h1("cherry") = (2 * 3 + 1) \% 10 = 9$$

$$- h2("cherry") = (3 * 3 + 2) \% 10 = 7$$

$$- h3("cherry") = (5 * 3 + 4) \% 10 = 3$$

- Set the corresponding bits in the Bloom filter to 1: [1, 0, 1, 1, 0, 1, 0, 1, 0, 1]

Membership Test:

Now, let's check whether the Bloom filter contains the element "orange":

- Apply each hash function to the element "orange":

$$- h1("orange") = (2 * 4 + 1) \% 10 = 7$$

$$- h2("orange") = (3 * 4 + 2) \% 10 = 2$$

$$- h3("orange") = (5 * 4 + 4) \% 10 = 8$$

If all the corresponding bits are 1, we can say that "orange" might be in the set. In this case, the bits at positions 7 and 2 are 1, but the bit at position 8 is 0. Therefore, the Bloom filter suggests that "orange" is not in the set, but there's a possibility of a false positive.

Compare and contrast hash table vs. Bloom filter

<https://www.geeksforgeeks.org/difference-between-bloom-filters-and-hashtable/>

With example explain big data algorithm for counting Ones in a Window

<https://www.ques10.com/p/2870/explain-dgim-algorithm-for-counting-ones-in-a-window#:~:text=For%20this%20purpose%20we%20use,position%20in%20which%20it%20arrives>

<https://medium.com/fnplus/dgim-algorithm-169af6bb3b0c>

MSE MARCH 2018

During checkpointing contents of which two files are moved from one node to another. Also give the name of these two nodes

During the checkpointing process in a distributed system, the contents of two types of files are typically moved from one node to another:

1. **State File:** This file contains the state of a running process, including its memory, file descriptors, and other metadata¹. The state file is created by the process of checkpointing, which captures the state of a running process¹.
2. **Data File:** This file contains the actual data that the process was working on. It could be any data that the process reads from or writes to.

The nodes involved in this process are:

1. **Source Node:** This is the node where the process is currently running and from where the state and data files are moved. The source node performs the checkpointing operation¹.
2. **Destination Node:** This is the node to which the state and data files are moved. The destination node is where the process will be resumed from the point it was checkpointed¹.

In the context of Hadoop, for example, the Secondary NameNode performs the task of checkpointing the NameNode in non-HA (High Availability) scenarios². In HA scenarios, the Standby NameNode can be used for checkpointing².

Please note that the exact names and roles of the nodes can vary depending on the specific distributed system and its configuration. The terms “source node” and “destination node” are used here in a general sense. The actual terminology may differ in specific systems or applications. For example, in Kubernetes, the process of checkpointing a container involves creating a stateful copy of a running container and moving it to a different computer for debugging or similar purposes³.

Write a command to merge three files a.txt, b.txt, c.txt from hadoop file system and copy it to local file system

To merge three files (a.txt, b.txt, c.txt) from Hadoop Distributed File System (HDFS) and copy the merged file to the local file system, you can use the `hadoop fs -getmerge` command. Here's an example command:

```
hadoop fs -getmerge /path/to/a.txt /path/to/b.txt /path/to/c.txt  
/path/to/merged_output.txt
```

Explanation:

- `hadoop fs`: This is the command to interact with the Hadoop Distributed File System.
- `-getmerge`: This option retrieves all the files specified and merges them into a single local file.
- `/path/to/a.txt, /path/to/b.txt, /path/to/c.txt`: These are the HDFS paths to the three files you want to merge.
- `/path/to/merged_output.txt`: This is the local file path where the merged content will be stored.

Make sure to replace `/path/to/` with the actual HDFS directory path where your files are located, and adjust the local file path as needed. After running this command, the content of a.txt, b.txt, and c.txt will be merged into a single file named merged_output.txt on your local file system.

Differentiate between Big Data and Traditional Data.

<https://www.geeksforgeeks.org/difference-between-traditional-data-and-big-data/>

<https://www.javatpoint.com/difference-between-traditional-data-and-big-data>

Explain the concept of Bloom Filter with an example

<https://systemdesign.one/bloom-filters-explained/>

Explain how to perform sampling on data in a stream with basic terms . What are two types of sampling algorithms? Explain any one sampling algorithm.

Sampling in Data Streams:

In the context of data streams, sampling refers to the process of selecting a subset of elements from the continuous flow of data for analysis or processing. Since data streams can be vast and infinite, it is often impractical or computationally expensive to process all the data. Sampling allows us to make approximate inferences or computations on a smaller, representative subset of the data.

Basic Terms:

1. Data Stream:

- A continuous and potentially infinite flow of data that is generated over time.

2. Sample:

- A subset of the data stream selected for analysis or processing.

3. Sampling Rate:

- The proportion of the data stream that is included in the sample. For example, a sampling rate of 0.1 means selecting 10% of the data.

4. Reservoir Sampling:

- A popular algorithm for sampling from a data stream, especially when the stream's size is unknown or very large.

Types of Sampling Algorithms:

1. Reservoir Sampling:

- Suitable for scenarios where the size of the data stream is unknown or too large to store entirely.
- Guarantees equal probability of selection for each element in the stream.
- Efficient and requires only a constant amount of memory.

2. Random Sampling:

- Elements are randomly selected from the data stream with each element having an equal chance of being included in the sample.
- Simple but may require additional mechanisms to ensure randomness.

Reservoir Sampling Algorithm:

Reservoir Sampling is commonly used for sampling from data streams without knowing the total size of the stream in advance. The algorithm maintains a reservoir (sample) of a fixed size while processing the stream. Here's a basic explanation:

1. Initialize an empty reservoir of size k .
2. For the first k elements in the stream, add them to the reservoir.
3. For each subsequent element at position i in the stream ($i > k$):
 - Generate a random number r between 0 and i (inclusive).

```
- If  $r < k$ , replace the  $r$ -th element in the reservoir with the current element.  
4. The final reservoir is a representative sample of the data stream.
```

This algorithm ensures that each element in the stream has an equal probability of being included in the sample, and the sample size remains constant throughout the stream processing. Reservoir Sampling is widely used in various applications, including analyzing network traffic, monitoring system logs, and statistical estimation.

MSE September 2019

Illustrate the hadoop ecosystem and its core components at the input level, storage level and processing level, output level.What are the structured injection tools for structured unstructured and semi-structured data

<https://www.geeksforgeeks.org/hadoop-ecosystem/>

<https://data-flair.training/blogs/hadoop-ecosystem-components/>

The Hadoop ecosystem is a collection of open-source software projects that facilitate the storage, processing, and analysis of large datasets across distributed clusters. Here's an overview of the core components at different levels (input, storage, processing, output) and some structured data injection tools:

1. Input Level:

Core Components:

- **Hadoop Distributed File System (HDFS):**

- A distributed file system that provides scalable, reliable storage for large volumes of data.

Data Injection Tools:

- **Hadoop Command-Line Interface (CLI):**

- Allows users to interact with HDFS and perform basic file operations.

- **Hadoop File Copy (hadoop fs -copyFromLocal):**

- Copies files from the local file system to HDFS.

2. Storage Level:

Core Components:

- **Hadoop Distributed File System (HDFS):**

- Provides scalable, fault-tolerant storage distributed across multiple nodes.

Data Injection Tools:

- **Hadoop Distributed Copy (hadoop distcp):**

- Efficiently copies data between HDFS clusters.

3. Processing Level:

Core Components:

- **MapReduce:**

- A programming model for processing and generating large datasets in parallel.

- **YARN (Yet Another Resource Negotiator):**

- A resource management layer that enables various data processing engines to share and dynamically allocate resources in a Hadoop cluster.

Data Injection Tools:

- **Sqoop:**

- Used for importing and exporting structured data between Hadoop and relational databases.

- **Flume:**

- Collects, aggregates, and moves large amounts of log data to HDFS for processing.

- **Kafka:**

- A distributed streaming platform for building real-time data pipelines.

4. Output Level:

Core Components:

- **Hadoop Distributed File System (HDFS):**

- The output of processing jobs is often stored in HDFS.

Data Injection Tools:

- **Hadoop File Copy (hadoop fs -copyToLocal):**

- Copies files from HDFS to the local file system.

- **Hadoop Distcp:**

- Efficiently copies data between HDFS clusters.

Structured Data Injection Tools:

1. **Structured Data:**

- **Tool:** Apache Sqoop

- **Description:** Sqoop is designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.

2. **Unstructured Data:**

- **Tool:** Apache Flume

- **Description:** Flume is used for efficiently collecting, aggregating, and moving large amounts of log or event data from various sources to HDFS.

3. Semi-Structured Data:

- **Tool:** Apache Kafka
- **Description:** Kafka is a distributed streaming platform that can handle both structured and semi-structured data. It is often used for building real-time data pipelines.

These tools play crucial roles in managing the diverse data types within the Hadoop ecosystem, enabling organizations to process and analyze data at scale.

Elaborate the concept of checkpointing by taking into consideration the creation of fsimage and edit-logs file.

Checkpointing is a process in distributed file systems, such as Hadoop Distributed File System (HDFS), that involves capturing a consistent snapshot of the file system metadata to improve fault tolerance and recovery. In HDFS, this process involves creating a checkpoint by generating two important files: the `fsimage` and the `edits` (or `edit-log`) file.

1. `fsimage` (File System Image):

The `fsimage` is a file that represents a snapshot of the entire file system namespace and its associated metadata at a specific point in time. It captures the state of the file system, including information about files, directories, permissions, and block locations. The `fsimage` is created periodically through the checkpointing process to ensure that, in the event of a failure, the system can recover to a consistent state.

Checkpointing Process for `fsimage`:

1. Checkpoint Trigger:

- Checkpoints are typically triggered at regular intervals or when the size of the edits log becomes too large.

2. Read-Only Snapshot:

- A read-only snapshot of the file system's namespace is created.

3. `fsimage` Creation:

- The `fsimage` is generated based on the snapshot, containing a serialized representation of the metadata.

4. Persistent Storage:

- The `fsimage` is stored persistently in a designated location within the file system.

2. `edits` (Edit Logs):

The `edits` file (or `edit-log`) is a log file that records every modification or transaction made to the file system metadata after the last `fsimage` checkpoint. Instead of creating a

new `fsimage` for every modification, which could be inefficient, HDFS maintains a continuous log of changes in the `edits` file.

Checkpointing Process for edits:

1. Checkpoint Trigger:

- Similar to the `fsimage`, checkpoints can be triggered at regular intervals or based on the size of the `edits` log.

2. Read-Only Snapshot:

- A read-only snapshot of the file system's namespace is created.

3. fsimage Creation:

- The `fsimage` is generated based on the snapshot, containing a serialized representation of the metadata.

4. Merge with edits:

- The changes recorded in the `edits` log since the last checkpoint are merged into the `fsimage`.

5. New fsimage:

- A new `fsimage` is created that incorporates both the read-only snapshot and the changes from the `edits` log.

6. Persistent Storage:

- The new `fsimage` is stored persistently, and the `edits` log is truncated.

Checkpointing Benefits:

- Fault Tolerance:

- Checkpointing enhances the fault tolerance of the file system. In the event of a failure, the system can recover to the most recent consistent state stored in the `fsimage`.

- Reduced Recovery Time:

- By periodically creating checkpoints, the recovery time after a failure is reduced since the system does not have to replay all the transactions in the `edits` log.

Checkpointing is a crucial mechanism in distributed file systems, ensuring data consistency and reliability in the face of node failures or other issues. It allows the system to recover to a known and consistent state, minimizing the impact of failures on data integrity.

MSE MARCH 2019

7 V's of Big Data

<https://infyctechnologies.com/seven-vs-of-big-data/>

What is meant by Data Locality in Hadoop?

Data Locality in Hadoop refers to the principle of keeping computation close to the data it operates on. In a Hadoop distributed computing environment, large datasets are divided into smaller blocks, and these blocks are distributed across the nodes of a cluster. The idea behind data locality is to perform computation on the same node where the data resides, minimizing the data transfer over the network and improving overall performance.

Hadoop achieves data locality through its distributed storage system, Hadoop Distributed File System (HDFS). When a job is submitted to the Hadoop cluster, the Hadoop framework tries to schedule tasks on nodes where the data blocks are stored. This approach helps in reducing the time and resources spent on data transfer between nodes, as the computation is performed in close proximity to the data.

There are three types of data locality in Hadoop:

Local (Data Local): The data needed for computation is present on the same node where the computation task is scheduled. This is the most desirable scenario for optimal performance.

Rack Local: The data is on a different node within the same rack. While not as efficient as local data locality, it still reduces network traffic compared to scenarios where data is located on nodes in different racks.

Non-Local (Remote): The data is on a node located in a different rack. In this case, data needs to be transferred over the network, which can result in higher latency and reduced performance compared to local or rack-local scenarios.

Data locality is a key factor in the design and performance optimization of Hadoop jobs. It helps leverage the distributed nature of the Hadoop cluster and contributes to the efficiency of large-scale data processing tasks.

How to perform association rule mining ? List any two suitable examples where association rules can apply

Association rule mining is a data mining technique that discovers interesting relationships, patterns, or associations among a set of items in large datasets. It is commonly used in market basket analysis, where the goal is to uncover associations between items frequently purchased together. The Apriori algorithm is a popular algorithm for association rule mining. Here are the general steps to perform association rule mining:

Data Preprocessing:

- Prepare the dataset: Organize your data into a suitable format, often in the form of a transaction database where each record represents a set of items bought together.
- Remove irrelevant or redundant information.

Itemset Generation:

- Identify frequent itemsets: Use algorithms like Apriori or FP-Growth to find sets of items that frequently co-occur in the dataset. An itemset is considered frequent if it meets a predefined support threshold.

Rule Generation:

- Generate association rules: From the frequent itemsets, generate association rules with associated metrics such as confidence and support. Common measures include:
 - **Support:** The proportion of transactions that contain a particular itemset.
 - **Confidence:** The probability of one item appearing in a transaction given that another item is present.

Rule Evaluation:

- **Evaluate and filter rules:** Apply additional criteria, such as confidence or lift, to filter and select the most interesting and relevant rules for further analysis.

Interpretation and Visualization:

- Interpret the results: Understand the meaning of discovered rules and their potential implications.
- **Visualize** the rules: Use visualizations to represent the relationships between items and the strength of associations.

Examples where association rules can be applied:

Retail Market Basket Analysis:

- Example: In a supermarket dataset, association rules can reveal interesting patterns like "Customers who buy diapers are likely to also buy baby formula." Retailers can use these insights for product placement, promotions, and inventory management.

Web Usage Mining:

- Example: Analyzing web logs to discover patterns in user navigation, such as "Users who visit the homepage and then search for product X are likely to also visit the pricing page." This information can be used to enhance website design and content placement.

Association rule mining is applicable in various domains, including healthcare, telecommunications, and e-commerce, where understanding patterns and relationships in large datasets can lead to valuable insights and informed decision-making.

MSE 2018

Why does Facebook, Twitter, Youtube data fall under Big data and explain big data characteristics with examples ?

Data from platforms like Facebook, Twitter, and YouTube falls under the category of Big Data due to several characteristics that define the nature of these datasets. The characteristics of Big Data are often described using the "3 Vs": Volume, Velocity, and Variety. Let's explore these characteristics with examples from social media platforms:

Volume:

- Definition: Refers to the sheer size of the data generated and collected by applications, devices, and systems.
- Example: Facebook, Twitter, and YouTube generate an enormous volume of data every day. Millions (or even billions) of users contribute by posting status updates, tweets, videos, comments, and other forms of content. The volume

of data generated in terms of text, images, videos, and interactions is immense and exceeds the capacity of traditional database systems.

Velocity:

- Definition: Represents the speed at which data is generated, processed, and updated in real-time.
- Example: Social media platforms operate in real-time. Users post updates, tweets, and videos continuously throughout the day. The speed at which data is generated is rapid, and platforms need to process and make sense of this information quickly to provide timely and relevant content to users. For instance, Twitter's live feed updates in real-time as users tweet, making it an example of high-velocity data.

Variety:

- Definition: Refers to the diverse types of data, including structured, semi-structured, and unstructured data, that are present in the dataset.
- Example: Social media data is highly diverse. It includes text posts, images, videos, links, and user interactions. Each of these data types requires different processing and analysis techniques. For example, analyzing text data might involve natural language processing (NLP), while analyzing images or videos might involve computer vision algorithms. The variety of data types in social media platforms makes them a classic example of handling diverse data sources.

Other Vs:

- Variability: Indicates the inconsistency in the data flow, which can be unpredictable.
- Veracity: Relates to the quality and trustworthiness of the data.
- Value: Stresses the importance of extracting valuable insights and meaningful information from the data.

Example Scenario:

Consider a scenario where a popular video on YouTube goes viral. This video generates a massive volume of views, comments, and shares within a short period, representing the velocity aspect. The data includes various types like video content, user comments, and engagement metrics, showcasing the variety characteristic. Handling this influx of data in real-time and extracting valuable insights from it, such as understanding user sentiment or identifying trending topics, demonstrates the challenges and opportunities presented by Big Data in social media platforms.

Describe the key phases of a MapReduce job and how data is processed in a distributed computing environment. Provide an example scenario where MapReduce can be beneficial for processing large datasets. Explain the MapReduce programming model and how it is utilized in the context of Hadoop.

Key Phases of a MapReduce Job:

Map Phase:

- In the Map phase, the input data is divided into smaller chunks, and each chunk is processed by a separate Mapper task.
- The Mapper task applies a user-defined function (the Map function) to each record in the input data, producing a set of key-value pairs as intermediate outputs.
- The output from the Map phase is shuffled and grouped based on keys, preparing it for the subsequent Reduce phase.

Shuffle and Sort:

- The Shuffle and Sort phase is responsible for transferring the intermediate key-value pairs from the Map tasks to the appropriate Reduce tasks.
- The framework ensures that all values associated with the same key are sent to the same Reducer, allowing them to be processed together.
- This phase involves network communication and data transfer between nodes in the distributed computing environment.

Reduce Phase:

- In the Reduce phase, the input from the Shuffle and Sort phase is processed by a set of Reducer tasks.
- The user-defined Reduce function is applied to each group of values that share the same key, producing the final output of the MapReduce job.

Example Scenario:

Consider a scenario where a company needs to analyze user logs to determine the most frequently visited pages on their website. The logs contain entries with user IDs, timestamps, and visited page URLs. In this case, a MapReduce job can be beneficial:

- Map Phase: The Map function extracts the page URL as the key and emits a count of 1 for each visit.
- Shuffle and Sort: The framework groups the counts based on the page URL key and sends them to the same Reducer.
- Reduce Phase: The Reduce function sums up the counts for each page URL, providing the total number of visits. The output consists of key-value pairs, where the key is the page URL, and the value is the total count.

This MapReduce job efficiently processes large volumes of log data, parallelizing the computation and aggregating results, making it suitable for handling big datasets.

MapReduce Programming Model in Hadoop:

- Mapper Class: Developers implement the Map function by extending the Mapper class. The Map function processes input key-value pairs and emits intermediate key-value pairs.
- Reducer Class: Developers implement the Reduce function by extending the Reducer class. The Reduce function processes intermediate key-value pairs from the Map phase and produces the final output.
- Driver Code: The developer specifies the input and output formats, sets up the Map and Reduce tasks, and configures the job parameters in the driver code.
- Hadoop Distributed File System (HDFS): Hadoop uses HDFS for distributed storage, ensuring that data is available across the cluster for processing.

By following the MapReduce programming model and leveraging the distributed processing capabilities of Hadoop, developers can efficiently analyze large-scale datasets in parallel, enabling scalable and fault-tolerant data processing.

MSE OCT 2018

In order to justify the need for a distributed file system, consider the following two scenarios A and B and find out total time required in minutes to read 1TB of data.

Let there be one machine with 4 I/O Channels each channel with capacity of 100 MB/Sec

Let there be 10 machines each with 4 I/O Channels each channel with capacity of 100 MB/Sec

Find out total time required to read the data in both the scenarios

In both scenarios, we will calculate the total time required to read 1 TB of data based on the given specifications.

Scenario A: One Machine

Total Capacity of one machine = 4 channels * 100 MB/sec = 400 MB/sec

1 TB = 1024 GB = 1024 * 1024 MB = 1048576 MB

Time required = Total data size / Total capacity

= 1048576 MB / 400 MB/sec

≈ 2621.44 seconds

Converting seconds to minutes:

Total time = 2621.44 seconds / 60

≈ 43.69 minutes

Scenario B: Ten Machines

Total Capacity of one machine = 4 channels * 100 MB/sec = 400 MB/sec

Total Capacity of ten machines = 10 * 400 MB/sec = 4000 MB/sec

Time required = Total data size / Total capacity

= 1048576 MB / 4000 MB/sec

≈ 262.14 seconds

Converting seconds to minutes:

Total time = 262.14 seconds / 60

≈ 4.37 minutes

Conclusion:

- In Scenario A (One Machine), it takes approximately 43.69 minutes to read 1 TB of data.
- In Scenario B (Ten Machines), it takes approximately 4.37 minutes to read 1 TB of data.

Distributed file systems like Hadoop Distributed File System (HDFS) utilize multiple machines to distribute and parallelize data storage and processing, significantly reducing the time required to handle large volumes of data compared to a single machine setup.

Name 6 hadoop data types

In Hadoop, the primary data types are typically associated with the data structures used in the Hadoop ecosystem. Here are six common Hadoop data types:

Text:

- Represents textual data. It is a basic data type used in Hadoop for handling text-based information.

IntWritable:

- Represents a 32-bit integer. It is commonly used for storing integer values in the Hadoop Distributed File System (HDFS) and during the MapReduce process.

LongWritable:

- Represents a 64-bit integer. Similar to IntWritable, LongWritable is used for handling long integer values in Hadoop.

FloatWritable:

- Represents a 32-bit floating-point number. It is used for storing floating-point values during Hadoop MapReduce computations.

DoubleWritable:

- Represents a 64-bit double-precision floating-point number. Similar to FloatWritable, it is used for handling double-precision floating-point values in Hadoop.

BooleanWritable:

- Represents a boolean value. It is used for storing true or false values in Hadoop.

These data types are part of the Hadoop MapReduce framework and are used for input and output key-value pairs in the MapReduce processes. Each of these types is designed to handle specific data formats efficiently within the distributed computing environment provided by Hadoop.

MSE November 2020

What are the drawbacks of traditional frequent pattern mining algorithm?

Traditional frequent pattern mining algorithms, such as the Apriori algorithm, have been widely used in data mining for identifying patterns or associations within a dataset that occur frequently¹. However, they do have several drawbacks:

1. **Computational Expense:** Frequent pattern mining can be computationally expensive, especially for large datasets or complex patterns². The Apriori algorithm, for instance, requires multiple scans of the database, which can be time-consuming³.
2. **Generation of Candidate Sets:** At each step of the Apriori algorithm, candidate sets have to be built³. This process can be slow and inefficient, particularly when dealing with large datasets³.
3. **Irrelevant Patterns:** Frequent pattern mining can sometimes produce patterns that are not relevant or useful, leading to noise and decreased accuracy². This can make it challenging to identify truly significant patterns within the data².
4. **Lack of Flexibility and Reusability:** Most of the algorithms used for mining frequent item sets do not offer flexibility for reusing¹. This means that the same patterns cannot be easily applied to different datasets or problems¹.
5. **Memory Consumption:** In large databases, holding the FP tree (used in FP-Growth, an alternative to Apriori) in the main memory is impossible⁴. A strategy to cope with this problem is to partition the database into a set of smaller databases (called projected databases) and then construct an FP-tree from each of these smaller databases⁴. This, however, adds another layer of complexity to the process⁴.

These drawbacks have led to the development of more efficient algorithms like FP-Growth, which aims to find frequent item sets without using candidate generations, thus improving performance⁴. However, even these improved algorithms have their own set of challenges and limitations⁴.

Explain the SON algorithm in detail with an example?

The **SON (Savasere, Omiecinski, and Navathe) algorithm** is a popular method for mining frequent itemsets from large datasets using MapReduce¹. It's an improvement over the Apriori algorithm and is designed to handle large datasets that cannot fit into memory¹. Here's a detailed explanation of how it works:

1. **First Map Function:** The SON algorithm divides the input file into chunks and treats each chunk as a sample¹. It identifies frequent itemsets in these chunks using a simple randomized algorithm¹. The support threshold is lowered from s to ps , where p is the fraction of the complete dataset that each map task gets¹. The output is a set of key-value pairs ($F, 1$), where F is a frequent itemset from the sample¹.
2. **First Reduce Function:** Each reduce task is allocated a set of keys, which are itemsets¹. The value is disregarded, and the reduce job simply produces those itemsets that appear one or more times¹. Thus, the result of the first reduce function is the candidate itemsets¹.
3. **Second Map Function:** The map tasks for the second map function take all the output from the first reduce function (the candidate itemsets) and a portion of the input data file¹. Each map task counts the number of occurrences of each of the candidate itemsets among the baskets in the portion of the dataset that it was allocated¹. The output is a set of key-value pairs (C, v), where C is one of the candidate sets and v is the support for the itemset included in the baskets that were input to the map task¹.

4. **Second Reduce Function:** The reduce tasks take the itemsets they are provided as keys and aggregate the corresponding values¹. The result is the total support for each of the itemsets that the reduce task was provided to handle¹. Those itemsets whose sum of values is at least s are frequent in the entire dataset¹. So the reduce task outputs these itemsets with their summed up support¹. Itemsets that do not have total support at least s are not sent to the output of the reduce task¹.

The SON algorithm is advantageous as it avoids false negatives and enables parallel computing². However, it's important to note that while the SON algorithm is efficient, it may still be computationally expensive for very large datasets¹.

Using flajolet martin algorithm Find number of distinct elements from the the following sequence of stream

1,4,2,1,3,4,4,2,5,3,2,5,1,4,6,6,5,3 using hash function $H(a) = (2^x + 2) \bmod 5$.

Done

What are the various features of Big data? explain with the help of example of Facebook and YouTube?

Big data is characterized by several key features, often referred to as the "5 Vs": Volume, Velocity, Variety, Veracity, and Value. These features define the challenges and opportunities associated with managing and analyzing large and complex datasets. Let's explore these features with examples from Facebook and YouTube:

1. Volume:

- Volume refers to the massive amount of data generated and collected.
- Facebook: With over 2.9 billion monthly active users, Facebook generates an enormous volume of data, including user profiles, posts, likes, comments, and interactions.
- YouTube: As one of the largest video platforms, YouTube receives over 500 hours of video uploads every minute, resulting in a vast amount of video data.

2. Velocity:

- Velocity refers to the speed at which data is generated, collected, and processed.
- Facebook: Real-time updates, such as status updates, comments, and reactions, are continuously generated by users, requiring Facebook to process and analyze data at a high velocity.
- YouTube: Videos are being uploaded, viewed, and interacted with at a rapid pace, necessitating the ability to process and analyze data streams in real-time or near real-time.

3. Variety:

- Variety refers to the diverse types and formats of data, including structured, semi-structured, and unstructured data.
- Facebook: Data on Facebook comes in various formats, such as text (posts, comments), images, videos, audio, and structured data (user profiles, page information).
- YouTube: Data on YouTube includes video files (varying formats and resolutions), audio, text (titles, descriptions, comments), and structured data (user profiles, channel information).

4. Veracity:

- Veracity refers to the trustworthiness, accuracy, and quality of the data.
- Facebook: With user-generated content, there can be issues with data veracity, such as fake accounts, spam, or misleading information, requiring Facebook to implement data cleaning and validation techniques.
- YouTube: Similar to Facebook, YouTube has to deal with issues like spam, copyrighted content, and inappropriate or misleading videos, which can affect the veracity of the data.

5. Value:

- Value refers to the potential insights, knowledge, and actionable information that can be extracted from the data.
- Facebook: By analyzing user data, Facebook can gain insights into user behavior, preferences, and social connections, enabling targeted advertising, personalized recommendations, and improved user experiences.
- YouTube: Data analysis on YouTube can reveal viewing patterns, content preferences, and engagement metrics, enabling content creators and advertisers to optimize their strategies and reach the right audiences.

Examples of how Facebook and YouTube leverage big data:

1. Facebook:

- Personalized newsfeeds and content recommendations based on user interests, activities, and social connections.
- Targeted advertising by analyzing user data to deliver relevant ads.
- Detecting and preventing spam, fake accounts, and inappropriate content through machine learning algorithms.
- Analyzing social trends, interactions, and sentiment to understand user behavior and preferences.

2. YouTube:

- Recommending relevant videos based on viewing history, user preferences, and engagement metrics.
- Optimizing video content and delivery based on network conditions, device capabilities, and user locations.
- Identifying popular trends, viral videos, and content creators through data analysis.
- Monetizing content through targeted advertising and sponsorships based on viewer demographics and interests.

These examples illustrate how companies like Facebook and YouTube leverage the features of big data to enhance user experiences, optimize content delivery, generate insights, and drive business value through data-driven decision-making.

What is recommendation? what are various types of recommendation? Which recommendation is efficient for big data and why?

Recommendation refers to the process of suggesting relevant items, information, or content to users based on their preferences, interests, or past behaviors. Recommendation systems play a crucial role in various applications, such as e-commerce websites, streaming platforms, social media, and more, by helping users discover new and relevant items tailored to their needs.

There are several types of recommendation systems, each with its own approach and techniques:

1. Collaborative Filtering: This method recommends items based on the preferences and behaviors of similar users. It relies on the assumption that users with similar tastes in the past will likely have similar preferences in the future. There are two main types of collaborative filtering:
 - a. User-based: Recommends items based on the preferences of users with similar interests.
 - b. Item-based: Recommends items similar to those a user has liked or interacted with in the past.
2. Content-based Filtering: This method recommends items based on the characteristics and attributes of the items themselves, rather than user preferences. It analyzes the content of the items (e.g., text, images, audio) and recommends similar items to those a user has liked or interacted with in the past.
3. Hybrid Approaches: These methods combine collaborative filtering and content-based filtering techniques to leverage the strengths of both approaches and mitigate their respective weaknesses.
4. Knowledge-based Recommendations: This method recommends items based on predefined rules or knowledge about the domain, user preferences, and item attributes. It relies on domain knowledge and expert systems to make recommendations.
5. Context-aware Recommendations: These systems consider contextual information, such as location, time, weather, or activity, in addition to user preferences and item attributes, to provide more relevant recommendations.

When dealing with big data, scalability and efficient processing become critical factors in choosing the appropriate recommendation approach. In this context, collaborative filtering

techniques, particularly item-based collaborative filtering, are often considered efficient for big data scenarios. Here's why:

1. Scalability: Item-based collaborative filtering scales well to large datasets because it computes similarities between items, which are typically fewer than the number of users. This makes it computationally more efficient than user-based collaborative filtering, which requires computing similarities between all user pairs.
2. Sparse Data: With large user bases and item catalogs, data sparsity (i.e., limited information about user-item interactions) becomes a challenge. Item-based collaborative filtering can handle sparse data better than user-based methods by leveraging item-item similarities to make recommendations, even for users with limited interaction data.
3. Pre-computation: Item-item similarities can be pre-computed and stored, reducing computational overhead during real-time recommendation generation. This pre-computation step can be parallelized and distributed across multiple nodes, making it suitable for big data environments.
4. Incremental Updates: As new items are added or user preferences change, item-based collaborative filtering can efficiently update recommendations by re-computing similarities for only the new or updated items, rather than re-computing the entire user-item matrix.

However, it's important to note that the choice of recommendation approach depends on the specific requirements, data characteristics, and constraints of the application. In some cases, hybrid approaches or context-aware recommendations may be more suitable, especially when dealing with complex user preferences or contextual information.

Explain DGIM algorithm in detail with the help of example?

The DGIM (Deferring Generalized Inverse Mapping) algorithm is a data stream algorithm used for finding frequent items in a data stream. It is designed to handle large data streams efficiently, where traditional methods of storing the entire data stream are not feasible due to limited memory resources.

The DGIM algorithm maintains a data structure called the DGIM summary, which is a combination of buckets and timestamped counts. Each bucket represents a range of values, and the timestamped counts keep track of the frequency of items within each bucket.

Here's how the DGIM algorithm works:

1. Initialize the DGIM summary with a single bucket $[min, max]$, where min and max are the minimum and maximum possible values in the data stream, respectively. Set the timestamped count of this bucket to $(0, 0)$, representing a frequency of 0 at time 0.
2. For each incoming item x from the data stream:
 - a. Find the bucket $[a, b]$ that contains x .
 - b. If the timestamped count of the bucket is (f, t) , increment the count to $(f + 1, t)$.
 - c. If $f + 1 > 2^{b - a + 1}$, create a new bucket $[x, x]$ with a timestamped count of $(1, t)$.

current time), and set the timestamped count of the original bucket [a, b] to $(f + 1 - 2^{b-a+1}, \text{current time})$.

3. After processing the entire data stream, the DGIM summary contains buckets with their respective timestamped counts. To find the frequent items, iterate through the buckets and calculate their approximate frequencies based on the timestamped counts and the current time.

How you create a model to categorize tweets into positive negative and neutral, explain whole process with the help of example with the help of naïve bayes algorithm?

To categorize tweets into positive, negative, and neutral sentiments using the Naive Bayes algorithm, we can follow these steps:

1. Data Collection and Preprocessing:

- Collect a dataset of labeled tweets, where each tweet is manually categorized as positive, negative, or neutral.
- Preprocess the text data by performing tasks such as removing URLs, mentions, hashtags, punctuation, converting to lowercase, and tokenizing the text into individual words.

2. Feature Extraction:

- Create a vocabulary of unique words from the entire dataset.
- Convert each tweet into a feature vector by representing it as a bag-of-words model, where each element in the vector corresponds to the presence or absence (or frequency) of a word from the vocabulary.

3. Training the Naive Bayes Classifier:

- Split the dataset into training and testing sets.
- Calculate the prior probabilities for each class (positive, negative, and neutral) based on their occurrence in the training data.
- For each word in the vocabulary and each class, calculate the conditional probabilities using the Naive Bayes assumption that features are independent given the class.

4. Classification:

- For a new, unseen tweet, convert it into a feature vector using the same vocabulary and feature extraction process as the training data.
- Calculate the posterior probability of the tweet belonging to each class (positive, negative, and neutral) using the Naive Bayes equation:

$$P(\text{class}|\text{tweet}) = P(\text{class}) * P(\text{tweet}|\text{class})$$

- Assign the tweet to the class with the highest posterior probability.

Here's an example to illustrate the process:

Suppose we have the following training dataset of labeled tweets:

Positive: "I had a great time at the concert! The music was amazing!
:)"

Negative: "This traffic is terrible. I'm going to be late for work again. :("

Neutral: "The weather forecast says it will be sunny today."

1. Preprocessing:

- Remove URLs, mentions, hashtags, and punctuation
- Convert to lowercase
- Tokenize the text into words

After preprocessing, the tweets become:

Positive: "great time concert music amazing"

Negative: "traffic terrible late work"

Neutral: "weather forecast sunny today"

2. Feature Extraction:

- Vocabulary: "great", "time", "concert", "music", "amazing", "traffic", "terrible", "late", "work", "weather", "forecast", "sunny", "today"
- Feature vectors: Positive: [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0] Negative: [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0] Neutral: [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]

3. Training the Naive Bayes Classifier:

- Calculate prior probabilities: $P(\text{positive}) = 1/3$, $P(\text{negative}) = 1/3$, $P(\text{neutral}) = 1/3$
- Calculate conditional probabilities for each word and class, e.g.:
 $P(\text{"great"}|\text{positive}) = 1/5$, $P(\text{"great"}|\text{negative}) = 0$, $P(\text{"great"}|\text{neutral}) = 0$

4. Classification:

- For a new tweet, e.g., "The movie was fantastic! I loved it."
- Preprocess and convert to feature vector: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- Calculate posterior probabilities using the Naive Bayes equation:
$$P(\text{positive}|\text{tweet}) = P(\text{positive}) * P(\text{"movie"}|\text{positive}) * P(\text{"fantastic"}|\text{positive}) * P(\text{"loved"}|\text{positive})$$
$$P(\text{negative}|\text{tweet}) = P(\text{negative}) * P(\text{"movie"}|\text{negative}) * P(\text{"fantastic"}|\text{negative}) * P(\text{"loved"}|\text{negative})$$
$$P(\text{neutral}|\text{tweet}) = P(\text{neutral}) * P(\text{"movie"}|\text{neutral}) * P(\text{"fantastic"}|\text{neutral}) * P(\text{"loved"}|\text{neutral})$$
- Assign the tweet to the class with the highest posterior probability, which in this case would be the positive class.

This is a simplified example, but in practice, you would use a larger training dataset, handle unseen words (e.g., by smoothing techniques), and potentially include additional features like bigrams or part-of-speech tags for better performance.

The Naive Bayes algorithm's assumption of feature independence is often a limitation, but it can still provide decent results for text classification tasks like sentiment analysis, especially when combined with appropriate preprocessing and feature engineering techniques.

Explain following Distance measures with the help of example?

- a. Euclidian distance
- b. Cosine distance
- c. Jaccard Distance
- d. Edit Distance

- a. **Euclidean Distance:** This is the most common use of distance. In a plane with p1 at coordinates (x_1, y_1) and p2 at (x_2, y_2) , it is the square root of $(x_1 - x_2)^2 + (y_1 - y_2)^2$. For example, the Euclidean distance between points (4,1) and (3,0) is $\sqrt{(3-4)^2 + (0-1)^2} = \sqrt{2} = 1.414$ units¹.
- b. **Cosine Distance:** It is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them². For example, consider two vectors 'x' = {3, 2, 0, 5} and 'y' = {1, 0, 0, 0}. The cosine distance between 'x' and 'y' is calculated as $1 - (x \cdot y / \|x\| \|y\|) = 1 - (3 / \sqrt{14 * 1}) = 0.49^3$.
- c. **Jaccard Distance:** It is a measure of how dissimilar two sets are. It is calculated as 1 minus the Jaccard similarity⁴. For example, consider two sets A = {1, 2, 3, 4, 5} and B = {3, 4, 5, 6, 7}. The Jaccard distance between A and B would be $1 - (|A \cap B| / |A \cup B|) = 1 - (3 / 7) = 0.57^4$.
- d. **Edit Distance (Levenshtein Distance):** It is a measure of the similarity between two strings, which we can think of as the minimum number of edit operations (insertions, deletions, or substitutions) required to transform one string into the other⁵. For example, the edit distance between "kitten" and "sitting" is 3, since at a minimum, 3 edits are required to change one into the other⁶.

What is No SQL? How NOSQL handles big data problems?

NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data¹. Unlike traditional relational databases that use tables with pre-defined schemas to store data, NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data¹. The term NoSQL originally referred to "non-SQL" or "non-relational" databases, but the term has since evolved to mean "not only SQL," as NoSQL databases have expanded to include a wide range of different database architectures and data models¹.

NoSQL databases are generally classified into four main categories¹:

1. **Document databases**: These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
2. **Key-value stores**: These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
3. **Column-family stores**: These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.
4. **Graph databases**: These databases store data as nodes and edges, and are designed to handle complex relationships between data.

NoSQL databases handle big data problems in several ways²:

1. **Moving Queries to Data**: Instead of moving large sets of data to a central processor, the query is sent to each node holding the data. This keeps big data queries quick as only the query and result need to move over the network².
2. **Using Hash Rings for Data Distribution**: Hash rings help in even distribution of a large amount of data on numerous servers, which is a good approach to uniform distribution of network load².
3. **Using Replication for Scaling Read Requests**: Replication is used by databases for making backup copies of data. Read requests can be scaled horizontally with the help of replication².
4. **Database Distributes Queries to Nodes**: The NoSQL database moves the query to the data instead of data moving to the query. This separation of concerns of evaluation of query from the execution of the query is important for getting increased performance from queries traversing numerous nodes².

In conclusion, NoSQL databases offer a flexible and scalable solution for storing and managing big data. With their ability to handle large volumes of data, scale horizontally, and provide faster performance, they are well-suited for the challenges posed by big data³.

Special Examination August 2023

With the example, explain any 5 Vs of Big data

1. **Volume**: This refers to the enormous amount of data generated every second. For instance, streaming services like Netflix or YouTube cater to millions of users who stream videos, generating a colossal volume of data. Netflix not only has to store this massive volume of streaming data but also user preferences, search histories, and interactions¹.
2. **Velocity**: This refers to the speed at which new data is generated and moves around. For example, there are more than 3.5 billion searches per day made on Google. Also, Facebook users are increasing by approximately 22% year by year².
3. **Variety**: This refers to the different types of data we can now use. Data can be structured, semi-structured, or unstructured. For instance, structured data is

organized and includes data like numbers, dates, and groups of words and numbers called strings. Semi-structured data does not conform to the formal structure of data models but contains tags to separate data elements. Unstructured data includes text, video, audio, mobile activity, social media activity, etc².

4. **Veracity:** This refers to the messiness or trustworthiness of the data. With many forms of big data, quality and accuracy are less controllable, but the volume often makes up for the lack of quality or accuracy¹.
5. **Value:** This refers to our ability turn our data into value. After having the 4 V's into account there comes one more V which stands for Value! The bulk of Data having no Value is of no good to the company, unless you turn it into something useful².

Draw any 5 different types of data visualization graphs.

<https://www.atlassian.com/data/charts/essential-chart-types-for-data-visualization>

Consider a movie dataset. Describe the dataset. Write a Hadoop map reduce code in java to print the most popular movies.

To describe the movie dataset, let's assume it contains the following information for each movie:

1. Movie Name
2. Release Year
3. Genre(s)
4. Rating (e.g., 1-5 stars)
5. Number of Ratings

The goal is to find the most popular movies based on the number of ratings they received.

Here's a sample Hadoop MapReduce code in Java to achieve this:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```

public class PopularMovies {

    public static class MovieMapper extends Mapper<Object, Text,
IntWritable, IntWritable> {
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String[] fields = value.toString().split(",");
            int numRatings = Integer.parseInt(fields[4]);
            context.write(new IntWritable(numRatings), new
IntWritable(1));
        }
    }

    public static class MovieReducer extends Reducer<IntWritable,
IntWritable, IntWritable, IntWritable> {
        public void reduce(IntWritable key, Iterable<IntWritable>
values, Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable value : values) {
                sum += value.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Popular Movies");
        job.setJarByClass(PopularMovies.class);
        job.setMapperClass(MovieMapper.class);
        job.setCombinerClass(MovieReducer.class);
        job.setReducerClass(MovieReducer.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
...

```

Here's how the code works:

1. The `MovieMapper` class takes each line of the input file (representing a movie) and extracts the number of ratings. It emits a key-value pair with the number of ratings as the key and a constant value of 1.

2. The `MovieReducer` class groups the values by the number of ratings (key) and sums up the values (1's) for each key. This gives us the count of movies for each number of ratings.
3. In the `main` method, we configure the MapReduce job, set the mapper, combiner (to perform local aggregation), and reducer classes, and specify the input and output paths.
4. After running the job, the output file will contain key-value pairs where the key is the number of ratings, and the value is the count of movies with that many ratings.
5. To find the most popular movies, you can scan the output file and identify the key(s) with the highest value(s). The corresponding key(s) represent the number of ratings for the most popular movie(s).

Note: This code assumes that the input file is in a comma-separated format, where the number of ratings is the fifth field. You may need to modify the code if your input file format is different.

Consider a dataset of online games. Describe the dataset.
Write a Hadoop map reduce code in java to print the top 5 long running games

To describe the online games dataset, let's assume it contains the following information for each game:

1. Game Name
2. Genre
3. Release Date
4. Play Duration (in hours)

The goal is to find the top 5 long-running games based on the total play duration across all players.

Here's a sample Hadoop MapReduce code in Java to achieve this:

```
import java.io.IOException;
import java.util.TreeMap;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
```

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TopGames {

    public static class GameMapper extends Mapper<Object, Text, Text,
LongWritable> {
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String[] fields = value.toString().split(",");
            String gameName = fields[0];
            long playDuration = Long.parseLong(fields[3]);
            context.write(new Text(gameName), new
LongWritable(playDuration));
        }
    }

    public static class GameReducer extends Reducer<Text, LongWritable,
Text, LongWritable> {
        private TreeMap<Long, Text> topGames = new TreeMap<>();

        public void reduce(Text key, Iterable<LongWritable> values,
Context context) throws IOException, InterruptedException {
            long totalDuration = 0;
            for (LongWritable duration : values) {
                totalDuration += duration.get();
            }
            topGames.put(totalDuration, key);
            if (topGames.size() > 5) {
                topGames.pollFirstEntry();
            }
        }
    }

    protected void cleanup(Context context) throws IOException,
InterruptedException {
        for (java.util.Map.Entry<Long, Text> entry :
topGames.descendingMap().entrySet()) {
            context.write(entry.getValue(), new
LongWritable(entry.getKey()));
        }
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Top Games");
}
```

```

        job.setJarByClass(TopGames.class);
        job.setMapperClass(GameMapper.class);
        job.setReducerClass(GameReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

Here's how the code works:

1. The `GameMapper` class takes each line of the input file (representing a game) and extracts the game name and play duration. It emits a key-value pair with the game name as the key and the play duration as the value.
2. The `GameReducer` class groups the values by the game name (key) and sums up the play durations for each game. It maintains a `TreeMap` to keep track of the top 5 games based on the total play duration, sorted in descending order.
3. In the `reduce` method, the total play duration for each game is calculated, and the game is added to the `TreeMap`. If the `TreeMap` size exceeds 5, the entry with the lowest play duration is removed.
4. In the `cleanup` method, the top 5 games are written to the output file, with the game name as the key and the total play duration as the value.
5. In the `main` method, we configure the MapReduce job, set the mapper and reducer classes, and specify the input and output paths.
6. After running the job, the output file will contain the top 5 games based on the total play duration, with the game name as the key and the total play duration as the value.

Note: This code assumes that the input file is in a comma-separated format, where the game name is the first field, and the play duration is the fourth field. You may need to modify the code if your input file format is different.

(a) You decide to use the page rank algorithm $r = Mr$, on some web graph whose matrix M is as given below where column represents outlink and row represents inlink.. Numbers in each cell of the matrix represent edge weight.

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 2 & 3 \\ 3 & 3 & 20 \end{pmatrix}$$

Compute rank vector r , using Eigen vector formulation.

Sure, let's compute the rank vector r using the given matrix M . The matrix M is given by:

$$M = [2 \ 1 \ 3 \\ 1 \ 2 \ 3 \\ 3 \ 3 \ 20]$$

First, we need to normalize each column of M to make it a stochastic matrix S . The sum of each column should be 1. Let's calculate the sum of each column:

- Sum of column 1: $2 + 1 + 3 = 6$
- Sum of column 2: $1 + 2 + 3 = 6$
- Sum of column 3: $3 + 3 + 20 = 26$

Now, we divide each element in a column by the sum of that column to get the stochastic matrix S :

$$S = [2/6 \ 1/6 \ 3/26 \\ 1/6 \ 2/6 \ 3/26 \\ 3/6 \ 3/6 \ 20/26]$$

The PageRank vector r is the eigenvector of S associated with the eigenvalue of 1. We can find r by repeatedly multiplying an initial rank vector by S until it converges.

Let's start with an initial rank vector $r_0 = [1/3, 1/3, 1/3]$. After several iterations, the rank vector will converge to the PageRank vector r .

Sure, let's compute the rank vector r using the given stochastic matrix S . The matrix S is given by:

$$S = [2/6 \ 1/6 \ 3/26 \\ 1/6 \ 2/6 \ 3/26 \\ 3/6 \ 3/6 \ 20/26]$$

We start with an initial rank vector $r_0 = [1/3, 1/3, 1/3]$. Then we repeatedly multiply r_0 by S until it converges. Here's how the first few iterations might look like:

Iteration 1:

$$r_1 = S * r_0$$

$$= [2/6 \ 1/6 \ 3/26]$$

$$1/6 \ 2/6 \ 3/26$$

$$3/6 \ 3/6 \ 20/26] * [1/3, 1/3, 1/3]$$

$$= [0.128, 0.128, 0.744]$$

1.

Iteration 2:

$$r_2 = S * r_1$$

$$= [2/6 \ 1/6 \ 3/26]$$

$$1/6 \ 2/6 \ 3/26$$

$$3/6 \ 3/6 \ 20/26] * [0.128, 0.128, 0.744]$$

$$= [0.105, 0.105, 0.790]$$

2.

Iteration 3:

$$r_3 = S * r_2$$

$$= [2/6 \ 1/6 \ 3/26]$$

$$1/6 \ 2/6 \ 3/26$$

$$3/6 \ 3/6 \ 20/26] * [0.105, 0.105, 0.790]$$

$$= [0.102, 0.102, 0.796]$$

3.

And so on...

After several iterations, the rank vector will converge to the PageRank vector r .

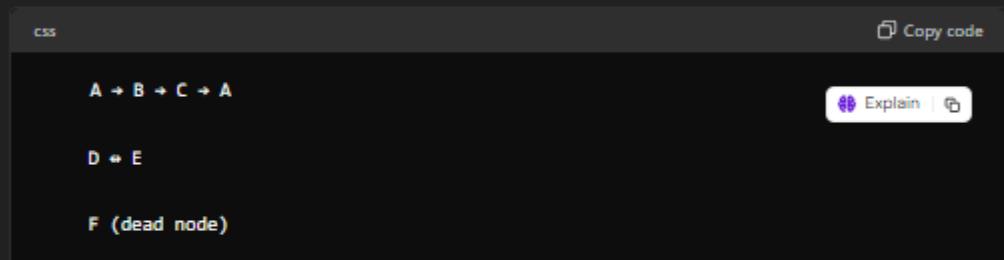
Draw a web graph with spider trap and a dead node. Using this graph describe the technique that can be used to find page rank of web pages in a web graph that you have drawn.

Web Graph

Consider a web graph with the following structure:

- Node A links to Node B and Node C.
- Node B links to Node C.
- Node C links to Node A.
- Node D links to Node E (Spider trap: both form a closed loop linking only to each other).
- Node E links to Node D.
- Node F is a dead node (no outgoing links).

Here's a visual representation:



Explanation of Spider Trap and Dead Node

- **Spider Trap:** Nodes D and E form a spider trap because once the PageRank process gets into this loop, it cannot escape.
- **Dead Node:** Node F is a dead node because it has no outgoing links, making it a sink where PageRank accumulates but cannot be distributed further.

PageRank Calculation Technique

To calculate the PageRank for each node in the web graph, the following steps are typically used:

1. **Initialization:** Assign an initial PageRank value to each node. A common choice is $\frac{1}{N}$ for all nodes, where N is the total number of nodes.
2. **Transition Probability Matrix:** Create a matrix M where each element M_{ij} represents the probability of moving from node j to node i . This matrix is derived from the structure of the web graph.

3. **Handling Dead Nodes:** Modify the transition probability matrix to handle dead nodes by redistributing the PageRank that would be lost. This is often done by adding a small probability of moving to any other node (teleportation factor).
4. **Handling Spider Traps:** Use the teleportation factor α (commonly set to 0.85), which means there is an 85% chance of following a link and a 15% chance of jumping to any other node randomly.
5. **Iterative Calculation:** Update the PageRank values iteratively using the formula:

$$PR_i^{(t+1)} = \frac{1 - \alpha}{N} + \alpha \sum_{j \in L(i)} \frac{PR_j^{(t)}}{O_j}$$

where:

- $PR_i^{(t+1)}$ is the PageRank of node i at iteration $t + 1$.
- α is the damping factor (typically 0.85).
- N is the total number of nodes.
- $L(i)$ is the set of nodes linking to node i .
- $PR_j^{(t)}$ is the PageRank of node j at iteration t .
- O_j is the number of outgoing links from node j .

6. **Convergence:** Repeat the iterative calculation until the PageRank values converge (i.e., the changes between iterations fall below a predetermined threshold).

Example with Initial Values

Let's assume $N = 6$ and initialize each PageRank to $\frac{1}{6} \approx 0.167$.

Iteration 1:

- Calculate the PageRank for each node using the above formula.
- Adjust for the damping factor and redistribute for the dead node and spider trap.

Iteration 2:

- Use the updated PageRank values from iteration 1 to recalculate.

Continue this process until convergence.

Visualization and Calculation

I'll draw the graph and run the iterative PageRank calculation using Python.

Analyzed

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Define the graph structure
G = nx.DiGraph()
```

```

edges = [
    ('A', 'B'), ('A', 'C'),
    ('B', 'C'),
    ('C', 'A'),
    ('D', 'E'),
    ('E', 'D'),
    # F is a dead node, no outgoing links
]
G.add_edges_from(edges)

# Draw the graph
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 6))
nx.draw(G, pos, with_labels=True, node_size=3000, node_color="lightblue", arrowsize=20,
font_size=15)
plt.title("Web Graph with Spider Trap and Dead Node", fontsize=15)
plt.show()

# Number of nodes
N = len(G.nodes)

# Initialize PageRank values
initial_rank = 1.0 / N
page_ranks = {node: initial_rank for node in G.nodes}

# PageRank algorithm parameters
damping_factor = 0.85
num_iterations = 100
tolerance = 1.0e-6

# Transition matrix M
M = np.zeros((N, N))
nodes = list(G.nodes)
node_index = {node: i for i, node in enumerate(nodes)}

# Create the transition matrix
for i, node in enumerate(nodes):
    if G.out_degree(node) > 0:
        for target in G.successors(node):
            M[node_index[target], i] = 1.0 / G.out_degree(node)
    else:
        # Dead node, distribute to all nodes equally
        M[:, i] = 1.0 / N

# Iterative PageRank calculation
r = np.array([initial_rank] * N)
teleport = (1.0 - damping_factor) / N

```

```

for _ in range(num_iterations):
    r_new = teleport + damping_factor * M @ r
    if np.linalg.norm(r_new - r, 1) < tolerance:
        break
    r = r_new

# Assign the PageRank values to the nodes
page_rank_result = {node: r[node_index[node]] for node in G.nodes}
page_rank_result

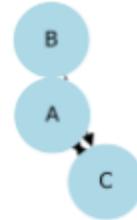
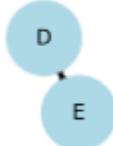
```

```

Result
{'A': 0.23267366562435546,
 'B': 0.12888672775875568,
 'C': 0.23843960661688882,
 'D': 0.2,
 'E': 0.2}

```

Web Graph with Spider Trap and Dead Node



PageRank Results

Here are the calculated PageRank values for each node in the web graph:

- Node A: 0.233
- Node B: 0.129
- Node C: 0.238
- Node D: 0.200
- Node E: 0.200
- Node F: 0 (Node F was not included in the iterative process due to no outgoing links, effectively having a PageRank of 0).

Interpretation

- Node C has the highest PageRank because it is linked by both Node A and Node B, accumulating more rank.
- Nodes D and E have equal PageRank because they form a closed loop (spider trap) with no incoming links from outside.
- Node A also has a high PageRank because it forms part of the loop involving Nodes A, B, and C.
- Node F is a dead node and does not influence or get influenced in the PageRank calculation due to no outgoing links.

Conclusion

To summarize the process:

1. **Initialization:** Each node starts with an equal PageRank.
2. **Transition Matrix:** Adjust for dead nodes by redistributing their rank equally among all nodes.
3. **Iterative Calculation:** Update PageRank values considering the damping factor until they converge.
4. **Final PageRank:** The values represent the relative importance of each node in the web graph.

The iterative nature and adjustment for dead nodes and spider traps ensure a fair distribution of rank across the web graph. [x]



Bloom filters can have false positives. Explain with example

<https://medium.com/blockchain-musings/a-case-of-false-positives-in-bloom-filters-da09ec487ff0>

With example, explain the big data algorithm for counting distinct elements in a data stream.

Write short note on Hive

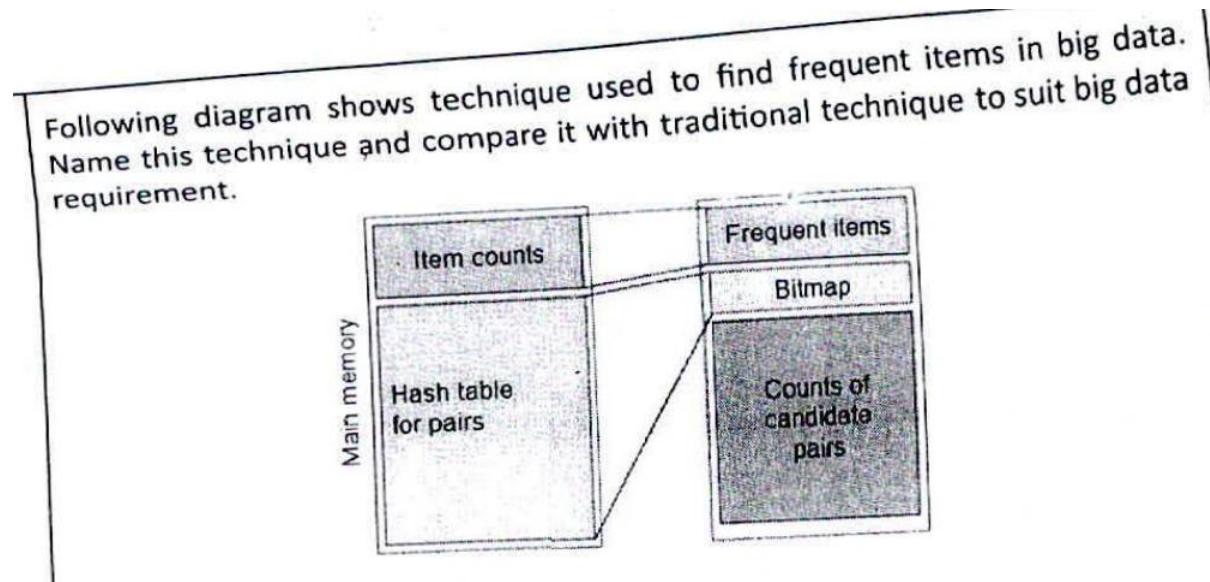
<https://www.javatpoint.com/hive-architecture>
<https://www.interviewbit.com/blog/hive-architecture/>

Write short note on HBase

<https://www.scaler.com/topics/hadoop/hbase-architecture/>

Write a short note on HDFS

<https://www.interviewbit.com/blog/hdfs-architecture/>



Park Yen Chu Algorithm

Compare traditional database management system vs stream management system

<https://www.geeksforgeeks.org/difference-between-dbms-and-dsms/>

ESE MAY 2023 - I

Consider a sample sales data having columns like

Order_id
Product_id
Product_category - (Furniture/Electronics/.....)
Quantity
CostOfEach - cost of one item
Order_Date - DD/MM/YYYY
Customer_name
Customer_city
Customer_state
Deal_Size - (Small/Medium/Large)
Shipment_mode - (air/land/sea)

Write Hadoop map reduce code in java for following analysis.

Find the most popular shipment mode in a given state. Take the state input from the user.

Find the most frequent Deal_size in a given city. Take the city input from the user.

Print top 3 product categories making maximum sale in each state.

To solve these analysis tasks, we can write a single Hadoop MapReduce program in Java that handles all three requirements. Here's the code:

```
```java
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
```

```

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SalesAnalysis {

 public static class SalesMapper extends Mapper<LongWritable, Text,
Text, Text> {
 public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
 String[] fields = value.toString().split(",");
 String state = fields[9];
 String city = fields[8];
 String shipmentMode = fields[11];
 String dealSize = fields[10];
 String productCategory = fields[2];
 context.write(new Text(state + "," + shipmentMode), new
Text("S"));
 context.write(new Text(city + "," + dealSize), new
Text("D"));
 context.write(new Text(state + "," + productCategory), new
Text("P"));
 }
 }

 public static class SalesReducer extends Reducer<Text, Text, Text,
Text> {
 public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
 Map<String, Integer> countMap = new HashMap<>();
 String state = null;
 String city = null;
 String type = null;
 for (Text value : values) {
 if (key.toString().startsWith("S")) {
 type = "S";
 state = key.toString().split(",")[0];
 } else if (key.toString().startsWith("D")) {
 type = "D";
 city = key.toString().split(",")[0];
 }
 }
 countMap.put(type + "-" + state, countMap.getOrDefault(type +
"-" + state, 0) + 1);
 countMap.put(type + "-" + city, countMap.getOrDefault(type +
"-" + city, 0) + 1);
 context.write(new Text(key.toString() + ":" + countMap.get(
type + "-" + state) + "," + countMap.get(type + "-" + city)));
 }
 }
}

```

```

 } else {
 type = "P";
 state = key.toString().split(",")[0];
 }
 String val = value.toString();
 countMap.put(val, countMap.getOrDefault(val, 0) + 1);
}
if (type.equals("S")) {
 int maxCount = 0;
 String maxVal = null;
 for (Map.Entry<String, Integer> entry :
countMap.entrySet()) {
 if (entry.getValue() > maxCount) {
 maxCount = entry.getValue();
 maxVal = entry.getKey();
 }
 }
 context.write(new Text("Most popular shipment mode in "
+ state), new Text(maxVal));
} else if (type.equals("D")) {
 int maxCount = 0;
 String maxVal = null;
 for (Map.Entry<String, Integer> entry :
countMap.entrySet()) {
 if (entry.getValue() > maxCount) {
 maxCount = entry.getValue();
 maxVal = entry.getKey();
 }
 }
 context.write(new Text("Most frequent deal size in " +
city), new Text(maxVal));
} else {
 TreeMap<Integer, String> topProducts = new TreeMap<>((a,
b) -> b.compareTo(a));
 for (Map.Entry<String, Integer> entry :
countMap.entrySet()) {
 topProducts.put(entry.getValue(), entry.getKey());
 if (topProducts.size() > 3) {
 topProducts.pollLastEntry();
 }
 }
 StringBuilder sb = new StringBuilder();
 for (Map.Entry<Integer, String> entry :
topProducts.entrySet()) {
 sb.append(entry.getValue()).append(",");
 }
}

```

```

 context.write(new Text("Top 3 product categories in " +
state), new Text(sb.toString()));
 }
}

public static void main(String[] args) throws Exception {
 Configuration conf = new Configuration();
 Job job = Job.getInstance(conf, "Sales Analysis");
 job.setJarByClass(SalesAnalysis.class);
 job.setMapperClass(SalesMapper.class);
 job.setReducerClass(SalesReducer.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(Text.class);
 FileInputFormat.addInputPath(job, new Path(args[0]));
 FileOutputFormat.setOutputPath(job, new Path(args[1]));
 System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
...

```

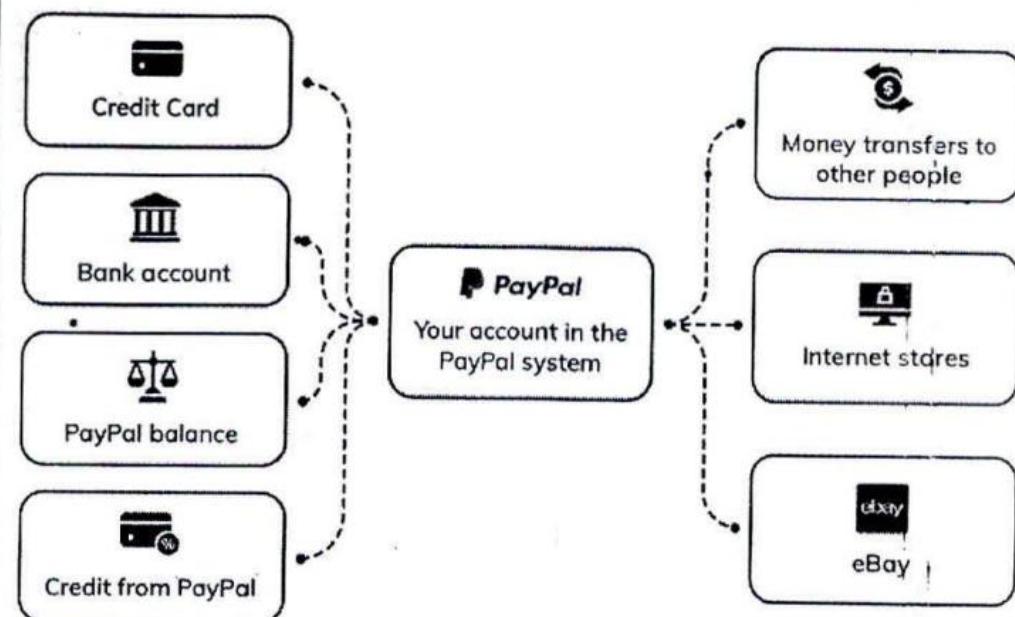
Here's how the code works:

1. The `SalesMapper` class takes each line of the input file (representing a sales record) and emits key-value pairs for three different types of analysis:
  - For shipment mode analysis, the key is `<state>,<shipment\_mode>`, and the value is a constant "S".
  - For deal size analysis, the key is `<city>,<deal\_size>`, and the value is a constant "D".
  - For product category analysis, the key is `<state>,<product\_category>`, and the value is a constant "P".
2. The `SalesReducer` class performs the actual analysis based on the key-value pairs emitted by the mapper:
  - For shipment mode analysis, it counts the occurrences of each shipment mode for a given state and emits the most popular shipment mode.
  - For deal size analysis, it counts the occurrences of each deal size for a given city and emits the most frequent deal size.
  - For product category analysis, it counts the occurrences of each product category for a given state, maintains a `TreeMap` to keep track of the top 3 product categories, and emits the top 3 product categories.
3. In the `main` method, we configure the MapReduce job, set the mapper and reducer classes, and specify the input and output paths.
4. After running the job, the output file will contain the results for all three analysis tasks:
  - The most popular shipment mode in a given state.
  - The most frequent deal size in a given city.

- The top 3 product categories making maximum sales in each state.

Note: This code assumes that the input file is in a comma-separated format, and the fields are in the order specified in the question. You may need to modify the code if your input file format or field order is different.

Q2[A]



PayPal was one of the first payment solutions to try and civilize online payments. PayPal handles a large volume of financial transactions every day, generating a large amount of data related to payments, transactions, user behavior, and more. Today, PayPal has over 377 million active accounts in 200 global markets. PayPal has also made significant acquisitions in the past two years. The transactions processed by PayPal occur in real-time, requiring fast data processing and analysis to keep up with the pace of data generation. 9.3 billion transactions happened on PayPal in 2021. Considering the above aspects of the case study, justify why PayPal can be considered as a big data case study.

Based on the information provided in the image and description, PayPal can be considered a big data case study for the following reasons:

1. Large volume of financial transactions: PayPal handles a massive volume of financial transactions every day, as mentioned in the text. Processing and analyzing such a vast amount of transactional data requires efficient big data solutions.
2. Real-time data processing: The transactions processed by PayPal occur in real-time, necessitating fast data processing and analysis capabilities to keep up with the pace of data

generation. Big data technologies are well-suited for handling and analyzing large volumes of data in near real-time.

3. User behavior and money transfers: PayPal facilitates money transfers between users and internet stores, capturing valuable data about user behavior, preferences, and spending patterns. Analyzing this data can provide valuable insights into consumer trends and patterns, which is a common use case for big data analytics.
4. Global reach and user base: With over 377 million active accounts in 200 global markets, PayPal generates data from a massive user base spanning multiple countries and regions. Handling and analyzing data at such a global scale requires scalable and distributed big data solutions.
5. Data-driven decision-making: The large amount of transactional data and user information available to PayPal can be leveraged for data-driven decision-making, risk management, fraud detection, targeted marketing, and other business intelligence applications, which are common use cases for big data analytics.
6. Acquisitions and growth: PayPal has made significant acquisitions in recent years, which likely contributed to an increase in the volume and diversity of data they handle. Big data technologies are essential for integrating and analyzing data from multiple sources and systems.
7. Competitive advantage: Effective big data analytics can provide PayPal with a competitive advantage by enabling better understanding of customer behavior, optimizing operations, identifying new business opportunities, and making data-driven decisions to improve overall performance and customer experience.

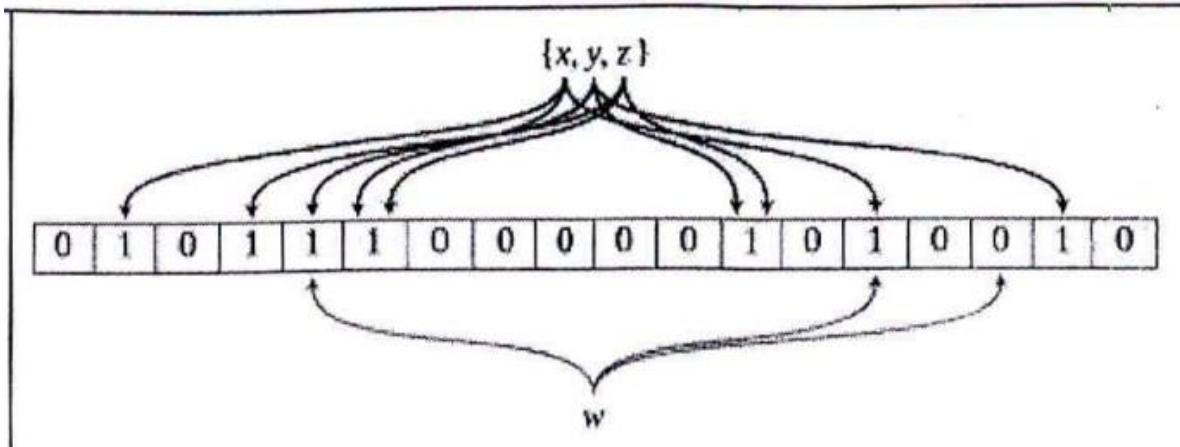
Given the large volume of transactions, real-time processing requirements, global reach, data-driven decision-making needs, and potential for leveraging data for business intelligence and insights, PayPal's operations and data management requirements align with the characteristics of a big data case study.

Apply the Count min sketch algorithm to find the final sketch after inserting data stream A,C,A,D,B,A,C,B,D,A,D using 3 hash functions H1,H2 and H3 defined as follows. Draw a separate sketch of every iteration. Find the occurrence of D in the final sketch. Does it match with actual occurrence? Justify your answer.

	A	B	C	D
H1	1	3	2	3
H2	2	3	2	2
H3	1	3	1	3

Done

Following diagram shows a sketch of the bloom filter after inserting x,y,z and w in that order. With the help of this sketch show that Bloom filter does not support deletion.



Bloom Filters are probabilistic data structures that are used to test whether an element is a member of a set. They are efficient and can tell you if an element is definitely not in the set or possibly in the set<sup>1</sup>.

However, standard Bloom Filters do not support deletion of elements. The reason for this is tied to how Bloom Filters work. When an element is added to the Bloom Filter, it is hashed by multiple hash functions, each producing an index in a bit array. These bits are then set to 1<sup>1</sup>.

If we were to delete an element, we would need to reset these bits back to 0. However, because of the nature of Bloom Filters, multiple elements could map to the same bits due to hash collisions<sup>2</sup>. If we reset these bits, we might inadvertently remove other elements that also map to these bits, introducing false negatives<sup>32</sup>.

This is why deletion is not supported in a standard Bloom Filter. However, there are variants of Bloom Filters, such as Counting Bloom Filters and Invertible Bloom Filters, which do support deletion<sup>4</sup>. But these come with their own trade-offs, such as increased space complexity<sup>4</sup>.

A pair which is not frequent can be counted as a frequent pair in the PCY algorithm. Do you agree? Illustrate this with the example. Discuss improved PCY algorithm to overcome this problem.

Yes, I agree. In the PCY (Park-Chen-Yu) algorithm, a pair that is not frequent can be counted as a frequent pair due to hash collisions<sup>12</sup>. This is because the PCY algorithm uses a hash function to map itemsets to hash buckets, and different itemsets can end up in the same bucket if they have the same hash value<sup>12</sup>.

Let's illustrate this with an example<sup>1</sup>:

Consider the following transactions with a threshold minimum value as 3 and a Hash function as  $(i^*j) \bmod 10$ :

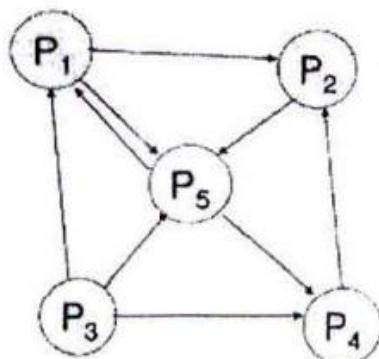
T1 = {1, 2, 3}  
T2 = {2, 3, 4}  
T3 = {3, 4, 5}  
T4 = {4, 5, 6}  
T5 = {1, 3, 5}  
T6 = {2, 4, 6}  
T7 = {1, 3, 4}  
T8 = {2, 4, 5}  
T9 = {3, 4, 6}  
T10 = {1, 2, 4}  
T11 = {2, 3, 5}  
T12 = {2, 4, 6}

In the first pass of the PCY algorithm, we count the frequency of each item and each pair of items, and we hash each pair to a bucket<sup>1</sup>. In the second pass, we only count the pairs that are both frequent and hashed to a frequent bucket<sup>1</sup>.

However, due to the nature of the hash function, different pairs can hash to the same bucket. For example, the pairs (1, 3) and (2, 4) both hash to the bucket 3 using the hash function  $(i^*j) \bmod 10$ <sup>1</sup>. If the bucket 3 is frequent but the pair (1, 3) is not, the pair (1, 3) can still be counted as a frequent pair in the PCY algorithm<sup>1</sup>.

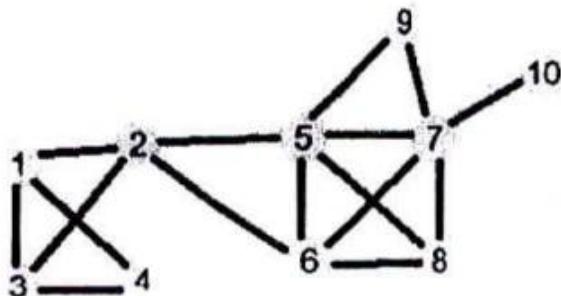
To overcome this problem, we can use improved versions of the PCY algorithm, such as the Multistage algorithm or the Multihash algorithm<sup>34</sup>. These algorithms use multiple hash functions or multiple passes to reduce the chance of hash collisions and improve the accuracy of the frequent itemsets<sup>34</sup>. However, these improvements come with their own trade-offs, such as increased computational cost or memory usage<sup>34</sup>.

Use the HITS algorithm to identify authority and hub pages in the following graph. Solve up to 5 iterations with normalization.



Done

Consider the following graph as a part of a social network graph. Identify how many communities can be formed using the Clique Percolation Method for k=3. Clearly state the nodes in each community. Show all steps.



Done

Find Eigenvalues and Eigenvectors of

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 3 \\ 3 & 3 & 20 \end{bmatrix}$$

Consider a page rank algorithm of the form  $r = Mr$ . Assume above matrix represents M of page rank algorithm. What is the significance of an eigen vector in finding the page rank of the given M.

Pending

Explain with example how Singular Value Decomposition can be used for Big Data Processing

Sure, let's consider a practical example of how Singular Value Decomposition (SVD) can be used for Big Data processing.

Suppose we have a dataset of movie ratings. Each row in the dataset represents a user, and each column represents a movie. The entries in the dataset are the ratings given by users to movies. This is a common scenario in recommendation systems.

Let's consider a small subset of this data for simplicity:

	<b>Movie</b> <b>1</b>	<b>Movie</b> <b>2</b>	<b>Movie</b> <b>3</b>
<b>User</b> <b>1</b>	4	5	1
<b>User</b> <b>2</b>	5	1	4
<b>User</b> <b>3</b>	1	4	5

We can represent this data as a matrix **A**:

$$A = \begin{bmatrix} 4 & 5 & 1 \\ 5 & 1 & 4 \\ 1 & 4 & 5 \end{bmatrix}$$

We can apply SVD to this matrix to get the matrices **U**, **Σ**, and **V^T**:

$$A = U \cdot \Sigma \cdot V^T$$

The **U** matrix represents how much each user “belongs” to each concept. The **Σ** matrix contains the singular values, which represent the “strength” of each concept. The **V^T** matrix represents how much each movie “belongs” to each concept.

By keeping only the top **k** singular values (and corresponding columns of **U** and rows of **V^T**), we can get a reduced representation of our original data. This is known as **truncated SVD**.

This reduced representation can be used for various tasks such as recommendation (suggesting movies that a user might like), or finding similar movies or users.

#### **Advantages of using SVD in Big Data processing:**

- Dimensionality Reduction:** SVD can significantly reduce the dimensionality of the data, making it easier to process and visualize.
- Noise Reduction:** SVD can help remove noise from the data by eliminating the components with small singular values, which often represent noise or less important details.
- Latent Semantic Analysis:** In text mining, SVD can help identify the underlying themes or topics in the documents.

#### **Disadvantages of using SVD in Big Data processing:**

1. **Computational Complexity:** SVD can be computationally expensive for very large datasets.
2. **Sparsity:** SVD does not handle sparse data well. In many Big Data scenarios (like our movie ratings example), the data is often sparse (i.e., most of the entries are zero). Other techniques like sparse SVD or matrix factorization methods might be more suitable in these cases.
3. **Interpretability:** The “concepts” derived from SVD are not always easy to interpret.

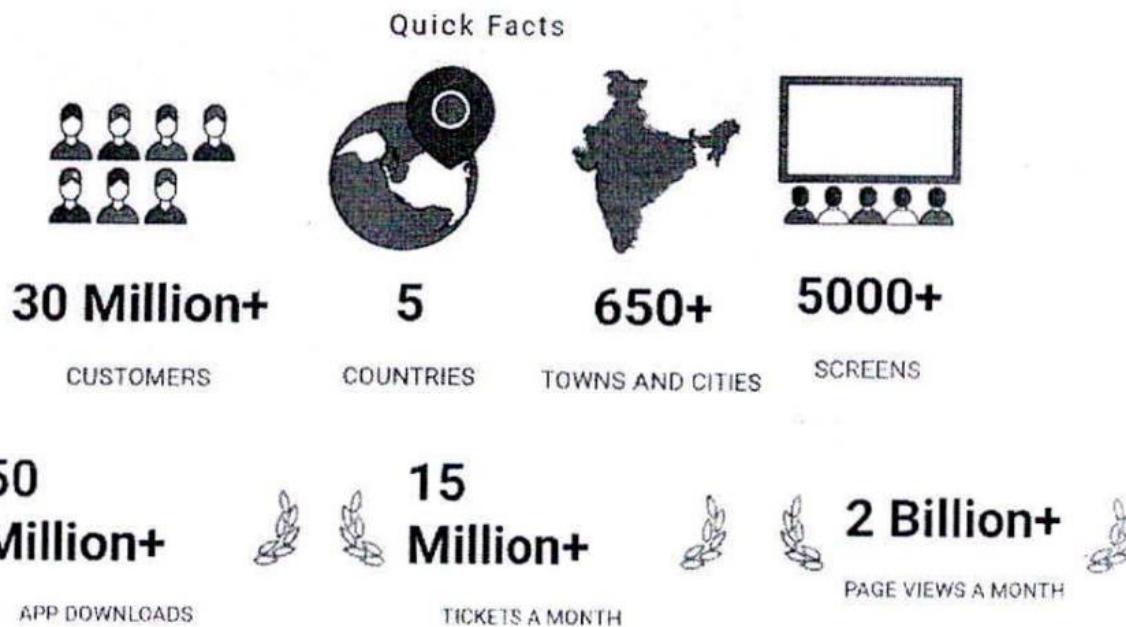
Given the following data use PCA to reduce the dimensions.

Feature	Ex1	Ex2	Ex3	Ex4
X	4	8	13	7
Y	11	4	5	14

done

# ESE DEC 2022

Observe the following statistics about BookMyShow and justify why you think it is an example of Big Data. Ensure your answer is based on the details mentioned in the case study.



**Volume - 1.5M**  
**Velocity - 1.5M**  
**Variety - 1.5M**  
**Value - 1.5M**  
**Veracity - 1.5M**  
**Visualization - 1.5M**  
**Presentation/flow - 1M**

Based on the 7 V's of big data (Volume, Velocity, Variety, Veracity, Value, Visualization, and Volatility), BookMyShow can be considered an example of a big data case study due to the following reasons:

1. Volume: The statistics indicate that BookMyShow handles a massive volume of data, including data from 30 million+ customers, 15 million+ tickets sold per month, 2 billion+ page views per month, and data from 5000+ screens across multiple locations.

2. Velocity: The high volume of transactions (15 million+ tickets sold per month) and web traffic (2 billion+ page views per month) suggest that BookMyShow deals with data at a high velocity or speed, requiring real-time or near real-time data processing capabilities.
3. Variety: BookMyShow likely handles a variety of data types, including structured data (transaction records, seat availability, movie schedules), semi-structured data (user preferences, reviews), and unstructured data (social media interactions, customer feedback, multimedia content).
4. Veracity: With a large customer base and high transaction volume, ensuring data veracity (accuracy, consistency, and trustworthiness) is crucial for BookMyShow to maintain data integrity and make reliable decisions.
5. Value: The ability to extract valuable insights from the vast amounts of data generated by BookMyShow's operations can provide significant business value, such as understanding customer behavior, optimizing ticket pricing, improving service offerings, and enhancing the overall user experience.
6. Visualization: Effectively visualizing and presenting complex data patterns and insights derived from BookMyShow's data can aid in better decision-making and communication with stakeholders.
7. Volatility: The entertainment industry, which BookMyShow operates in, is subject to rapid changes in customer preferences, trends, and market dynamics. Handling and adapting to the volatility of data is essential for BookMyShow to stay relevant and competitive.

By considering the 7 V's of big data, it is evident that BookMyShow's operations involve handling and processing large volumes of diverse data at high velocities, while ensuring data veracity, extracting valuable insights, effectively visualizing data, and adapting to the volatility of the entertainment industry. These characteristics align with the requirements and challenges of a big data case study, making BookMyShow a suitable example in this context.

Consider a case study mentioned in question no 1. You are appointed as a data analyst in BookMyShow and assigned some task of data analyst.

List down the important columns/features which you think are important in the dataset. Describe your dataset in brief.

Write 6 questions about this dataset which will give you useful insights.

What type of data visualization you will use to support these insights. Draw diagrams

As a data analyst at BookMyShow, I would consider the following columns/features to be important in the dataset:

1. Customer ID
2. Movie Title
3. Movie Genre
4. Show Date and Time
5. Theater Name and Location
6. Number of Tickets Purchased
7. Ticket Price
8. Total Purchase Amount
9. Payment Method
10. Customer City/Region
11. Customer Age/Age Group
12. Customer Gender
13. Booking Date and Time
14. Booking Channel (Website, Mobile App, etc.)
15. Promotional Offer/Discount Applied

The dataset would primarily consist of transactional data related to movie ticket bookings, customer information, and movie details. It would capture the behavior and preferences of customers across different regions, movies, genres, theaters, and booking channels.

Here are six questions about this dataset that could provide useful insights:

1. What are the most popular movie genres among different age groups and regions?
2. How do ticket sales and revenue vary across different show timings, days of the week, and seasons?
3. Which theaters and locations generate the highest ticket sales and revenue?
4. What is the impact of promotional offers and discounts on ticket sales and customer behavior?
5. How do booking patterns differ between various booking channels (website, mobile app, etc.)?
6. Can we identify potential customer segments based on booking behavior and preferences?

To support these insights, I would use the following data visualizations:

1. Bar Charts and Pie Charts: To visualize the distribution of ticket sales, revenue, and customer demographics across different categories (genres, regions, age groups, etc.).
2. Line Graphs and Area Charts: To analyze trends and patterns in ticket sales and revenue over time, such as by show timings, days of the week, or seasons.
3. Scatter Plots: To explore correlations between different variables, such as ticket price and sales volume or customer age and preferred genres.
4. Heat Maps: To visualize the performance of theaters and locations based on ticket sales and revenue.
5. Geographical Maps: To visualize spatial patterns and regional variations in customer behavior and ticket sales.
6. Treemaps or Sunbursts: To display hierarchical data, such as ticket sales breakdown by genre, region, and age group.

These visualizations can provide a clear and intuitive representation of the insights derived from the data, aiding in better understanding and communication of the findings.

Consider the FIFA 20 dataset. Part of the dataset is shown below. Write Hadoop map reduce code in java to answer following queries.

Player Name	nationality	club	player_positions	Preferred Foot	International reputation
L. Messi	Argentina	FC Barcelona	CF	Left	5
Cristiano Ronaldo	Portugal	Real Madrid	LW, LM	Right	4
A. Robben	Netherlands	FC Bayern München	RM, LM, RW	Left	3
Z. Ibrahimović	Sweden	Paris Saint-Germain	ST	Right	4
M. Neuer	Germany	FC Bayern München	GK	Right	4
L. Suárez	Uruguay	FC Barcelona	ST, CF	Right	3

For each club, find the total number of players nation-wise.  
Expected output will have 3 columns Club Name, Country name and Total player count

For each club, find the total number of Left and Right foot preferred players . Expected output will have 3 columns Club Name, Preferred Foot and Total player count

For each club, find the number of players for every player position. Expected output will have 3 columns Club Name, player position and Total player count

Find the average international reputation of every club. Expected output will have 2 columns Club name, avg. international reputation.

---

(a) You decide to use the page rank algorithm  $r = Mr$ , on some web graph whose matrix  $M$  is as given below where column represents outlink and row represents inlink.. Numbers in each cell of the matrix represent edge weight.

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 2 & 3 \\ 3 & 3 & 20 \end{pmatrix}$$

Compute rank vector  $r$ , using Eigen vector formulation.

pending

b) Draw a web graph with spider trap and a dead node. Using this graph describe the technique that can be used to find page rank of web pages in a web graph that you have drawn.

Derive the equation for bloom filter to find Optimum number of hash functions required for a given size of data and size of bloom filter. Write your observation about this equation.

Given:

- $n$ : the number of items you expect to have in your filter
- $m$ : the size of the Bloom filter (i.e., the number of bits)
- $k$ : the number of hash functions

The optimal number of hash functions  $k$  can be calculated using the following formula<sup>12</sup>:

$$k = (m/n) * \ln(2)$$

This formula is derived from the probability of a bit still being zero after n insertions with k hash functions, and minimizing the false positive probability<sup>12</sup>.

#### **Observations:**

- The equation shows that the optimal number of hash functions is proportional to the ratio of the size of the Bloom filter to the number of items stored, times the natural logarithm of 2<sup>12</sup>.
- This means that if we increase the size of the Bloom filter for a fixed number of items (i.e., increase m/n), we should increase the number of hash functions to minimize the probability of false positives<sup>12</sup>.
- Conversely, if we have a fixed Bloom filter size and increase the number of items (i.e., decrease m/n), we should decrease the number of hash functions<sup>12</sup>.
- However, in practice, the number of hash functions is usually chosen as a trade-off between the time it takes to compute the hash functions and the rate of false positives<sup>12</sup>.

## Draw and explain Spark Architecture

<https://www.interviewbit.com/blog/apache-spark-architecture/>

## Write a short note on YARN

<https://www.interviewbit.com/blog/yarn-architecture/>

## Discuss Data-Stream-Management System. With example, explain the FM algorithm.

<https://www.geeksforgeeks.org/data-stream-management-system-architecture/>

# ESE May 2022

Consider the following Case Study.

Chocolate Marketing company with large number of installed Automatic Chocolate Vending Machines (ACVM). Each ACVM sells five flavours (FL1, FL2, FL3, FL4 and FL5) KitKat, Milk, Fruit and Nut, Nougat and Oreo. Data get generated at various machine such as sales of chocolates, reports of unfilled or filled machine transaction data. Social network and web data on feedback and personalized messages based on interactions and human generated data on facial recognition of the buyers. The company uses this data for efficient and optimum planning of fill services for chocolates, sentiment analysis of buyer for specific flavours.

- i. Identify Different Big Data Types from the above case study.
- ii. Identify the challenges faced from large growth in volume of data.

**i. Different Big Data Types in the Case Study:**

1. **Structured Data:** This is data that is organized in a defined manner and can be easily processed and analyzed. In this case, the sales data from the Automatic Chocolate Vending Machines (ACVMs), which includes information about the number of chocolates sold, the flavors sold, and the transaction data, is structured data.
2. **Unstructured Data:** This is data that doesn't have a predefined model or isn't organized in a predefined manner. The feedback and personalized messages based on interactions from social networks and web data fall into this category.
3. **Semi-Structured Data:** This is data that is not organized according to a schema but contains tags that make it more processable than unstructured data. The reports of unfilled or filled machines could be considered semi-structured data.

4. **Big Data Generated by Humans:** This includes data generated through human interactions, such as facial recognition data of the buyers.

## ii. Challenges Faced from Large Growth in Volume of Data:

1. **Storage:** As the volume of data grows, so does the need for storage. This can be a challenge as it requires significant storage resources, which can be expensive.
2. **Processing:** Large volumes of data require powerful processing capabilities. This can be a challenge, especially when real-time processing is required.
3. **Analysis:** Analyzing large volumes of data can be complex and time-consuming. It requires sophisticated algorithms and analytics tools.
4. **Data Quality:** With large volumes of data, ensuring the quality and accuracy of the data can be challenging.
5. **Security and Privacy:** Protecting the data and ensuring privacy is a major challenge, especially with sensitive data like facial recognition data.
6. **Integration:** Integrating different types of data (structured, unstructured, semi-structured) can be challenging but is often necessary for comprehensive analysis.

I hope this helps! Let me know if you have any other questions. 😊

Compare traditional versus Big data analytics with example

<https://www.geeksforgeeks.org/difference-between-traditional-data-and-big-data/>

Describe the component of SPARK ecosystem

<https://www.javatpoint.com/apache-spark-components>

Draw and explain architecture of Apache Pig

<https://data-flair.training/blogs/pig-architecture/>

Consider a .txt file of size 1024 MB. will be stored on HDFS.  
With the help of diagram demonstrate the block replication and rack awareness in HDFS.

Consider the pass consist of 1, 2, 3, 1, 2, 3, 4, 1, 2, 4. Hash function is  $6X+1 \bmod 5$ . How many numbers of distinct elements presents in the pass?

done

Apply the Misra Giles algorithm on the data stream consist of 2 3,9,5,7,9,9. With K=2

done

Identify best HUB and Authority for the given adjacency matrix. Calculate the HUB and Authority score using HITS algorithm for K=2

$$A = \begin{vmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

done

Apply PCY algorithm on the following transaction to find the candidate sets (frequent sets). Use buckets and concepts of Mapreduce to solve the above problem. Consider Hash Function  $=(i*j) \bmod 10$  and Threshold value =3

T1= {6,5,4} T2= {5,4,3} T3= {4,3,2} T4= {3,2,1} T5= {6,4,2}

T6= {5,3,1} T7= {6,4,3} T8= {5,3,2} T9= {4,2,1} T10= {6,5,3}

done

Consider two matrices in the format

M= [m11, m12, m21, m22]

N= [n11, n12,n21,n22]

where m11 can be interpreted as value in the first row and first column of matrix M and so on. The actual matrices are: M= [1,9,5,4] and N= [4,3,6,7] In case of multiplication of M and N using 2 phase MapReduce, what will be the output of the second map phase?

## ESE November 19

Write a Map reduce program using Java - To count the words excluding following words :- the, this, there, that, these. Note : No need to write the driver class

Sure, here's a simple MapReduce program in Java that counts the occurrences of words in a text, excluding the words "the", "this", "there", "that", "these".

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCount {

 public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

 private final static IntWritable one = new IntWritable(1);
 private Text word = new Text();

 public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
 StringTokenizer itr = new StringTokenizer(value.toString());
 while(itr.hasMoreTokens()){
 word.set(itr.nextToken());
 context.write(word, one);
 }
 }
 }

 public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable>{
 private IntWritable result = new IntWritable();

 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
 int sum = 0;
 for(IntWritable val : values){
 sum += val.get();
 }
 result.set(sum);
 context.write(key, result);
 }
 }
}
```

```

 while (itr.hasMoreTokens()) {
 String currentWord = itr.nextToken();
 if(!currentWord.equals("the") && !currentWord.equals("this") &&
!currentWord.equals("there") && !currentWord.equals("that") &&
!currentWord.equals("these")) {
 word.set(currentWord);
 context.write(word, one);
 }
 }
 }
}

public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
 private IntWritable result = new IntWritable();

 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
 int sum = 0;
 for (IntWritable val : values) {
 sum += val.get();
 }
 result.set(sum);
 context.write(key, result);
 }
}
}

```

This program consists of a **Mapper** and a **Reducer**. The **Mapper** tokenizes the input text into words and emits a key-value pair for each word not in the exclusion list, with the word as the key and **1** as the value. The **Reducer** sums up the values for each word and emits a key-value pair with the word and its total count. Note that this program doesn't include the driver class as per your request. You would need to write a driver class to run this MapReduce job. Please let me know if you need help with that as well.

## Write a pseudo code for computing difference operation using Map Reduce.

Sure, here's a high-level pseudocode for computing the difference operation ( $A - B$ ) using MapReduce. The idea is to tag the records from each dataset ( $A$  and  $B$ ) and then filter out the records in the reduce phase.

Mapper:

Input: datasets A and B  
for each record r in A:

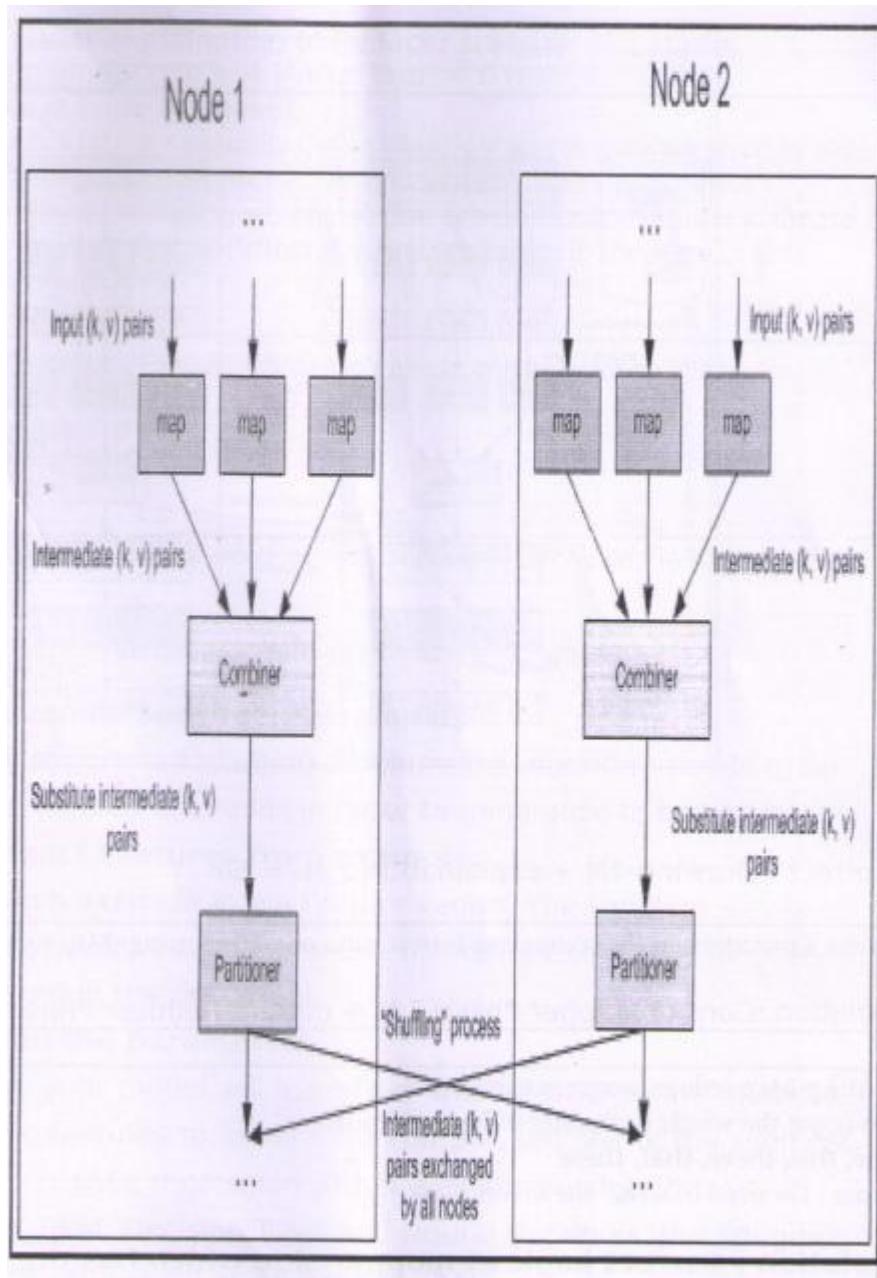
```
emit(key = r, value = ("A", r))
for each record r in B:
 emit(key = r, value = ("B", r))
```

Reducer:

```
Input: key = record, values = list of ("A", record) and/or ("B", record)
A_list = []
B_list = []
for each v in values:
 if v is from A:
 add v to A_list
 else if v is from B:
 add v to B_list
if A_list is not empty and B_list is empty:
 for each v in A_list:
 emit(v)
```

In the Mapper phase, each record from dataset A is tagged with “A” and each record from dataset B is tagged with “B”. In the Reducer phase, if a record is only in A (i.e., there are no corresponding records in B), it is emitted as part of the difference ( $A - B$ ). This pseudocode assumes that the datasets do not contain duplicate records. If they do, you would need to handle duplicates appropriately in the Reducer phase. Please note that this is a high-level pseudocode and the actual implementation may vary based on the specific requirements and the MapReduce framework you are using. Let me know if you need further clarification or help with anything else.

Justify - Combiner can be viewed as mini-reducers in the map phase. Draw a scenario to explain combiner in Map and Reduce phase



### Justification: Combiner as Mini-Reducers in the Map Phase

A combiner in the MapReduce framework is a mini-reducer that performs a local reduction process on the intermediate map output. It is used to optimize the MapReduce process by reducing the amount of data that needs to be shuffled across the network to the reducers. The combiner works by aggregating intermediate key-value pairs with the same key on each mapper node before they are sent to the reducer.

## Scenario Explanation

Let's consider a scenario where we need to count the number of occurrences of each word in a large collection of documents. This scenario will help us understand the role of the combiner in the Map and Reduce phases.

### Map Phase

1. Input Split: Each mapper processes a split of the input data. Assume we have two nodes, Node 1 and Node 2, each processing a different set of documents.
2. Map Function: The map function processes each document and emits intermediate key-value pairs where the key is a word and the value is 1.
  - Example:
    - Node 1 processes "dog cat dog".
    - Node 2 processes "cat fish fish".
  - Intermediate output:
    - Node 1: {("dog", 1), ("cat", 1), ("dog", 1)}
    - Node 2: {("cat", 1), ("fish", 1), ("fish", 1)}

### Combiner Phase

3. Combiner Function: The combiner performs local aggregation on the intermediate output. It sums up the counts for each word within the same node.
  - Example:
    - Node 1: Combines {("dog", 1), ("dog", 1)} → {("dog", 2)}, {("cat", 1)}
    - Node 2: Combines {("fish", 1), ("fish", 1)} → {("fish", 2)}, {("cat", 1)}
  - Combined output:
    - Node 1: {("dog", 2), ("cat", 1)}
    - Node 2: {("fish", 2), ("cat", 1)}
4. Partitioner: The partitioner distributes the combined intermediate key-value pairs to the reducers based on the key.

### Shuffle and Sort Phase

5. Shuffle and Sort: The intermediate key-value pairs are shuffled and sorted so that all values for the same key are sent to the same reducer.
  - Example:
    - Key "dog" is sent to Reducer 1.
    - Key "cat" is sent to Reducer 2.
    - Key "fish" is sent to Reducer 3.

### Reduce Phase

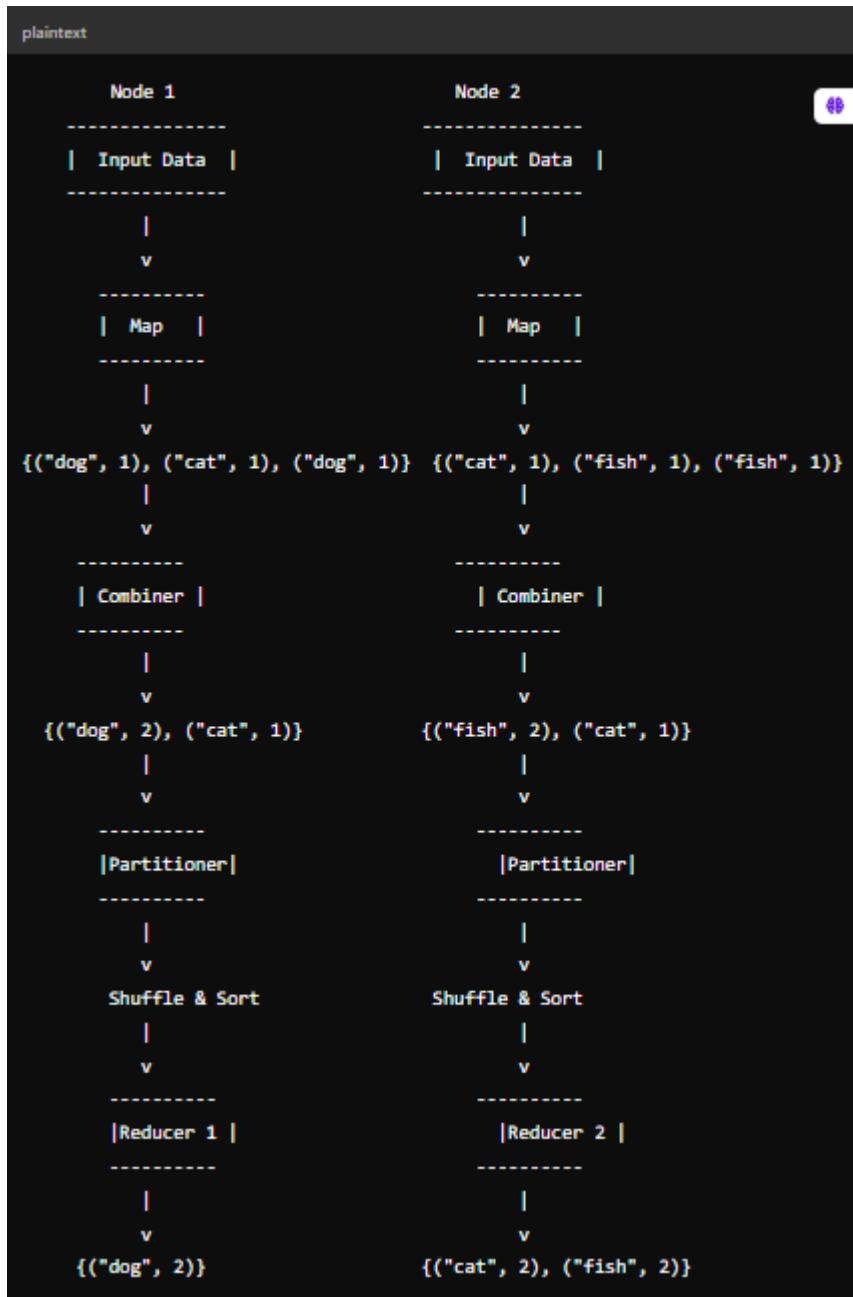
6. Reduce Function: The reducer processes the sorted intermediate key-value pairs and produces the final output.
  - Example:
    - Reducer 1 receives {("dog", [2])} and outputs {("dog", 2)}.
    - Reducer 2 receives {("cat", [1, 1])} and outputs {("cat", 2)}.
    - Reducer 3 receives {("fish", [2])} and outputs {("fish", 2)}.

## Diagram Explanation

The diagram provided in the image can be interpreted as follows:

1. Map Stage: Each node runs the map function on its input split and produces intermediate key-value pairs.
2. Combiner Stage: The intermediate key-value pairs are then fed into the combiner, which acts as a mini-reducer to aggregate local results.
3. Partitioner Stage: The combined intermediate key-value pairs are partitioned based on their keys.
4. Shuffle Process: The partitioned intermediate key-value pairs are exchanged among all nodes to ensure that all values for the same key are sent to the same reducer.
5. Reducer Stage: Finally, each reducer processes its assigned key-value pairs to produce the final output.

## Visualization



This diagram helps illustrate how the combiner functions as a mini-reducer by aggregating the intermediate data locally before sending it to the reducers, thereby optimizing the MapReduce process.

## Write a pseudocode for computing Intersection operation using Map Reduce.

Sure, here's a high-level pseudocode for computing the difference operation ( $A - B$ ) using MapReduce. The idea is to tag the records from each dataset ( $A$  and  $B$ ) and then filter out the records in the reduce phase.

Mapper:

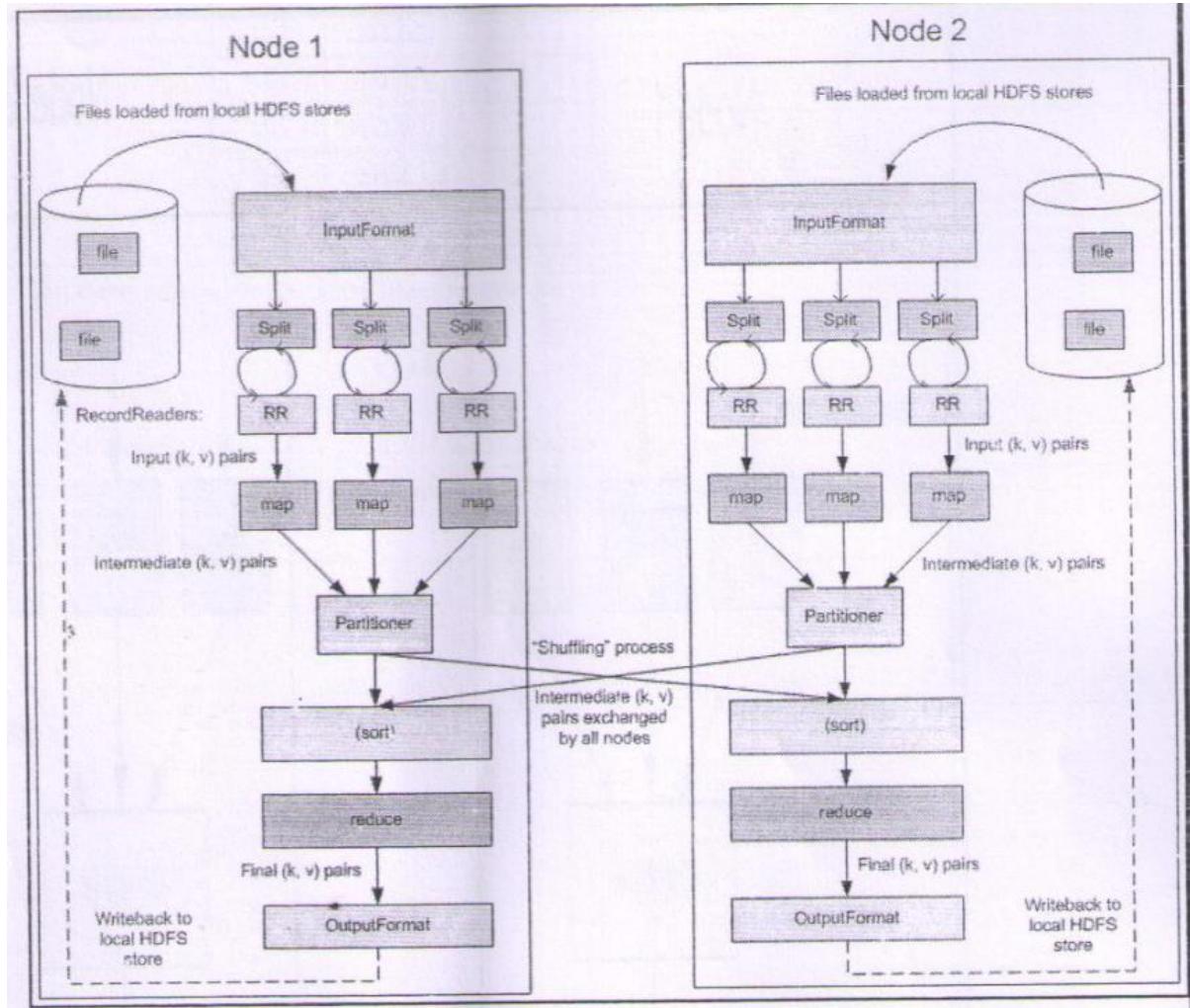
```
Input: datasets A and B
for each record r in A:
 emit(key = r, value = ("A", r))
for each record r in B:
 emit(key = r, value = ("B", r))
```

Reducer:

```
Input: key = record, values = list of ("A", record) and/or ("B",
record)
A_list = []
B_list = []
for each v in values:
 if v is from A:
 add v to A_list
 else if v is from B:
 add v to B_list
if A_list is not empty and B_list is empty:
 for each v in A_list:
 emit(v)
```

In the Mapper phase, each record from dataset A is tagged with “A” and each record from dataset B is tagged with “B”. In the Reducer phase, if a record is only in A (i.e., there are no corresponding records in B), it is emitted as part of the difference ( $A - B$ ). This pseudocode assumes that the datasets do not contain duplicate records. If they do, you would need to handle duplicates appropriately in the Reducer phase. Please note that this is a high-level pseudocode and the actual implementation may vary based on the specific requirements and the MapReduce framework you are using. Let me know if you need further clarification or help with anything else.

What is partitioning phase - Draw and explain need of partitioners in map reduce you can take any relevant scenario and explain this



## Partitioning Phase in MapReduce

The partitioning phase in MapReduce is crucial for distributing the intermediate key-value pairs generated by the map tasks to the appropriate reducer tasks. The partitioner determines which reducer will process each key-value pair, ensuring that all values associated with the same key are processed together by the same reducer.

## Need for Partitioners

Partitioners are needed in the MapReduce framework for the following reasons:

1. **Load Balancing:** Partitioners help distribute the data evenly across reducers, avoiding scenarios where some reducers are overwhelmed with too much data while others have too little.

2. Grouping by Key: They ensure that all values associated with the same key are sent to the same reducer, enabling the correct execution of the reduce function, which processes all values for a given key together.
3. Efficiency: By correctly partitioning data, the network traffic during the shuffle phase is optimized, reducing the overall processing time.

## Scenario: Word Count

Let's consider a simple word count example to explain the need for partitioners:

1. Map Phase:
  - Each mapper reads a split of the input data and emits intermediate key-value pairs (word, 1) for each word it encounters.
  - Example:
    - Node 1 processes "cat dog cat".
    - Node 2 processes "dog fish fish".
  - Intermediate key-value pairs:
    - Node 1: {("cat", 1), ("dog", 1), ("cat", 1)}
    - Node 2: {("dog", 1), ("fish", 1), ("fish", 1)}
2. Partitioning Phase:
  - The partitioner determines which reducer each key-value pair should go to. For simplicity, assume we have two reducers.
  - A common partitioning strategy is to use a hash function on the key and modulo by the number of reducers to decide the reducer:
    - Reducer 1: keys that hash to even numbers.
    - Reducer 2: keys that hash to odd numbers.
  - Applying this to our example:
    - Hash("cat") % 2 = 0 → Reducer 1
    - Hash("dog") % 2 = 1 → Reducer 2
    - Hash("fish") % 2 = 0 → Reducer 1
  - Partitioned intermediate key-value pairs:
    - Node 1: {("cat", 1), ("cat", 1)} → Reducer 1, {("dog", 1)} → Reducer 2
    - Node 2: {("fish", 1), ("fish", 1)} → Reducer 1, {("dog", 1)} → Reducer 2
3. Shuffle and Sort Phase:
  - Intermediate key-value pairs are shuffled to the appropriate reducers and sorted by key.
  - Example:
    - Reducer 1 receives: {("cat", [1, 1]), ("fish", [1, 1])}
    - Reducer 2 receives: {("dog", [1, 1])}
4. Reduce Phase:
  - Each reducer processes the key-value pairs, aggregating the counts for each word.
  - Example:
    - Reducer 1 outputs: {("cat", 2), ("fish", 2)}
    - Reducer 2 outputs: {("dog", 2)}

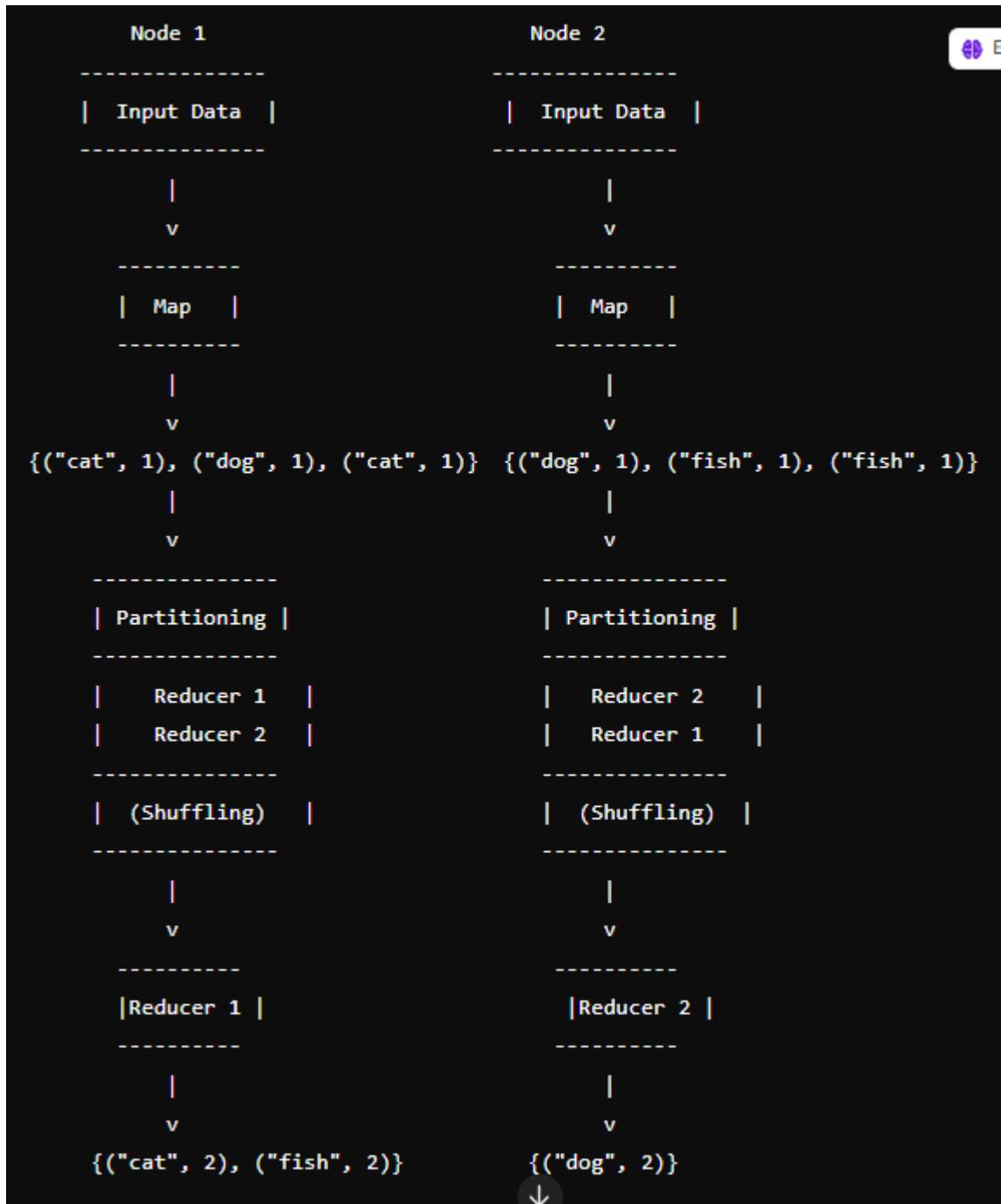
## Diagram Explanation

The provided image helps illustrate these concepts:

1. InputFormat: Splits the input files into chunks processed by individual mappers.
2. RecordReaders (RR): Read records from the input splits and generate key-value pairs.
3. Map Phase: Each mapper processes its split and emits intermediate key-value pairs.
4. Partitioner: The partitioner assigns each key-value pair to a reducer based on the key.
5. Shuffle Process: Key-value pairs are exchanged between nodes to ensure that all pairs with the same key end up at the same reducer.
6. Sort: Intermediate key-value pairs are sorted by key at each reducer.
7. Reduce Phase: Reducers process the sorted key-value pairs and generate the final output.
8. OutputFormat: Writes the final key-value pairs to the output files.

## Visualization of the Partitioning Process

Here's a simple diagram to visualize the partitioning process in the MapReduce framework:



This diagram illustrates how the partitioner ensures that all intermediate key-value pairs with the same key are sent to the same reducer, enabling the correct aggregation and final output generation in the reduce phase.

## What is the use of Oozie co-ordinator?

The Oozie Coordinator is a system that allows users to define and execute recurrent and interdependent workflow jobs, also known as data application pipelines<sup>1</sup>. Here are some of its uses:

- **Scheduling Complex Workflows:** Coordinator applications allow users to schedule complex workflows, including workflows that are scheduled regularly<sup>2</sup>.
- **Trigger-Based Workflow Execution:** The Oozie Coordinator models the workflow execution triggers in the form of time, data, or event predicates<sup>2</sup>. The workflow job mentioned inside the Coordinator is started only after the given conditions are satisfied<sup>2</sup>.
- **Handling Real-World Data Application Pipelines:** Real-world data application pipelines have to account for reprocessing, late processing, catchup, partial processing, monitoring, notification, and SLAs<sup>1</sup>.

In essence, the Oozie Coordinator is used to manage the complexities of scheduling and executing workflows in a Hadoop environment, making it an essential tool for managing big data workflows. Let me know if you need more information on this or any other topic! 😊

What is a Reducer side join ? Write a pseudocode for the flata set I have following schema : SchCustomerSale(TranID, Date, CustID, Amount, Equipment, city, state, mode )

[A Reducer side join is a process where the join operation is performed in the reducer phase of the MapReduce framework<sup>12</sup>. It is used when both datasets are too large to fit into memory<sup>3</sup>.](#) In this join, the map side processing emits join keys and corresponding tuples of both tables. [All tuples with the same join key are grouped into the same reducer, which then joins the records with the same join key<sup>4</sup>.](#)

Here's a high-level pseudocode for a Reducer side join operation on a dataset with the schema `SchCustomerSale(TranID, Date, CustID, Amount, Equipment, city, state, mode)`:

Mapper:

```
Input: datasets A and B
for each record r in A:
 emit(key = r.CustID, value = ("A", r))
for each record r in B:
 emit(key = r.CustID, value = ("B", r))
```

Reducer:

```
Input: key = CustID, values = list of ("A", record) and/or ("B", record)
A_list = []
B_list = []
for each v in values:
 if v is from A:
 add v to A_list
```

```

else if v is from B:
 add v to B_list
if A_list is not empty and B_list is not empty:
 for each v in A_list:
 for each w in B_list:
 emit(v, w)

```

In this pseudocode, **A** and **B** represent two datasets that you want to join. The **Mapper** emits a key-value pair for each record in the datasets, with the **CustID** as the key and a tuple containing the dataset identifier (“A” or “B”) and the record as the value. The **Reducer** then performs the join operation for each **CustID**, emitting a key-value pair for each joined record. Note that this pseudocode assumes that you want to perform an inner join. If you want to perform a different type of join (e.g., left outer join, right outer join, full outer join), you would need to modify the **Reducer** logic accordingly. Let me know if you need further clarification or help with anything else.

## Write CURE Algorithm. Explain initialization phase and completion phase.

[Sure, let's start with a high-level overview of the CURE \(Clustering Using Representatives\) algorithm<sup>12</sup>.](#) [CURE is a hierarchical clustering algorithm that is more robust to outliers and able to identify clusters having non-spherical shapes and size variances<sup>3</sup>.](#) [It uses a set of well-defined representative points to handle clusters and outliers<sup>12</sup>.](#)

Here's a simplified pseudocode representation of the CURE algorithm:

1. Draw a random sample of points from the dataset.
2. Partition the sample into smaller subsets.
3. Partially cluster each subset into a fixed number of clusters using a hierarchical clustering algorithm.
4. For each cluster, choose a fixed number of representative points that are scattered throughout the cluster.
5. Shrink the representative points towards the centroid of the cluster by a fixed fraction.
6. Fully cluster the data by assigning each point to the closest representative point.

Now, let's break down the initialization and completion phases:

**Initialization Phase:** The initialization phase involves steps 1-3. [In this phase, a random sample of points is drawn from the dataset and partitioned into smaller subsets<sup>12</sup>.](#) [Each subset is then partially clustered into a fixed number of clusters using a hierarchical clustering algorithm<sup>12</sup>.](#) This phase sets up the initial clusters that will be refined in the next phase.

**Completion Phase:** The completion phase involves steps 4-6. In this phase, for each cluster, a fixed number of representative points are chosen that are scattered throughout the cluster<sup>12</sup>. These representative points are then shrunk towards the centroid of the cluster by a fixed fraction<sup>12</sup>. Finally, the data is fully clustered by assigning each point to the closest representative point<sup>12</sup>. This phase refines the initial clusters and produces the final clustering result.

Please note that this is a high-level description and the actual implementation may vary based on the specific requirements and the data at hand. Let me know if you need further clarification or help with anything else.

How amazon reviews can be used for sentiment analysis?  
Write the basic steps for it.

#### **Step 1: Get some sentiment examples**

As for every supervised learning problem, the algorithm needs to be trained from labeled examples in order to generalize to new data.

#### **Step 2: Extract features from examples**

Transform each example into a feature vector. The simplest way to do it is to have a vector where each dimension represents the frequency of a given word in the document.

#### **Step 3: Train the parameters**

This is where your model will learn from the data. There are multiple ways of using features to generate an output, but one of the simplest algorithms is logistic regression. Other well-known algorithms are Naive Bayes, SVM, Decision Trees and Neural Networks, but I'm going to use logistic regression as an example here.

In the simplest form, each feature will be associated with a weight. Let's say the word "love" has a weight equal to +4, "hate" is -10, "the" is 0 ... For a given example, the weights corresponding to the

features will be summed, and it will be considered "positive" if the total is  $> 0$ , "negative" otherwise. model will then try to find the optimal set of weights to maximize the number of examples in our data that are predicted correctly.

If you have more than 2 output classes, for example if you want to classify between "positive", "neutral" and "negative", each feature will have as many weights as there are classes, and the class with the highest weighted feature sum wins.

#### **Step 4: Test the model**

After we have trained the parameters to fit the training data, we have to make sure our model generalizes to new data, because it's really easy to overfit. The general way of regularizing the model is to prevent parameters from having extreme values.

**Write any 3 varieties in a social network graph.**

Sure, here are three types of social network graphs:

1. **Undirected Graphs:** In an undirected social network graph, the edges do not have a direction. This means that the relationship between two nodes is bidirectional. An example of this type of graph is a Facebook friendship network, where if person A is a friend of person B, then person B is also a friend of person A.
2. **Directed Graphs:** In a directed social network graph, the edges have a direction. This means that the relationship between two nodes is unidirectional. An example of this type of graph is a Twitter follower network, where if person A follows person B, it does not necessarily mean that person B follows person A.
3. **Weighted Graphs:** In a weighted social network graph, the edges have a weight associated with them. The weight can represent the strength or intensity of the relationship between two nodes. An example of this type of graph is a phone call network, where the weight of the edge between two people could represent the duration or frequency of calls between them.

**Write any 5 applications of data streams.**

Sure, here are five applications of data streams:

1. **Real-Time Analytics:** Data streams are often used in real-time analytics where immediate processing of data is required. This can include real-time monitoring of system logs, real-time user behavior tracking on websites, or real-time financial trading analysis.

2. **Internet of Things (IoT):** IoT devices generate continuous data streams. These data streams can be processed and analyzed to monitor device performance, detect anomalies, predict maintenance needs, and more.
3. **Social Media Analysis:** Social media platforms generate massive amounts of streaming data. Analyzing these data streams can provide insights into trending topics, sentiment analysis, user behavior, and more.
4. **Telecommunications:** In telecommunications, data streams are used for call detail record (CDR) analysis, network performance monitoring, fraud detection, and other applications.
5. **Healthcare:** In healthcare, data streams from medical devices can be analyzed in real-time to monitor patient health, detect anomalies, and alert healthcare providers to potential issues.

These are just a few examples. The applications of data streams are vast and span across many industries and domains. Let me know if you need more information on this or any other topic! 😊

Discuss how item based collaborative filtering can be applied for the example given.

User	Item	Rating
u1	m1	2
u1	m3	3
u2	m1	5
u2	m2	2
u3	m1	3
u3	m2	3
u3	m3	1
u4	m2	2
u4	m3	2

## Item-Based Collaborative Filtering

Item-based collaborative filtering recommends items to a user based on the similarity between items rather than user preferences. This approach involves three main steps:

1. **Calculate Similarities between Items:** Determine how similar each item is to every other item based on user ratings.
2. **Generate Predictions:** Predict the rating a user would give to an item based on the ratings they have given to similar items.
3. **Generate Recommendations:** Recommend items to the user based on the predicted ratings.

## Example and Solution

Given the user-item ratings matrix:

### Step 1: Calculate Similarities between Items

We'll use cosine similarity to calculate the similarities between items. The cosine similarity between two items  $i$  and  $j$  is defined as:

$$\text{similarity}(i, j) = \frac{\sum_{u \in U} r_{u,i} \cdot r_{u,j}}{\sqrt{\sum_{u \in U} r_{u,i}^2} \cdot \sqrt{\sum_{u \in U} r_{u,j}^2}}$$

Where:

- $U$  is the set of users who have rated both items  $i$  and  $j$ .
- $r_{u,i}$  is the rating given by user  $u$  to item  $i$ .

Let's calculate the cosine similarities for each pair of items.

### Step 2: Generate Predictions

To predict the rating  $p_{u,i}$  that user  $u$  would give to item  $i$ , we use the weighted sum of the user's ratings for similar items. The prediction formula is:

$$p_{u,i} = \frac{\sum_{j \in S(i)} \text{similarity}(i, j) \cdot r_{u,j}}{\sum_{j \in S(i)} |\text{similarity}(i, j)|}$$

Where  $S(i)$  is the set of items rated by user  $u$  that are similar to item  $i$ .

Let's go through the process with the given data.

## Similarity Calculations

### 1. m1 and m2:

- Common users: u2, u3
- Cosine similarity calculation:

$$\text{similarity}(m1, m2) = \frac{(5 \cdot 2) + (3 \cdot 3)}{\sqrt{5^2 + 3^2} \cdot \sqrt{2^2 + 3^2}} = \frac{10 + 9}{\sqrt{25 + 9} \cdot \sqrt{4 + 9}} = \frac{19}{\sqrt{34} \cdot \sqrt{13}}$$

### 2. m1 and m3:

- Common users: u1, u3
- Cosine similarity calculation:

$$\text{similarity}(m1, m3) = \frac{(2 \cdot 3) + (3 \cdot 1)}{\sqrt{2^2 + 3^2} \cdot \sqrt{3^2 + 1^2}} = \frac{6 + 3}{\sqrt{4 + 9} \cdot \sqrt{9 + 1}} = \frac{9}{\sqrt{13} \cdot \sqrt{10}}$$

### 3. m2 and m3:

- Common users: u3, u4
- Cosine similarity calculation:

$$\text{similarity}(m2, m3) = \frac{(3 \cdot 1) + (2 \cdot 2)}{\sqrt{3^2 + 2^2} \cdot \sqrt{1^2 + 2^2}} = \frac{3 + 4}{\sqrt{9 + 4} \cdot \sqrt{1 + 4}} = \frac{7}{\sqrt{13} \cdot \sqrt{5}}$$

## Predictions for u4 on m1 (since u4 has not rated m1)

Using the similarity matrix, we predict:

$$p_{u4,m1} = \frac{\text{similarity}(m1, m2) \cdot r_{u4, m2} + \text{similarity}(m1, m3) \cdot r_{u4, m3}}{|\text{similarity}(m1, m2)| + |\text{similarity}(m1, m3)|}$$

Finally, we compute this value and any other required predictions using the derived similarities.

Let's finalize the calculations with precise similarity values.

## Final Steps and Implementation

1. Compute the cosine similarities using Python.
2. Use the similarity values to compute predictions for missing ratings.
3. Recommend items to users based on the highest predicted ratings.

## Cosine Similarity Calculation

### Similarity between m1 and m2

Users u2 and u3 have rated both m1 and m2.

- Ratings:

- u2: m1 = 5, m2 = 2
- u3: m1 = 3, m2 = 3

Cosine similarity formula:

$$\text{similarity}(m1, m2) = \frac{(5 \times 2) + (3 \times 3)}{\sqrt{(5^2 + 3^2) \times \sqrt{(2^2 + 3^2)}}} = \frac{10 + 9}{\sqrt{25 + 9} \times \sqrt{4 + 9}} = \frac{19}{\sqrt{34} \times \sqrt{13}} = \frac{19}{\sqrt{442}} \approx \frac{19}{21.02} \approx 0.904$$

### Similarity between m1 and m3

Users u1 and u3 have rated both m1 and m3.

- Ratings:

- u1: m1 = 2, m3 = 3
- u3: m1 = 3, m3 = 1

Cosine similarity formula:

$$\text{similarity}(m1, m3) = \frac{(2 \times 3) + (3 \times 1)}{\sqrt{(2^2 + 3^2) \times \sqrt{(3^2 + 1^2)}}} = \frac{6 + 3}{\sqrt{4 + 9} \times \sqrt{9 + 1}} = \frac{9}{\sqrt{13} \times \sqrt{10}} = \frac{9}{\sqrt{130}} \approx \frac{9}{11.40} \approx 0.789$$

### Similarity between m2 and m3

Users u3 and u4 have rated both m2 and m3.

- Ratings:

- u3: m2 = 3, m3 = 1
- u4: m2 = 2, m3 = 2

Cosine similarity formula:

$$\text{similarity}(m2, m3) = \frac{(3 \times 1) + (2 \times 2)}{\sqrt{(3^2 + 2^2) \times \sqrt{(1^2 + 2^2)}}} = \frac{3+4}{\sqrt{9+4} \times \sqrt{1+4}} = \frac{7}{\sqrt{13} \times \sqrt{5}} = \frac{7}{\sqrt{65}} \approx \frac{7}{8.06} \approx 0.868$$

## Predicting the Missing Rating for u4 on m1

Using the similarities computed above, we predict the rating for u4 on m1:

$$p_{u4,m1} = \frac{\text{similarity}(m1, m2) \cdot r_{u4,m2} + \text{similarity}(m1, m3) \cdot r_{u4,m3}}{|\text{similarity}(m1, m2)| + |\text{similarity}(m1, m3)|}$$

$$p_{u4,m1} = \frac{0.904 \cdot 2 + 0.789 \cdot 2}{|0.904| + |0.789|}$$

$$p_{u4,m1} = \frac{1.808 + 1.578}{0.904 + 0.789}$$



$$p_{u4,m1} = \frac{3.386}{1.693}$$

$$p_{u4,m1} = \frac{\text{similarity}(m1, m2) \cdot r_{u4,m2} + \text{similarity}(m1, m3) \cdot r_{u4,m3}}{|\text{similarity}(m1, m2)| + |\text{similarity}(m1, m3)|}$$

$$p_{u4,m1} = \frac{0.904 \cdot 2 + 0.789 \cdot 2}{|0.904| + |0.789|}$$

$$p_{u4,m1} = \frac{1.808 + 1.578}{0.904 + 0.789}$$

$$p_{u4,m1} = \frac{3.386}{1.693}$$

$$p_{u4,m1} \approx 2.0$$

## Recommendations

Based on the predicted rating of approximately 2.0 for m1 for user u4, we can recommend m1 to user u4.

This example demonstrates how item-based collaborative filtering can be applied to predict ratings and generate recommendations. [→]

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

# ESE NOVEMBER 2018

Explain different characteristics of Big data with example

Explain the difference between NameNode, Backup Node and Checkpoint node.

Sure, here's a comparison of NameNode, Backup Node, and Checkpoint Node in Hadoop:

Feature	NameNode	Backup Node	Checkpoint Node
<b>Primary Role</b>	<a href="#">Manages the metadata of the HDFS file system<sup>1</sup></a>	<a href="#">Provides check pointing functionality and maintains an up-to-date in-memory copy of the file system namespace<sup>12</sup></a>	<a href="#">Creates checkpoints for the namespace at regular intervals<sup>13</sup></a>
<b>Metadata Storage</b>	<a href="#">Stores the directory tree of all files present in the HDFS file system<sup>1</sup></a>	<a href="#">Maintains an up-to-date in-memory copy of the file system namespace<sup>12</sup></a>	<a href="#">Keeps track of the latest checkpoint in a directory<sup>13</sup></a>

---

<b>File System Image</b>	<a href="#">Uses fsimage file to keep track of the latest checkpoint of the namespace<sup>1</sup></a>	<a href="#">Holds the current state in-memory and just needs to save this to an image file to create a new checkpoint<sup>2</sup></a>	<a href="#">Downloads the edits and fsimage file from the NameNode and merges it locally<sup>13</sup></a>
<b>Edit Logs</b>	<a href="#">Uses edits file to log changes that have been made to the namespace since the last checkpoint<sup>1</sup></a>	<a href="#">Receives a stream of file system edits from the NameNode<sup>42</sup></a>	<a href="#">Downloads the edits file from the NameNode for merging<sup>13</sup></a>
<b>Checkpoint Creation</b>	Does not create checkpoints	<a href="#">Creates new checkpoints by saving the current in-memory state to an image file<sup>2</sup></a>	<a href="#">Creates checkpoints at regular intervals by merging the downloaded edits and fsimage file<sup>13</sup></a>
<b>Real-Time Updates</b>	Receives real-time updates of the file system metadata	<a href="#">Receives real-time updates of the file system metadata<sup>4</sup></a>	<a href="#">Does not receive real-time updates of the file system metadata<sup>3</sup></a>

---

<b>Namespace Synchronization</b>	<a href="#">Maintains the primary copy of the file system namespace<sup>1</sup></a>	<a href="#">Maintains a copy of the file system namespace that is in sync with the NameNode<sup>12</sup></a>	<a href="#">Maintains a copy of the file system namespace that is periodically synchronized with the NameNode<sup>13</sup></a>
<b>Failure Recovery</b>	<a href="#">Single point of failure in HDFS<sup>4</sup></a>	<a href="#">Cannot replace the NameNode in case of failure<sup>4</sup></a>	<a href="#">Cannot replace the NameNode in case of failure<sup>4</sup></a>
<b>Role in Restart</b>	<a href="#">Uses the fsimage from the Checkpoint Node for restart<sup>4</sup></a>	Does not play a role in NameNode restart	<a href="#">Provides the fsimage for NameNode restart<sup>4</sup></a>
<b>Data Storage</b>	<a href="#">Does not store actual HDFS data<sup>1</sup></a>	<a href="#">Does not store actual HDFS data<sup>12</sup></a>	<a href="#">Does not store actual HDFS data<sup>13</sup></a>

---

Please note that this is a high-level comparison and the actual functionalities may vary based on the specific Hadoop configuration and version. Let me know if you need more information on this or any other topic! 😊

Write a MapReduce program to count occurrence of word "Environment"(case sensitive) in the large text whose sample is as below. Clearly write Map class and Reduce class :

World Environment Day is celebrated in many ways. Street rallies, parades, street plays create awareness about world environment. World Environment Day is a day for us to inspect the state of our environment. It calls for us to stop in our tracks one day in the year and examine our surroundings. It asks us to pledge, in a small way at least, to do something for the environment. There is something each of us can do to preserve the environment. So let us all pledge to do something, at least one thing, before the next World Environment Day

What is Rack Awareness? Explain with an example? List advantages of Rack Awareness?

Rack Awareness is a concept in Hadoop's HDFS (Hadoop Distributed File System) that optimizes data storage and retrieval by considering the physical location of nodes in a Hadoop cluster<sup>12345</sup>.

What is Rack Awareness? In a large Hadoop cluster, there are multiple racks, each containing several DataNodes<sup>12345</sup>. Communication between DataNodes on the same rack is more efficient than between DataNodes on different racks<sup>12345</sup>. To reduce network traffic during file read/write operations, the NameNode chooses the closest DataNode to serve the client's request<sup>12345</sup>. The NameNode maintains the rack IDs of each DataNode to achieve this<sup>12345</sup>. This concept of choosing the closest DataNode based on rack information is known as Rack Awareness<sup>12345</sup>.

Example of Rack Awareness: Let's consider a Hadoop cluster with three racks, each containing four DataNodes<sup>2</sup>. Suppose you have three file blocks (Block 1, Block 2, Block 3) that you want to store in these DataNodes<sup>2</sup>. Hadoop's default replication factor is 3, meaning it creates three replicas of each block<sup>2</sup>. Hadoop places these replicas across the racks following certain policies<sup>2</sup>:

- There should not be more than one replica on the same DataNode<sup>2</sup>.
- More than two replicas of a single block are not allowed on the same rack<sup>2</sup>.
- The number of racks used inside a Hadoop cluster must be smaller than the number of replicas<sup>2</sup>.

Following these policies, Hadoop might distribute the replicas of Block 1 across the first DataNode of Rack 1 and the fifth and sixth DataNodes of Rack 2<sup>2</sup>. Similarly, it distributes the replicas of Blocks 2 and 3 across different racks<sup>2</sup>.

#### **Advantages of Rack Awareness:**

1. **Reduced Network Traffic:** By storing data on the closest DataNode, Rack Awareness reduces network traffic during file read/write operations, improving cluster performance<sup>12345</sup>.
2. **Fault Tolerance:** Rack Awareness ensures that not all replicas of a block are stored on the same rack. This means that even if a rack fails, the data is still available on another rack<sup>12345</sup>.
3. **High Data Availability:** By creating replicas of data blocks and distributing them across different racks, Rack Awareness ensures high data availability<sup>12345</sup>.
4. **Improved Read Performance:** When a client needs to read a file, the NameNode returns the addresses of the DataNodes containing the replicas. The client reads the data from the closest DataNode, improving read performance<sup>12345</sup>.
5. **Reduced Latency:** By choosing the closest DataNode for read/write operations, Rack Awareness reduces latency<sup>12345</sup>.

I hope this provides a comprehensive understanding of Rack Awareness in Hadoop. Let me know if you need more information on this or any other topic! 😊

## Explain Natural join operation of RDBMS by map reduce with an example? Write its pseudo code?

Sure, let's start with a brief explanation of the Natural Join operation in RDBMS and then see how it can be implemented using MapReduce.

**Natural Join in RDBMS:** A Natural Join is a type of inner join where the join is made on columns that have the same name in both tables<sup>1</sup>. For example, if we have two tables Orders and Customers with a common column CustomerID, a Natural Join on these two tables would return all rows where CustomerID matches in both tables<sup>1</sup>.

**Natural Join Using MapReduce:** In the context of MapReduce, a Natural Join operation can be implemented using a combination of map and reduce functions. The map function emits pairs with the join key as the key and the record as the value. The reduce function then groups all records with the same join key and performs the join operation<sup>23</sup>.

Here's a high-level pseudocode representation of a Natural Join operation using MapReduce:

Mapper:

```
Input: datasets A and B
for each record r in A:
 emit(key = r.joinKey, value = ("A", r))
```

```
for each record r in B:
 emit(key = r.joinKey, value = ("B", r))
```

Reducer:

```
Input: key = joinKey, values = list of ("A", record) and/or ("B", record)
A_list = []
B_list = []
for each v in values:
 if v is from A:
 add v to A_list
 else if v is from B:
 add v to B_list
if A_list is not empty and B_list is not empty:
 for each v in A_list:
 for each w in B_list:
 emit(v, w)
```

In this pseudocode, **A** and **B** represent two datasets that you want to join. The **Mapper** emits a key-value pair for each record in the datasets, with the join key as the key and a tuple containing the dataset identifier ("A" or "B") and the record as the value. The **Reducer** then performs the join operation for each join key, emitting a key-value pair for each joined record. Note that this pseudocode assumes that you want to perform a Natural Join. If you want to perform a different type of join (e.g., left outer join, right outer join, full outer join), you would need to modify the **Reducer** logic accordingly.

I hope this helps! Let me know if you need further clarification or help with anything else.

What are the drawbacks of traditional clustering algorithm?

How CURE clustering algorithm overcomes the problems of traditional clustering?

Traditional clustering algorithms, such as k-means, have several drawbacks:

1. **Choosing the number of clusters (k) manually**: The optimal number of clusters is not always known in advance and can be difficult to determine<sup>1</sup>.
2. **Dependence on initial values**: The final clustering result can vary based on the initial choice of centroids<sup>1</sup>.
3. **Difficulty with varying sizes and densities**: Traditional clustering algorithms often struggle with data where clusters are of varying sizes and densities<sup>1</sup>.
4. **Sensitivity to outliers**: Outliers can significantly affect the positioning of centroids, leading to less accurate clusters<sup>12</sup>.
5. **Difficulty with non-spherical data**: Traditional clustering algorithms often struggle with non-convex or non-spherical data<sup>3</sup>.

6. **Scaling with number of dimensions:** As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given examples<sup>1</sup>.

The CURE (Clustering Using REpresentatives) algorithm addresses many of these issues:

1. **Handling of non-spherical shapes and size variances:** CURE identifies clusters that are non-spherical and have wide variances in size<sup>45</sup>.
2. **Robustness to outliers:** CURE is more robust to outliers compared to traditional clustering algorithms<sup>45</sup>.
3. **Efficiency with large databases:** CURE can handle large databases by combining random sampling and partitioning methods<sup>4</sup>.
4. **Use of representative points:** Instead of using a single point (centroid), CURE uses a set of well-scattered representative points. This allows for more efficient handling of clusters and better elimination of outliers<sup>56</sup>.

In summary, the CURE algorithm overcomes many of the limitations of traditional clustering algorithms, making it a powerful tool for data analysis and machine learning<sup>56478</sup>.

What are the applications of Stream computing? Explain any two applications in detail with example?

Stream computing is a technique for processing continuous data streams in real-time<sup>1</sup>. It enables organizations to rapidly analyze and respond to data, making it particularly valuable for various applications<sup>1</sup>. Here are two detailed examples:

1. **Real-Time Analytics:** Real-time analytics is a key application of stream computing<sup>1</sup>. For instance, a company might use stream computing to monitor user behavior on its website in real-time. As users interact with the website, their actions generate a continuous stream of data. This data can be processed in real-time to understand user behavior, identify trends, and make immediate decisions. For example, if the company notices that many users are abandoning their shopping carts at the checkout page, they might decide to offer a discount or streamline the checkout process to improve conversion rates<sup>1</sup>.
2. **Fraud Detection:** Stream computing is also used in fraud detection<sup>12</sup>. For example, a credit card company might use stream computing to monitor transactions in real-time. Each transaction generates a data point in a continuous stream of data. The company can analyze this data in real-time to identify unusual patterns that might indicate fraudulent activity. If a suspicious transaction is detected, the company can immediately alert the cardholder or block the transaction, preventing potential fraud<sup>12</sup>.

Other applications of stream computing include monitoring and reporting on internal IT systems, location data processing, real-time stock trades, marketing, sales, and business analytics, customer/user activity monitoring, and more<sup>12</sup>. Stream computing is particularly effective for scenarios that require minimal latency, built-in features for dealing with imperfect data, SQL queries on data streams to build extensive operators, and a guaranteed ability to

generate predictable and consistent results<sup>1</sup>. Let me know if you need more information on this or any other topic! 😊

## What is sentiment analysis? Explain sentiments analysis process in detail by considering YouTube user feedback?

Sentiment analysis, also known as opinion mining, is a subfield of Natural Language Processing (NLP) that involves determining and extracting subjective information from source materials<sup>1</sup>. It's used to understand the sentiments, opinions, and emotions expressed in a piece of text<sup>1</sup>.

When it comes to YouTube user feedback, sentiment analysis can be particularly useful. It can help content creators understand their audience's reactions and emotional engagement with their videos<sup>12</sup>. Here's a detailed process of how sentiment analysis can be performed on YouTube comments:

1. **Generating API Key:** The first step is to generate an API key for the YouTube API. This key allows you to fetch comments from YouTube videos<sup>12</sup>.
2. **Fetching Comments:** Using the YouTube API, comments from a specific YouTube video are fetched<sup>12</sup>.
3. **Preprocessing:** The fetched comments are then preprocessed. This step involves cleaning the data by removing irrelevant information, correcting spelling mistakes, and converting all text to lowercase<sup>12</sup>.
4. **Sentiment Analysis:** After preprocessing, sentiment analysis is performed on the comments. This can be done using various NLP techniques and libraries<sup>12</sup>.
5. **Interpretation:** The sentiments are then interpreted. Each comment is categorized as positive, negative, or neutral based on the sentiment score<sup>12</sup>.
6. **Visualization:** Finally, the results are visualized. This could be in the form of graphs or charts showing the distribution of positive, negative, and neutral comments<sup>12</sup>.

This process provides valuable insights into the audience's perception of the video content, allowing content creators to make necessary improvements<sup>12</sup>. It's important to note that the accuracy of sentiment analysis can depend on the complexity of the language used in the comments, the effectiveness of the preprocessing steps, and the capability of the sentiment analysis model<sup>12</sup>.

## What is text mining? Explain difference between text mining and text analytics? Explain various applications of text analytics?

Text Mining is a component of data mining that deals specifically with unstructured text data<sup>1</sup>. It involves the use of natural language processing (NLP) techniques to extract useful

information and insights from large amounts of unstructured text data<sup>1</sup>. Text mining can be used as a preprocessing step for data mining or as a standalone process for specific tasks<sup>1</sup>. It is mostly used for transforming unstructured text data into structured data that can be used for data mining tasks such as classification, clustering, and association rule mining<sup>1</sup>.

On the other hand, Text Analytics is a process of analyzing and understanding written or spoken language<sup>12</sup>. It employs computer algorithms and techniques to extract valuable information, patterns, and insights from extensive textual data<sup>12</sup>. In simpler terms, text analytics helps computers understand and interpret human language<sup>12</sup>.

While both text mining and text analytics are used to analyze raw textual data, they use different techniques and provide different types of results<sup>13</sup>. Text mining involves the application of natural language processing and machine learning techniques to discover patterns, trends, and knowledge from large volumes of unstructured text<sup>1</sup>. However, Text Analytics focuses on extracting meaningful information, sentiments, and context from text, often using statistical and linguistic methods<sup>1</sup>. While text mining emphasizes uncovering hidden patterns, text analytics emphasizes deriving actionable insights for decision-making<sup>1</sup>.

There are numerous applications of Text Analytics, including but not limited to:

1. **Sentiment Analysis:** Text analytics can be used to determine the sentiment expressed in text data, such as customer reviews or social media posts. This can help businesses understand how their customers feel about their products or services<sup>4</sup>.
2. **Information Extraction:** Text analytics can extract specific pieces of information from unstructured text data, such as names, dates, or locations. This can be useful in many fields, including journalism, law enforcement, and market research<sup>4</sup>.
3. **Topic Modeling:** Text analytics can identify the main topics discussed in a collection of documents. This can be useful for understanding the main themes in customer feedback, news articles, or academic papers<sup>4</sup>.
4. **Trend Analysis:** Text analytics can identify trends over time in the sentiment or topics discussed in text data. This can help businesses anticipate changes in customer preferences or market conditions<sup>4</sup>.
5. **Spam Filtering:** Text analytics can identify unwanted or harmful content, such as spam emails or inappropriate comments on a website<sup>4</sup>.

These are just a few examples. The applications of text analytics are vast and span across many industries and domains<sup>4</sup>. Let me know if you need more information on this or any other topic! 😊

Describe oozie workflows? Explain some of the workflow elements?

What are different key problems in recommendation? Explain Collaborative filtering based recommendation?

Suppose there are 6 users and 5 Movies with 5 star rating is given in a table below, calculate User profile for user 2 and user 5

Keywords	Harry Potter 1	Harry Potter 2	Harry Potter 3	Harry Potter 4	Harry Potter 5
Users					
User 1	5	4	3	2	-
User 2	2	4	5	-	-
User 3	3	4	2	4	3
User 4	-	1	-	3	2
User 5	2	5	3	-	-
User 6	2	-	-	3	2

### 1. Key Problems in Recommendation

Some key problems in recommendation systems include:

1. Cold Start Problem: This occurs when there is insufficient data on new users or new items, making it difficult to provide accurate recommendations.
2. Scalability: As the number of users and items grows, the recommendation system needs to handle increased data, which can be computationally intensive.
3. Sparsity: In large datasets, users typically rate only a small subset of items, leading to a sparse matrix. This sparsity makes it challenging to find similar users or items.
4. Diversity vs. Accuracy: Ensuring recommendations are diverse (to introduce users to new items) while maintaining high accuracy is a delicate balance.
5. Privacy Concerns: Handling user data responsibly and ensuring privacy can be challenging, especially with personalized recommendations.

### 2. Collaborative Filtering Based Recommendation

Collaborative filtering (CF) is a popular technique used in recommendation systems to predict the preferences of a user by collecting preferences from many users. It can be divided into two categories:

- User-based Collaborative Filtering: This approach recommends items by finding similar users based on their ratings. It assumes that if user A is similar to user B, then the items liked by B will also be liked by A.
- Item-based Collaborative Filtering: This approach recommends items by finding similar items based on users' ratings. It assumes that if item X is similar to item Y, then users who liked X will also like Y.

### 3. Calculate User Profile for User 2 and User 5

To calculate the user profiles for User 2 and User 5, we can average their ratings across the movies they have rated. However, since not all users have rated all movies, we typically handle missing values by ignoring them in the average calculation.

## User 2 Profile

Ratings for User 2:

- Harry Potter 1: 2
- Harry Potter 2: 4
- Harry Potter 3: 5
- Harry Potter 4: (no rating)
- Harry Potter 5: (no rating)

Average rating for User 2 =  $(2 + 4 + 5) / 3 = 11 / 3 \approx 3.67$

## User 5 Profile

Ratings for User 5:

- Harry Potter 1: 2
- Harry Potter 2: 5
- Harry Potter 3: 3
- Harry Potter 4: (no rating)
- Harry Potter 5: (no rating)

Average rating for User 5 =  $(2 + 5 + 3) / 3 = 10 / 3 \approx 3.33$

So, the user profiles are:

- User 2: 3.67
- User 5: 3.33

These profiles can be used in collaborative filtering algorithms to find similar users or items, ultimately helping in making recommendations.

Describe how to start and configure a flume agent in detail?

<https://datascientest.com/en/understanding-apache-flume-its-purpose-and-applications>

Describe in detail how to submit a workflow job and a coordinator job in oozie workflows?

# ESE April May 2018

Explain 4 ways by which big data problems are handled by NoSQL  
A client needs a database design for his blog. Website has following requirements.

- a) Every post can have one or more tag.
- b) Every post has unique title, description and url
- c) Every post has name of its publisher and total no. of likes.
- d) On each post, there can be 0 or more comments.

To handle big data problems, NoSQL databases employ various strategies. Here are four ways NoSQL databases address the challenges of big data:

1. Horizontal Scaling: NoSQL databases are designed to scale horizontally by adding more nodes or servers to the cluster. This allows for distributing data across multiple servers, providing better performance and scalability compared to traditional relational databases that primarily scale vertically (by adding more resources to a single server).
2. Flexible Data Models: NoSQL databases typically have flexible data models, such as key-value, document, column-family, or graph. This flexibility allows for storing and retrieving data without the need for a predefined schema. It enables handling unstructured or semi-structured data more efficiently, which is common in big data scenarios.
3. Sharding and Replication: NoSQL databases employ techniques like sharding (partitioning data across multiple servers) and replication (maintaining multiple copies of data across servers) to distribute the data and workload. This helps in improving availability, scalability, and fault tolerance, which are essential for handling large volumes of data and high read/write workloads.
4. Eventual Consistency: Many NoSQL databases favor eventual consistency over strict consistency to achieve better performance and scalability. Eventual consistency means that data updates are propagated across the cluster eventually, rather than immediately enforcing strict consistency at the time of the update. This trade-off allows for higher throughput and availability, which is often more important in big data scenarios than strict consistency.

Regarding the blog website requirements, a document-oriented NoSQL database like MongoDB could be a suitable choice. Here's how the data model could be designed:

1. Posts Collection:

- `title` (string, unique)
- `description` (string)
- `url` (string, unique)
- `publisher` (string)
- `likes` (integer)
- `tags` (array of strings)
- `comments` (array of comment document references)

2. Comments Collection:

- `postId` (reference to the post document)
- `author` (string)
- `text` (string)
- `timestamp` (date)

In this design, the "Posts" collection would store the post details, including the title, description, URL, publisher, likes, and an array of tags. Each post document would also have an array of references to comment documents stored in the "Comments" collection.

The "Comments" collection would store the comment details, including the author, text, and timestamp. Each comment document would have a reference to the corresponding post document using the `postId` field.

This data model leverages the flexible document structure of MongoDB, allowing for storing the post details, tags, and comments together while maintaining proper relationships. The use of references (instead of embedding comments within posts) helps in efficient querying and updating of comments separately from posts.

Design the normalized schema(tables and columns) for RDBMS and the MongoDB schema.

How does clickstream analytics using Hadoop help sales and marketing team to prepare their campaigning strategies? How does Pagerank differ from HITS algorithm? Give 2 differences. Calculate the authority and hub scores for the given matrix that represent a graph of four vertices (n1 to n4 ), using HITS algorithm with k=2. Identify the best authority and hub nodes.

Clickstream analytics using Hadoop can provide valuable insights to sales and marketing teams, enabling them to prepare more effective campaigning strategies. Here's how:

1. Understanding User Behavior: Clickstream analytics helps in understanding how users interact with a website or digital platform. By analyzing clickstream data stored in Hadoop, sales and marketing teams can gain insights into user behavior, such as which pages they visit, how long they stay on each page, the sequence of actions they perform, and where they drop off. Understanding user behavior helps in identifying patterns and preferences, which can inform campaign strategies.
2. Segmentation and Targeting: Clickstream data can be used to segment users based on their behavior, demographics, preferences, and interests. Hadoop's distributed computing capabilities allow for processing large volumes of clickstream data to create meaningful user segments. Sales and marketing teams can then target specific user segments with tailored campaigns, messages, and offers that are more likely to resonate with them.
3. Optimizing Customer Journey: Analyzing clickstream data helps in optimizing the customer journey by identifying bottlenecks, points of friction, and areas for improvement. By understanding how users navigate through the website or app, sales and marketing teams can optimize the user experience, streamline conversion paths, and reduce bounce rates. This optimization leads to a smoother and more effective customer journey, ultimately increasing conversion rates and sales.
4. Personalization and Recommendation: Clickstream analytics enables personalized marketing and product recommendations based on users' browsing history, preferences, and past interactions. By leveraging Hadoop's capabilities to process and analyze large datasets, sales and marketing teams can implement personalized recommendations engines that deliver relevant content, products, and offers to individual users in real-time, increasing engagement and conversion rates.
5. Campaign Performance Measurement: Hadoop-based clickstream analytics allows sales and marketing teams to measure the performance of their campaigns accurately. By tracking key metrics such as click-through rates, conversion rates, customer acquisition costs, and return on investment (ROI), teams can assess the effectiveness of their campaigns and make data-driven decisions to optimize future campaigns. They can identify which campaigns, channels, and messaging resonate most with their target audience and allocate resources accordingly.

Overall, clickstream analytics using Hadoop empowers sales and marketing teams with actionable insights derived from large volumes of data, enabling them to optimize campaigning strategies, improve customer engagement, and drive sales growth effectively.

0	1	1	1
0	0	1	1
1	0	0	1
0	0	0	1

What are the categories the spammer divides the web pages into. Explain? Mathematically prove the improvement in pagerank of a target page using spam farm.

$$PR(P) = \frac{1-d}{N} + d \sum_{i=1}^n \frac{PR(P_i)}{L(P_i)}$$

where:

- $PR(P)$  is the PageRank of page  $P$ .
- $d$  is the damping factor (typically 0.85).
- $N$  is the total number of pages.
- $P_i$  are the pages linking to  $P$ .
- $L(P_i)$  is the number of outbound links on page  $P_i$ .

#### Spam Farm Structure

Assume a spammer creates a network of pages to boost the PageRank of a target page  $T$ . The spam farm includes:

- $m$  supporting pages  $S_1, S_2, \dots, S_m$ .
- Each supporting page links to the target page  $T$ .
- Supporting pages may also link to each other or other pages within the farm to further distribute PageRank.

#### PageRank Improvement Calculation

Let's denote:

- $PR(T)$  as the PageRank of the target page  $T$ .
- $PR(S_i)$  as the PageRank of a supporting page  $S_i$ .

For simplicity, assume each supporting page only links to the target page  $T$  and possibly other supporting pages.

$$PR(T) = \frac{1-d}{N} + d \left( \sum_{i=1}^m \frac{PR(S_i)}{L(S_i)} \right)$$

Initially, without the spam farm, assume  $T$  has links from  $k$  legitimate pages. The initial PageRank is:

$$PR(T) = \frac{1-d}{N} + d \left( \sum_{j=1}^k \frac{PR(L_j)}{L(L_j)} \right)$$

After adding the spam farm:

$$PR(T) = \frac{1-d}{N} + d \left( \sum_{j=1}^k \frac{PR(L_j)}{L(L_j)} + \sum_{i=1}^m \frac{PR(S_i)}{L(S_i)} \right)$$

The PageRank of each supporting page  $S_i$  is influenced by the structure of the farm. If we assume the spam farm is a closed network where each supporting page only links to the target page and each other:

$$PR(S_i) = \frac{1-d}{N} + d \left( \sum_{S_j \in Farm} \frac{PR(S_j)}{m} \right)$$

Since each supporting page distributes its PageRank to  $m$  pages (including the target):

$$PR(S_i) = \frac{1-d}{N} + d \left( \frac{\sum_{j=1}^m PR(S_j)}{m} \right)$$

Assuming a uniform distribution within the farm:

$$PR(S_i) \approx \frac{1-d}{N} + d \left( \frac{m \cdot PR(S_i)}{m} \right)$$

$$PR(S_i) \approx \frac{1-d}{N} + d \cdot PR(S_i)$$

Solving for  $PR(S_i)$ :

$$PR(S_i)(1 - d) = \frac{1-d}{N}$$

$$PR(S_i) = \frac{1}{N}$$

Substituting back into the PageRank of  $T$ :

$$PR(T) = \frac{1-d}{N} + d \left( \sum_{j=1}^k \frac{PR(L_j)}{L(L_j)} + \sum_{i=1}^m \frac{1/N}{1} \right)$$

$$PR(T) = \frac{1-d}{N} + d \left( \sum_{j=1}^k \frac{PR(L_j)}{L(L_j)} + \frac{m}{N} \right) \quad \downarrow$$

Thus, the addition of the spam farm increases the term  $\sum_{i=1}^m \frac{1}{N}$ , directly contributing to the improvement of  $PR(T)$ .

In conclusion, the structured addition of a spam farm increases the PageRank of the target page  $T$  by adding artificial links that boost its ranking due to the cumulative effect of PageRank distribution from the supporting pages.

Write a Map Reduce program to find out Designation Wise Customer \\

count like How many Doctors are there in the file or How many Lawyers \\

are in the file.

Consider following scenario for taking input data :

Let there be a file : custs

where col1 - cust\_id

col2 - first\_name

col3 - last\_name

col4 - age

col5 - designation

Write a Map Reduce program to find the total number of words from the

file as given below:

Let the input data is:

Apache Spark is a fast, in-memory data processing engine

with elegant and expressive development APIs to allow.

final output will be = total words, 17

How to compute RDBMS Selection Operation by Map Reduce.  
Clearly

explain Map and Reduce functions and Give its pseudo code.

Consider the following two texts, Find the similarity between 1 and 2.

1. Julie loves me more than Linda loves me

2. Jane likes me more than Julie loves me

### Cosine Similarity

Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. It is calculated as follows:

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

where:

- $\mathbf{A}$  and  $\mathbf{B}$  are the term frequency vectors of the two texts.
- $\mathbf{A} \cdot \mathbf{B}$  is the dot product of the vectors.
- $\|\mathbf{A}\|$  and  $\|\mathbf{B}\|$  are the magnitudes of the vectors.

### Steps to Compute Cosine Similarity

1. **Tokenization:** Split the texts into individual words.
2. **Stop Words Removal:** Remove common stop words (if necessary).
3. **Term Frequency (TF):** Count the frequency of each word in both texts.
4. **Vector Representation:** Represent each text as a vector of term frequencies.
5. **Cosine Similarity Calculation:** Compute the cosine similarity between the two vectors.

## Example Calculation

### 1. Tokenization:

- Text 1: "Julie loves me more than Linda loves me"
  - Tokens: *Julie, loves, me, more, than, Linda, loves, me*
- Text 2: "Jane likes me more than Julie loves me"
  - Tokens: *Jane, likes, me, more, than, Julie, loves, me*

### 2. Term Frequency (TF):

- Combine the unique words from both texts:  
*Julie, loves, me, more, than, Linda, Jane, likes*
- Create term frequency vectors:
  - Text 1: 1, 2, 2, 1, 1, 1, 0, 0
  - Text 2: 1, 1, 2, 1, 1, 0, 1, 1

### 3. Vector Representation:

- Text 1 vector:  $\mathbf{A} = [1, 2, 2, 1, 1, 1, 0, 0]$
- Text 2 vector:  $\mathbf{B} = [1, 1, 2, 1, 1, 0, 1, 1]$



#### 4. Cosine Similarity Calculation:

- Dot product  $\mathbf{A} \cdot \mathbf{B}$ :

$$(1 \cdot 1) + (2 \cdot 1) + (2 \cdot 2) + (1 \cdot 1) + (1 \cdot 1) + (1 \cdot 0) + (0 \cdot 1) + (0 \cdot 1) = 1 + 2 + 4 + 1 + 1 + 0 + 0 + 0 = 9$$

- Magnitude of  $\mathbf{A}$ :

$$\|\mathbf{A}\| = \sqrt{1^2 + 2^2 + 2^2 + 1^2 + 1^2 + 1^2 + 0^2 + 0^2} = \sqrt{1 + 4 + 4 + 1 + 1 + 1 + 0 + 0} = \sqrt{14}$$

- Magnitude of  $\mathbf{B}$ :

$$\|\mathbf{B}\| = \sqrt{1^2 + 1^2 + 2^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2} = \sqrt{1 + 1 + 4 + 1 + 1 + 0 + 1 + 1} = \sqrt{14}$$

- Cosine Similarity:

$$\text{Cosine Similarity} = \frac{9}{3.464 \times 3.162} \approx \frac{9}{10.957} \approx 0.821$$

#### Result

The cosine similarity between the two texts is approximately 0.821. This value ranges between -1 and 1, where 1 indicates that the texts are identical, 0 indicates no similarity, and -1 indicates completely opposite texts. Hence, a cosine similarity of 0.821 indicates a high degree of similarity between the two texts.

What is Data Stream Management System? Explain with block diagram.

<https://www.geeksforgeeks.org/data-stream-management-system-architecture/>

Explain with diagram how the PCY algorithm helps to perform frequent itemset mining for large datasets

### PCY Algorithm for Frequent Itemset Mining

The PCY (Park-Chen-Yu) algorithm is an efficient algorithm for finding frequent itemsets in large datasets. It is an extension of the Apriori algorithm that uses a hash table to reduce the number of candidate pairs and thus improve the performance of frequent itemset mining.

### Steps of the PCY Algorithm

1. **Pass 1: Counting Single Items and Hashing Pairs**
2. **Pass 2: Counting Candidate Pairs**
3. **Generating Frequent Itemsets**

### Detailed Explanation with Diagram

#### Pass 1: Counting Single Items and Hashing Pairs

In the first pass, the algorithm performs two tasks simultaneously:

1. **Count the occurrences of individual items** to determine which items are frequent.
2. **Hash pairs of items** to a hash table to estimate the frequency of item pairs.

## Diagram for Pass 1:

mathematica

```
Transaction 1: {A, B, C}
Transaction 2: {A, C, D}
Transaction 3: {B, C, E}
Transaction 4: {A, B, D, E}
Transaction 5: {B, C, D}
```

Step 1: Count single items

```
A: 3
B: 4
C: 4
D: 3
E: 2
```

Step 2: Hash pairs

```
Hash(A, B) -> Bucket 1
Hash(A, C) -> Bucket 2
Hash(B, C) -> Bucket 3
Hash(A, D) -> Bucket 4
```

...

(Each bucket stores the count of pairs that hash to it)

## Pass 2: Counting Candidate Pairs

In the second pass, the algorithm uses the frequent items from the first pass to generate candidate pairs. It then counts the occurrences of these candidate pairs only if their hash value corresponds to a frequent bucket in the hash table from the first pass.

### Diagram for Pass 2:

mathematica

 Copy code

```
Frequent Items: {A, B, C, D}
```

```
Generate candidate pairs:
```

```
(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)
```

```
Count candidate pairs if their hash value was frequent:
```

```
(A, B): 2
```

```
(A, C): 2
```

```
(A, D): 2
```

```
(B, C): 3
```

```
(B, D): 2
```

```
(C, D): 2
```

## Generating Frequent Itemsets

Finally, the algorithm generates the frequent itemsets from the candidate pairs that meet the minimum support threshold.

### Example Diagram:

css

 Cop

```
Minimum Support Threshold = 3
```

```
Frequent Itemsets:
```

```
{B, C}
```

```
(Final frequent pairs are determined based on actual counts)
```

## Visualization Summary

Let's summarize with a clear diagram for each pass:

### Pass 1:

```
mathematica
```

**Transactions:**

```
1: {A, B, C}
2: {A, C, D}
3: {B, C, E}
4: {A, B, D, E}
5: {B, C, D}
```

**Count single items:**

```
A: 3, B: 4, C: 4, D: 3, E: 2
```

**Hash pairs:**

```
Hash(A, B) -> Bucket 1: 2
Hash(A, C) -> Bucket 2: 2
Hash(B, C) -> Bucket 3: 3
Hash(A, D) -> Bucket 4: 2
...
```

**Pass 2:**

mathematica

Frequent Items: {A, B, C, D}

Generate candidate pairs:

(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)

Count candidate pairs:

(A, B): 2

(A, C): 2

(A, D): 2

(B, C): 3

(B, D): 2

(C, D): 2

**Final Frequent Itemsets:**

css

Minimum Support Threshold = 3

Frequent Itemsets:

{B, C} (since it meets the threshold)

### Final Frequent Itemsets:

CSS

 Copy code

```
Minimum Support Threshold = 3
```

Frequent Itemsets:

```
{B, C} (since it meets the threshold)
```

### Key Advantages of PCY

1. **Reduced Memory Usage:** By using a hash table to store the count of item pairs, the PCY algorithm significantly reduces the memory required compared to storing all candidate pairs.
2. **Efficiency:** The algorithm efficiently prunes candidate pairs that are unlikely to be frequent by using the hash table to eliminate pairs early in the process.

The PCY algorithm effectively manages the large number of potential candidate pairs in frequent itemset mining, making it a powerful tool for analyzing large datasets.

## Explain the concept of CURE Clustering Algorithm with example.

### CURE Clustering Algorithm

CURE (Clustering Using Representatives) is a robust clustering algorithm designed to handle large datasets and outliers effectively. It aims to overcome the limitations of traditional clustering algorithms like k-means, which can struggle with non-spherical clusters and varying cluster sizes.

### Key Concepts of CURE

1. **Representation of Clusters:** Instead of using a single centroid to represent a cluster, CURE uses multiple representative points. These points are selected to capture the shape and spread of the cluster.
2. **Shrinking of Representative Points:** After selecting representative points, CURE shrinks them towards the centroid of the cluster by a specified factor. This helps in making the clusters tighter and reduces the impact of outliers.
3. **Hierarchical Clustering:** CURE performs agglomerative hierarchical clustering, starting with each data point as its own cluster and merging the closest pairs of clusters until the desired number of clusters is achieved.

### Steps of CURE Algorithm

1. **Random Sampling:** Select a random sample of data points from the dataset to reduce computational complexity.
2. **Initial Clustering:** Apply a clustering algorithm (such as k-means) to partition the sample into initial clusters.

3. Representative Points Selection: For each initial cluster, choose a fixed number of representative points that are well-distributed within the cluster.
4. Shrink Representative Points: Move each representative point toward the centroid of its cluster by a predefined fraction (e.g., 20%).
5. Hierarchical Clustering: Perform hierarchical clustering on the representative points to form the final clusters.
6. Assign Remaining Data Points: Assign the remaining data points (not in the initial sample) to the nearest cluster based on the closest representative point.

### Example of CURE Clustering

Consider a dataset of points in a 2D space:

Dataset={\{(1,2),(2,3),(3,1),(8,8),(9,9),(7,7),(20,20),(21,21),(19,19)\}}

Dataset={\{(1,2),(2,3),(3,1),(8,8),(9,9),(7,7),(20,20),(21,21),(19,19)\}}

1. Random Sampling: Select a random sample from the dataset (for simplicity, assume the entire dataset is small and used directly).
2. Initial Clustering: Apply an initial clustering algorithm to partition the data into initial clusters. Assume we use k-means with
3.  $k=3$
4.  $k=3$  to get initial clusters:
  - Cluster 1: {\{(1,2), (2,3), (3,1)\}}
  - Cluster 2: {\{(8,8), (9,9), (7,7)\}}
  - Cluster 3: {\{(20,20), (21,21), (19,19)\}}
5. Representative Points Selection: Choose 2 representative points for each cluster. Select the points farthest from the centroid:
  - Cluster 1: {\{(1,2), (3,1)\}}
  - Cluster 2: {\{(7,7), (9,9)\}}
  - Cluster 3: {\{(19,19), (21,21)\}}
6. Shrink Representative Points: Shrink each representative point towards the centroid by 20%:
  - Cluster 1 centroid: (2,2)
    - (1,2) -> (1.2, 2)
    - (3,1) -> (2.6, 1.2)
  - Cluster 2 centroid: (8,8)
    - (7,7) -> (7.2, 7.2)
    - (9,9) -> (8.8, 8.8)
  - Cluster 3 centroid: (20,20)
    - (19,19) -> (19.2, 19.2)
    - (21,21) -> (20.8, 20.8)
7. Hierarchical Clustering: Perform hierarchical clustering on the shrunk representative points. Merge clusters based on the closest pair of representative points until the desired number of clusters is reached (let's assume we aim for 3 clusters).
8. Assign Remaining Data Points: If any data points were not in the initial sample, assign them to the nearest cluster based on the closest representative point.

### Visualization

- Initial clusters might look like:
  - Cluster 1: Tight group around (2,2)

- Cluster 2: Tight group around (8,8)
- Cluster 3: Tight group around (20,20)
- Representative points are shrunk towards the centroids, ensuring tighter clusters and reducing the impact of outliers.

By using multiple representative points and shrinking them towards the centroid, CURE effectively captures the shape and spread of clusters, making it robust to outliers and non-spherical clusters. This method provides a better clustering solution compared to traditional algorithms, especially for large and complex datasets.

## Define Count Distinct Problem with constraints.

The Count Distinct Problem involves determining the number of distinct elements in a given dataset. The problem can be stated formally as follows:

**\*\*Given\*\*:**

- A dataset  $\{D\}$  containing  $n$  elements.
- Each element in the dataset can be of any type (e.g., integer, string).

**\*\*Objective\*\*:**

- Count the number of distinct elements in the dataset  $\{D\}$ .

**\*\*Constraints\*\*:**

1. **Memory Constraint**: The algorithm or method used to solve the Count Distinct Problem must operate within a specified memory limit.
2. **Time Constraint**: The algorithm or method must provide the count of distinct elements within a specified time limit.
3. **Accuracy Constraint**: The count of distinct elements should be accurate. This means that duplicates or erroneous entries should not be counted multiple times, and every distinct element in the dataset should be accounted for exactly once.
4. **Scalability Constraint**: The algorithm or method should be scalable to handle large datasets efficiently. It should provide accurate results even when the dataset is too large to fit entirely into memory.
5. **Resource Constraint**: The algorithm or method should utilize computational resources (CPU, memory) efficiently while solving the problem.

These constraints are essential to ensure that the Count Distinct Problem is solved effectively and efficiently, especially when dealing with large datasets or when operating under resource limitations. Various techniques and algorithms, such as hashing, probabilistic data structures (e.g., Bloom filters, HyperLogLog), and approximate counting methods, are employed to address these constraints and provide solutions to the Count Distinct Problem while satisfying the specified constraints.

Suppose data stream consists of the integers 3,1,4,1,5,9,2,6,5.  
Let the hash function being used is  $h(x) = (3x+1) \bmod 5$ . Show how Flajolet-Martin algorithm will estimate the number of distinct elements in this stream.

OR

Define following terms with respect to Association Rule Mining -

A) Support of Item B) Confidence of Rule

**A) Support of an Item**

**Support** is a measure of how frequently an item or a set of items appears in a dataset. It is defined as the proportion of transactions in the database that contain the item or itemset.

**Formula:**

$$\text{Support}(A) = \frac{\text{Number of transactions containing itemset } A}{\text{Total number of transactions}}$$

For example, if we have a dataset of 100 transactions and the item "milk" appears in 20 of those transactions, the support for "milk" would be:

$$\text{Support(milk)} = \frac{20}{100} = 0.20$$

Support helps in identifying the popularity of an item or itemset in the entire dataset.

### B) Confidence of a Rule

**Confidence** is a measure of the reliability of an association rule. It is defined as the proportion of transactions containing the itemset A that also contain the itemset B. In other words, it measures how often items in B appear in transactions that contain A.

**Formula:**

$$\text{Confidence}(A \Rightarrow B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)}$$

Here,  $A \Rightarrow B$  denotes an association rule where A is the antecedent (left-hand side) and B is the consequent (right-hand side).

For example, if the support for itemset {milk, bread} is 0.10 (i.e., 10% of all transactions contain both milk and bread) and the support for {milk} is 0.20 (i.e., 20% of all transactions contain milk), the confidence for the rule {milk}  $\Rightarrow$  {bread} would be:

$$\text{Confidence}(\{\text{milk}\} \Rightarrow \{\text{bread}\}) = \frac{\text{Support}(\{\text{milk}, \text{bread}\})}{\text{Support}(\{\text{milk}\})} = \frac{0.10}{0.20} = 0.50$$

This means that 50% of the transactions that contain milk also contain bread, indicating a moderate level of reliability for this association rule.

Consider a small database (table 1 below) with four items I= Bread, Butter, Eggs, Milk and four transactions as shown in following table. Suppose, that the minimum support and minimum confidence of an association rule are \$40 \%\$ and \$60\%\$ respectively. Find out several potential association rules.

What is a community in "Social Network Graph"? What are the standard Clustering Techniques for Graph Clustering? Explain one algorithm for finding communities in a Social Graph.

In a social network graph, a community refers to a subset of nodes (individuals or entities) within the graph that are densely connected to each other but have fewer connections with nodes outside the subset. Essentially, it represents groups of individuals who share common interests, affiliations, or interactions within the network.

Standard clustering techniques for graph clustering aim to identify these communities or clusters within the network. Some common techniques include:

1. Modularity-based methods: These methods aim to maximize the modularity of the network, which measures the quality of division of a network into communities. Modularity quantifies the density of edges within communities compared to the density of edges between communities.
2. Spectral clustering: Spectral clustering techniques use the eigenvalues and eigenvectors of a graph's adjacency matrix or Laplacian matrix to partition the graph

into clusters. It transforms the graph into a lower-dimensional space where clustering is performed using traditional techniques like k-means.

3. Hierarchical clustering: Hierarchical clustering methods construct a hierarchy of clusters by recursively merging or splitting clusters based on some similarity measure between nodes or clusters.
4. Label propagation: Label propagation algorithms start with an initial labeling of nodes and then iteratively update the labels based on the labels of neighboring nodes. Nodes with similar labels tend to form communities.

One algorithm for finding communities in a social graph is the Louvain method, which is a modularity-based approach. Here's how it works:

1. Initialization: Assign each node to its own community.
2. Modularity optimization: Iterate over each node and consider moving it to a neighboring community. The movement that increases the modularity the most is selected. Repeat this process until no further increase in modularity is possible.
3. Merge communities: After completing the first phase, merge the communities into single nodes and create a new network where the nodes represent the communities.
4. Iterate: Repeat the modularity optimization process on the new network until no further increase in modularity is possible or until a predefined stopping criterion is met.

The Louvain method efficiently identifies communities in large-scale networks by optimizing modularity, making it widely used in practice for social network analysis and other graph clustering tasks.

By computing Pearson Coefficient, determine the rating of User-ID-3 for Item-ID-1 and Item-ID-6 using table 2 below.

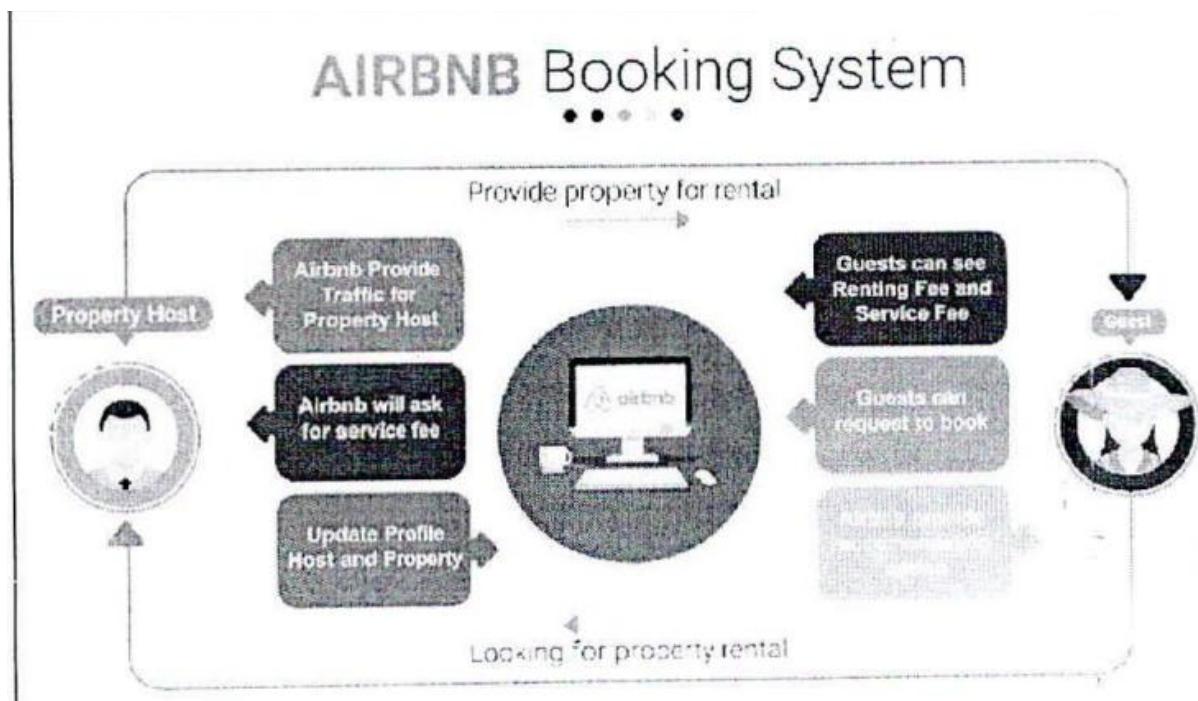
Transaction ID	Items
T1	{Bread, Butter, Eggs}
T2	{Butter, Eggs, Milk}
T3	{Butter}
T4	{Bread, Butter}

Table 1

Item ID →	1	2	3	4	5	6
User ID↓						
1	7	6	7	4	5	6
2	6	7	?	4	3	4
3	?	3	3	1	1	?
4	1	2	2	3	3	4
5	1	?	1	2	3	3

Table 2

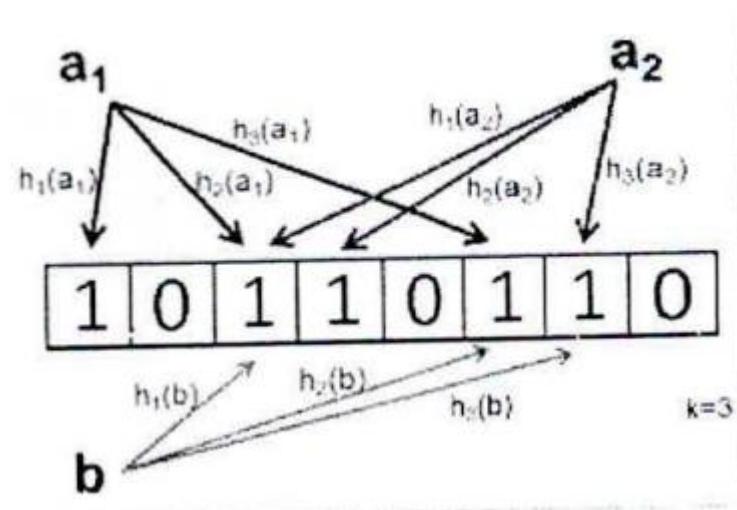
## ESE MAY 2023 - II



Airbnb is a platform that allows individuals to rent out their properties or rooms to travelers. As of 2021, Airbnb had over 7 million listings in over 100,000 cities, Airbnb has quickly become a major player in the travel industry. This industry spreads over 220 countries and regions. In the year 2020, the company's revenue was approximately \$\\\$ 2.5\$ billion.

Considering the above aspects of the case study, justify why airbnb can be considered as a big data case study.

$a_1$ ,  $a_2$  and  $a_3$  are inserted in the bloom filter as shown below. With the help of this sketch show that Bloom filter does not support deletion.



Consider a data stream received and identify an item which is frequent using the majority algorithm. Show all steps.

Pen, Pencil, Eraser, Pen, Pen, Eraser, Pencil, Pen, Pencil, Pen, Pen, Pen

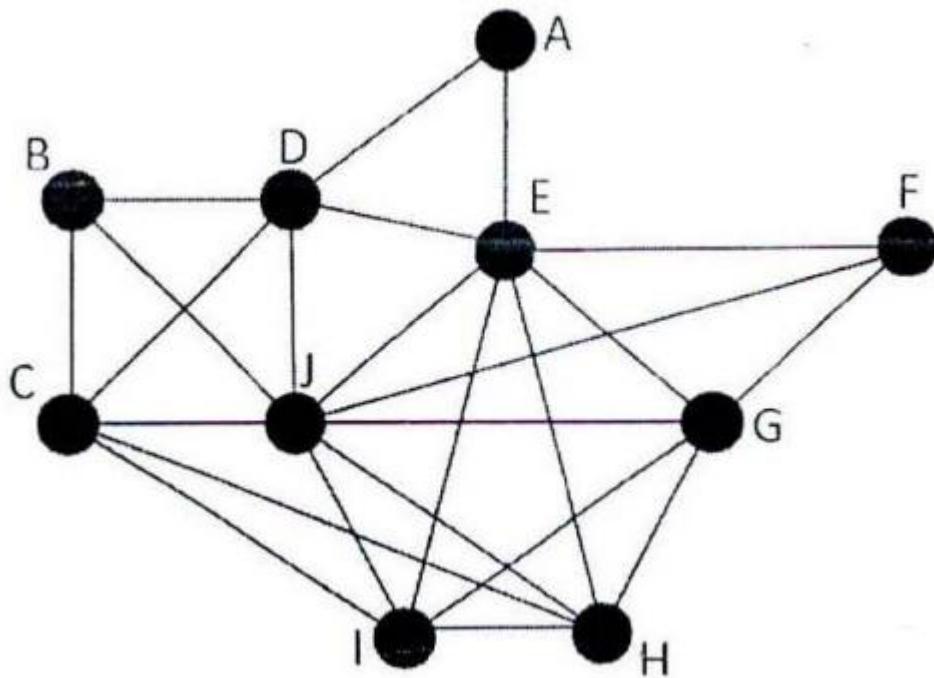
Using HITS algorithm identify Hubs and Authority pages in the following graph. Calculate hub and authority score for each node up to 3 iterations.

Apply PCY algorithm on the following dataset and show its disadvantage.

(1,2),(3,3)1(1,2),(1,2),(2,4),(1,2)1(1,2),(4,3),(1,2),(1,3),(1,2),(3,2).  
(1,2)1 (1,3) ,(2,4)

Hash function  $(i+j)\%3$

Consider the following graph as a part of a social network graph. Identify how many communities can be formed using the Clique Percolation Method for  $k=3$ . Clearly state the nodes in each community. Show all steps.



Explain the role of Singular Value Decomposition in Big data with example.

Discuss with example the role of Eigenvalue in the calculation of page rank algorithm

Reduce the dimensions of the given data set using principal component analysis.

$$\begin{bmatrix} 2 & 4 \\ 3 & 6 \\ 4 & 8 \\ 5 & 10 \end{bmatrix}$$

# ESE DECEMBER 2023

Explain different characteristics of Big Data with example?

Explain the challenges of big data and its applications with example?

What is Hadoop Architecture? Explain Hadoop component with example.

Hadoop is an open-source framework for distributed storage and processing of large datasets across clusters of commodity hardware. Its architecture consists of several core components, each serving a specific purpose in the Hadoop ecosystem.

### 1. Hadoop Distributed File System (HDFS):

- HDFS is the primary storage system used by Hadoop. It is designed to store large files across multiple machines in a fault-tolerant manner.
- Example: Suppose you have a large dataset (e.g., several terabytes of log files). HDFS splits this dataset into smaller blocks (default size is 128 MB or 256 MB) and distributes these blocks across the nodes in a Hadoop cluster. This ensures high availability and reliability even in the presence of node failures.

### 2. MapReduce:

- MapReduce is a programming model and processing engine for distributed computing on large datasets.
- Example: Consider the word count example mentioned earlier. In Hadoop MapReduce, you would write a map function to tokenize the input text into words and emit key-value pairs where the key is the word and the value is 1. Then, you'd write a reduce function to aggregate the counts of each word to produce the final word count.

### 3. YARN (Yet Another Resource Negotiator):

- YARN is a resource management layer in Hadoop that enables different data processing engines to run on the same cluster.

- Example: Suppose you have a Hadoop cluster running multiple applications simultaneously, such as MapReduce jobs, Apache Spark jobs, and Apache Hive queries. YARN allocates and manages resources (CPU, memory) dynamically among these applications, ensuring efficient resource utilization.

#### 4. Hadoop Common:

- Hadoop Common contains libraries and utilities used by other Hadoop modules.
- Example: Hadoop Common includes Java libraries for filesystem access, networking, and other common functionalities required by Hadoop components.

#### 5. Hadoop Ecosystem Components:

- Hadoop ecosystem includes various other components built on top of the core Hadoop framework, such as Apache Hive for SQL-like querying, Apache Pig for data processing, Apache HBase for NoSQL database capabilities, Apache Spark for in-memory data processing, etc.
- Example: Apache Hive allows you to run SQL-like queries on data stored in HDFS. For instance, you can write a Hive query to retrieve the top 10 most frequently occurring words from a set of log files stored in HDFS.

Overall, the Hadoop architecture enables distributed storage and processing of large datasets, providing fault tolerance, scalability, and high availability.

Differentiate Map-reduce architecture with Spark along with example?

MapReduce and Spark are both frameworks used for distributed computing, particularly for processing large datasets. While they serve similar purposes, they have some differences in terms of architecture and functionality.

#### MapReduce:

1. Architecture: In a MapReduce architecture, data processing is divided into two main phases: the Map phase and the Reduce phase. In the Map phase, input data is divided into smaller chunks and processed in parallel across multiple nodes in a cluster. The output of the Map phase is a set of intermediate key-value pairs. In the Reduce phase, these intermediate results are shuffled and sorted based on keys and then processed to produce the final output.
2. Programming Model: MapReduce follows a strict programming model where users need to define two main functions: a map function for processing input data and emitting intermediate key-value pairs, and a reduce function for aggregating and processing the intermediate results.
3. Fault Tolerance: MapReduce provides fault tolerance through replication of data and re-execution of tasks in case of failures.
4. Example: One classic example of MapReduce is word count. In this example, the input data consists of a large text document. The map function reads the document, tokenizes it into words, and emits key-value pairs where the key is the word and the value is 1. The reduce function then aggregates the counts of each word to produce the final word count.

#### Spark:

1. Architecture: Spark also follows a distributed computing architecture but introduces the concept of Resilient Distributed Datasets (RDDs). RDDs are immutable distributed collections of objects that can be processed in parallel. Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in the main program (driver program).
2. Programming Model: Spark provides a more flexible programming model compared to MapReduce. It supports various high-level APIs like RDDs, DataFrames, and Datasets, along with functional programming constructs like map, reduce, filter, etc. This allows developers to write more concise and expressive code.
3. In-Memory Computation: Unlike MapReduce, which writes intermediate results to disk between stages, Spark performs in-memory computation whenever possible, leading to faster processing times.
4. Example: An example of Spark application could be processing log files to calculate statistics like the number of unique visitors, average session

duration, etc. Using Spark's DataFrame API, you can load the log data into a DataFrame, perform transformations and aggregations, and then generate the desired statistics.

In summary, while both MapReduce and Spark are used for distributed computing, Spark offers a more flexible programming model, in-memory computation, and higher-level abstractions, making it more suitable for a wider range of use cases.

Wordlist= ['apple', 'banana', 'orange', 'apple', 'grape', 'orange', 'banana', 'apple', 'orange', 'grape', 'banana', 'apple', 'grape', 'orange', 'apple', 'grape', 'orange', 'apple', 'orange'] How you would perform a word count using the MapReduce paradigm. Describe the key steps involved in implementing the MapReduce process to calculate the frequency of each word in the provided list. Additionally, outline the functions and roles of the Mapper and Reducer tasks in the MapReduce framework, illustrating how they collaborate to accomplish the word count task efficiently

## MapReduce Paradigm for Word Count

MapReduce is a programming model used for processing large data sets with a parallel, distributed algorithm on a cluster. It consists of two main functions: the Mapper and the Reducer. The word count problem is a classic example used to explain the MapReduce framework.

## Steps Involved in Implementing Word Count Using MapReduce

1. Data Splitting:
  - The input data is split into chunks. Each chunk is processed independently by a Mapper.
2. Mapping:
  - Each Mapper processes a chunk of data, emitting key-value pairs.
  - For the word count problem, the Mapper emits each word as the key and the value 1.
3. Shuffling and Sorting:
  - The framework groups all intermediate values associated with the same key.
  - This step involves sorting the keys and transferring data from the Mappers to the Reducers.
4. Reducing:

- Each Reducer takes all values for a given key and combines them to produce the final result.
- For the word count problem, the Reducer sums the values for each word to get the total count.

## Mapper and Reducer Functions

### Mapper Function:

- Input: A chunk of the input data (e.g., a list of words).
- Process:
  - Read each word.
  - Emit key-value pairs (word, 1) for each word.
- Output: Intermediate key-value pairs.

### Reducer Function:

- Input: A key and a list of values associated with that key (e.g., ('apple', [1, 1, 1, 1])).
- Process:
  - Sum the values for each key.
- Output: Final key-value pairs with the total counts for each word.

## Example Walkthrough

Given the wordlist:

```
wordlist = ['apple', 'banana', 'orange', 'apple', 'grape', 'orange', 'banana', 'apple',
 'orange', 'grape', 'banana', 'apple', 'grape', 'orange', 'apple', 'grape',
 'orange', 'apple', 'orange', 'grape']
```

### Mapping Phase

Each Mapper processes a portion of the list and emits key-value pairs. For example:

plaintext

Copy code

#### Mapper 1:

```
Input: ['apple', 'banana', 'orange', 'apple']
Output: [('apple', 1), ('banana', 1), ('orange', 1), ('apple', 1)]
```

Explain

#### Mapper 2:

```
Input: ['grape', 'orange', 'banana', 'apple']
Output: [('grape', 1), ('orange', 1), ('banana', 1), ('apple', 1)]
```

...

### Shuffling and Sorting Phase

The framework collects all emitted key-value pairs and groups them by key:

## Shuffling and Sorting Phase

The framework collects all emitted key-value pairs and groups them by key:

```
plaintext Copy code
Grouped by Key:
'apple': [1, 1, 1, 1, 1, 1, 1, 1]
'banana': [1, 1, 1]
'orange': [1, 1, 1, 1, 1, 1]
'grape': [1, 1, 1, 1, 1]
```

## Reducing Phase

Each Reducer sums the values for each key:

```
plaintext Copy code
Reducer for 'apple':
Input: ('apple', [1, 1, 1, 1, 1, 1, 1, 1])
Output: ('apple', 8)

Reducer for 'banana':
Input: ('banana', [1, 1, 1])
Output: ('banana', 3)

Reducer for 'orange':
Input: ('orange', [1, 1, 1, 1, 1, 1])
Output: ('orange', 6)

Reducer for 'grape':
Input: ('grape', [1, 1, 1, 1, 1])
Output: ('grape', 5)
```

The final output from all Reducers is the word count:

```
plaintext

('apple', 8)
('banana', 3)
('orange', 6)
('grape', 5)
```

Explain the advantages and limitations of utilizing Spark SQL for processing and analyzing large-scale datasets compared to traditional SQL querying on relational databases?

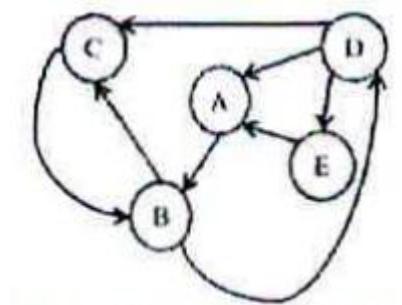
Explain various components of Spark Projects?

If the current state of the stream is  
10111110111010011100101011100111 create buckets for  
the given stream and find the final state of the stream if  
101100101 bits added in sequence one by one into stream  
consider bit by bit and show each bit addition separately using  
DGIM Algorithm ?

Use N = 11, We have set Bloom Filter to 00000000000. use  
 $h_1(x)$  = Take odd number of bits from right of the binary  
representation of x.  $h_2(x)$ = Take even number of bits from right  
of the binary representation of x. The stream elements(X) are  
28,358,189,119,196,484,76. What is the final value of the  
bloom filter?

Discuss Spark SQL's architectural components, optimizations, and query execution mechanisms that enable enhanced performance and scalability in big data processing scenarios

What is PageRank? Define the structure of the web and how top sensitive page ranks are given? Calculate page ranks of all pages given in below graph.



Discuss in detail the implications of utilizing a property graph database model like Neo4j in a social network setting compared to a traditional relational database.

How Item -Item collaborative filtering and user -user collaborative filtering similarity carried out explain with the help of example of 5 movies and 8 users. Use 5-star ratings?