BIG DATA ANALYSIS – UNIT-3

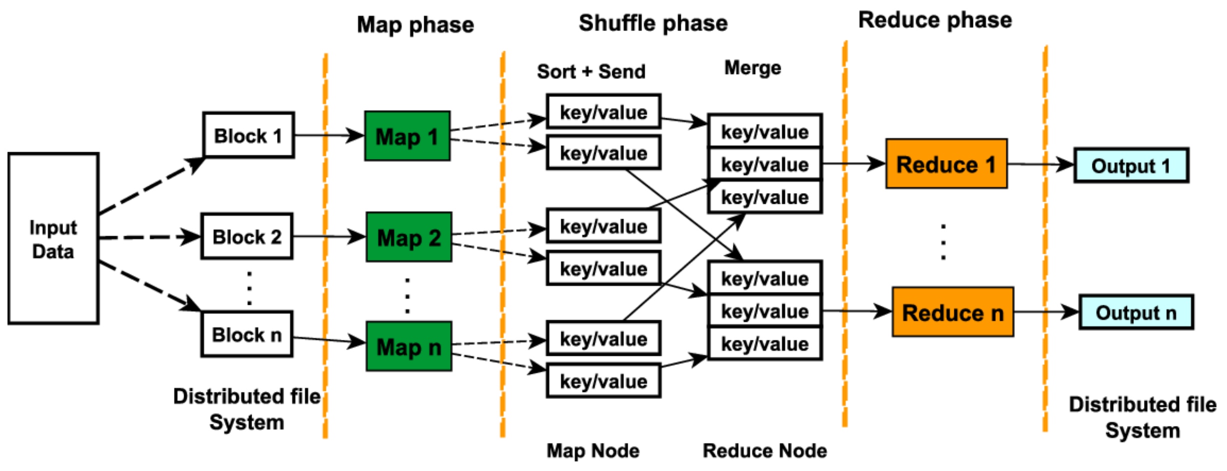
1) What is MapReduce?

Introduction To MapReduce: MapReduce was introduced by Google in 2004 to support distributed computing. It enabled Google to process large sets of data effectively, leading to its public adoption and the creation of many open-source versions, such as Apache Hadoop.

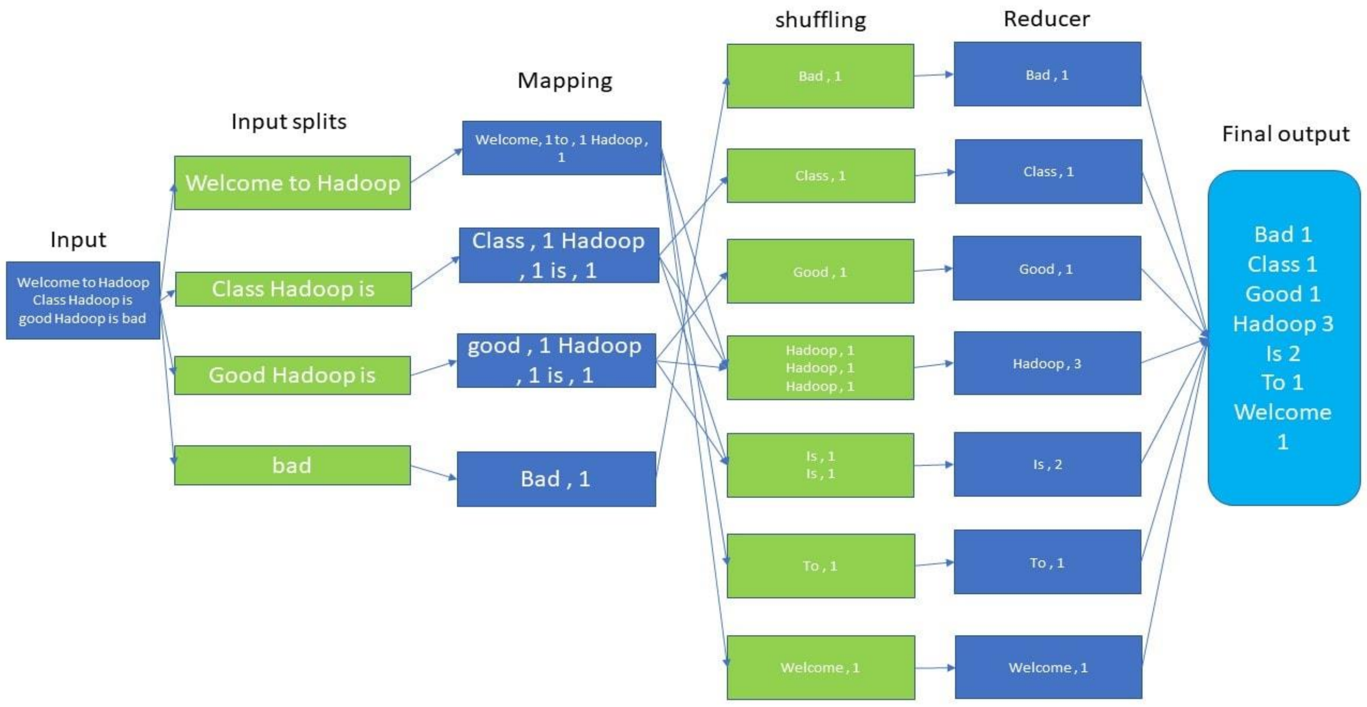
MapReduce is a Hadoop structure utilized for composing applications that can process large amounts of data on clusters. It can be known as a programming model in which we can handle huge datasets across PC clusters. This application permits information to be put away in a distributed form. It works on huge volumes of data and enormous scope of computing.

MapReduce consists of **two phases** namely **Map and Reduce** Map generally deals with the splitting and mapping of data while reducing tasks shuffle and reducing the data. Hadoop is fully capable of running MapReduce programs that are written in various languages: python, java, and C++. This is very useful for performing large-scale data analysis using multiple machines in the cluster.

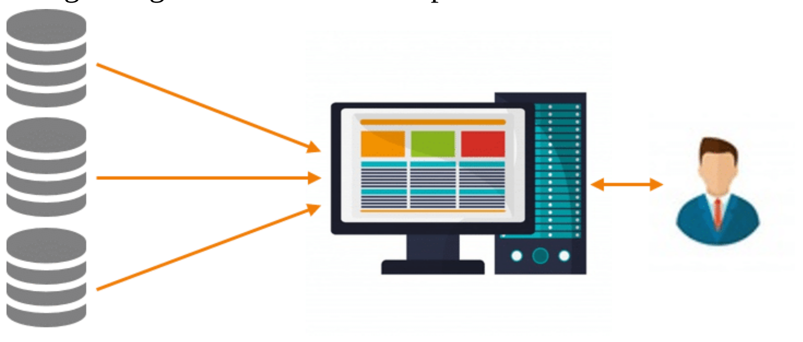
Mechanism behind MapReduce



Ex:



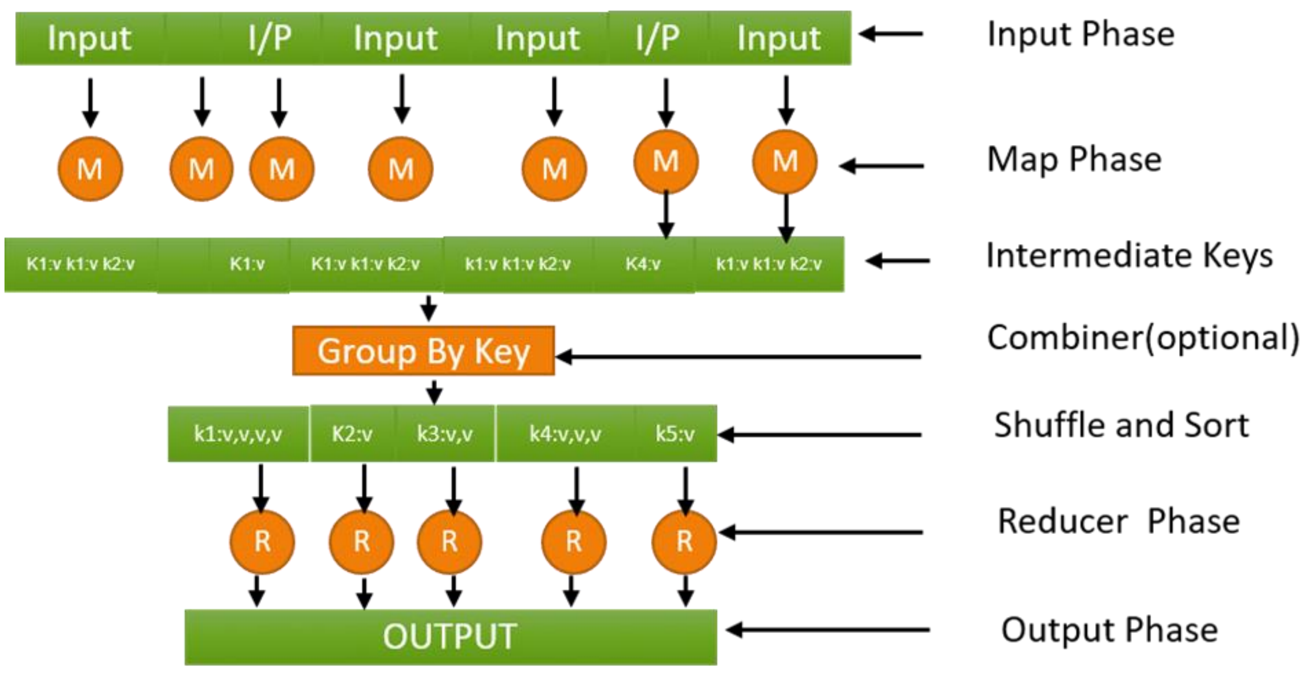
Need of MapReduce: Normally traditional enterprise system used centralized processing server to store and process data. This system is not suitable to process large volumes of data. If we are trying to process multiple files concurrently, then the centralized system creates too much of a bottleneck. Google gave the solution to this bottleneck issue by using an algorithm known as MapReduce.



In the MapReduce process the large tasks are split into smaller tasks, and then they are assigned to many systems.

Working of MapReduce: The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples. The reduced task is always performed after the map job.



Input Phase – Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

Map – Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.

Intermediate Keys – The key-value pairs generated by the mapper are known as intermediate keys.

Combiner − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.

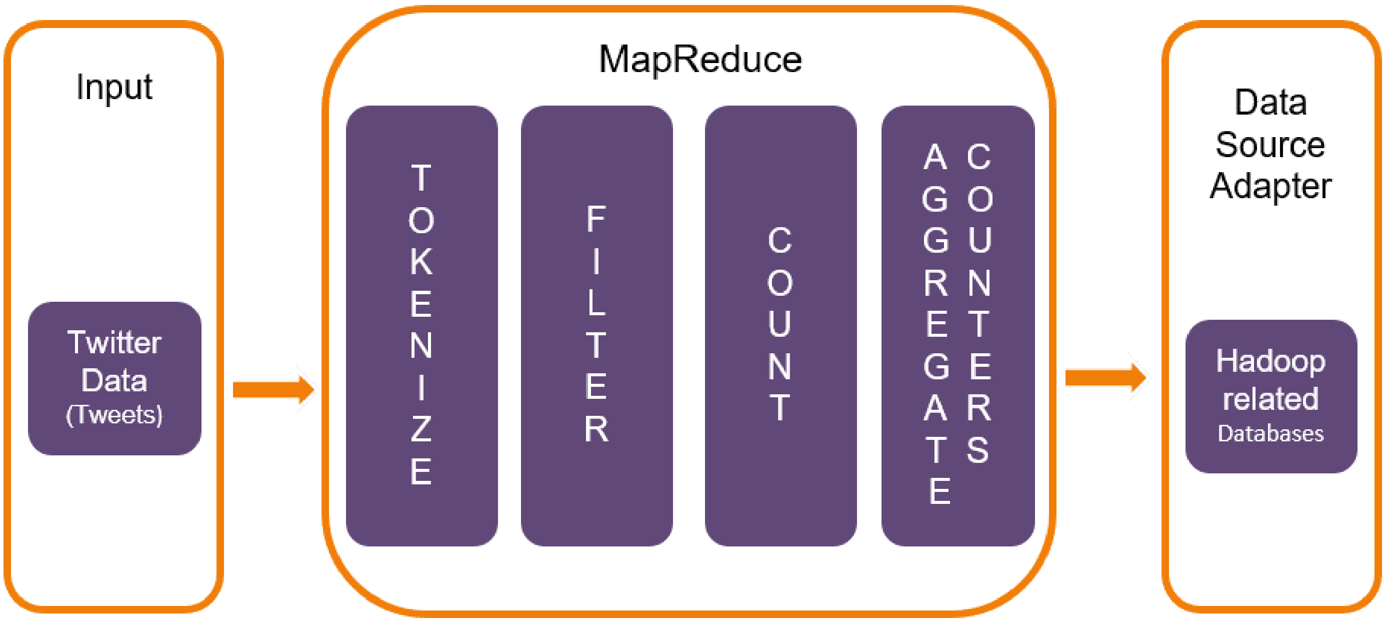
Shuffle and Sort − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

Reducer − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

Output Phase − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

The Workflow of MapReduce with an Example

On a daily basis the micro-blogging site Twitter receives nearly 500 million tweets, i.e., 3000 tweets per second. We can see the illustration on Twitter with the help of MapReduce.



In the above example Twitter data is an input, and MapReduce performs the actions like Tokenize, filter, count and aggregate counters.

Tokenize: Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
Filter: It filters the unwanted words from maps of tokens.
Count: Generates a token counter per word.
Aggregate counters: Prepares a combination of similar counter values into small manageable units.

Application Of MapReduce

Entertainment: To discover the most popular movies, based on what you like and what you watched in this case Hadoop MapReduce help you out. It mainly focuses on their logs and clicks.

E-commerce: Numerous E-commerce suppliers, like Amazon, Walmart, and eBay, utilize the MapReduce programming model to distinguish most loved items dependent on clients' inclinations or purchasing behavior. It incorporates making item proposal Mechanisms for E-commerce inventories, examining website records, buy history, user interaction logs, etc.
Data Warehouse: We can utilize MapReduce to analyze large data volumes in data warehouses while implementing specific business logic for data insights.
Fraud Detection: Hadoop and MapReduce are utilized in monetary enterprises, including organizations like banks, insurance providers, installment areas for misrepresentation recognition, pattern distinguishing proof, or business metrics through transaction analysis.

Advantage of MapReduce

- **Fault tolerance:** It can handle failures without downtime.
- **Speed**: It splits, shuffles, and reduces the unstructured data in a short time.
- **Cost-effective:** Hadoop MapReduce has a scale-out feature that enables users to process or store the data in a cost-effective manner.
- **Scalability:** It provides a highly scalable framework. MapReduce allows users to run applications from many nodes.
- **Parallel Processing:** Here multiple job-parts of the same dataset can be processed in a parallel manner. This can reduce the task that can be taken to complete a task.

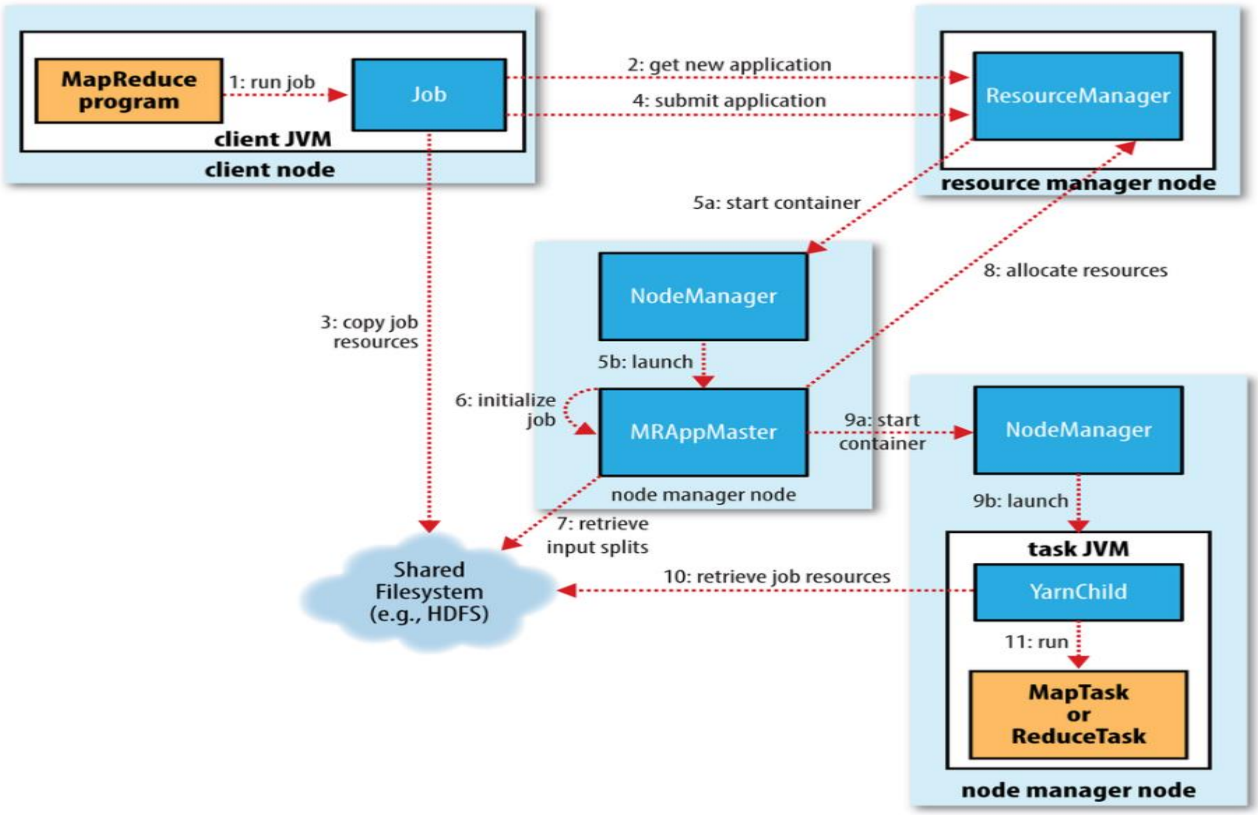
Limitations Of MapReduce

- MapReduce cannot cache the intermediate data in memory for a further requirement which diminishes the performance of Hadoop.
- It is only suitable for Batch Processing of a Huge amounts of Data.

2) How a MapReduce Job Run Works?

There is only one function we need to use to start a MapReduce job: **submit()** on a Job object. This method call hides a lot of processing that occurs in the background.

Anatomy of a Map Reduce Job run



In the provided figure, the entire procedure is depicted. There are **five** independent entities:

1. The client, which submits the MapReduce job.
2. The YARN resource manager, which coordinates the allocation of compute resources on the cluster.
3. The YARN node managers, which launch and monitor the compute containers on machines in the cluster.
4. The MapReduce **application master**, which coordinates the tasks running **the MapReduce job** The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
5. The distributed filesystem, which is used for sharing job files between the other entities.

Job Submission :

- The submit() method on Job creates an internal JobSubmitter instance and all submitJobInternal() on it.
- Having submitted the job, waitForCompletion polls the job's progress once per second and reports the progress to the console if it has changed since the last report.
- When the job completes successfully, the job counters are displayed, Otherwise, the error that caused the job to fail is logged to the console. The job submission process implemented by JobSubmitter does the following:
- Asks the resource manager for a new application ID, used for the MapReduce job ID.
- Checks the output specification of the job For example, if the output directory has not been specified or it already exists, the job is not submitted and an error is thrown to the MapReduce program.
- Computes the input splits for the job If the splits cannot be computed (because the input paths don't exist, for example), the job is not submitted and an error is thrown to the MapReduce program.
- Copies the resources needed to run the job, including the job JAR file, the configuration file, and the computed input splits, to the shared filesystem in a directory named after the job ID.
- Submits the job by calling submitApplication() on the resource manager.

Job Initialization :

- When the resource manager receives a call to its submitApplication() method, it hands off the request to the YARN scheduler.
- The scheduler allocates a container, and the resource manager then launches the application master's process there, under the node manager's management.
- The application master for MapReduce jobs is a Java application whose main class is MRAppMaster .
- It initializes the job by creating a number of bookkeeping objects to keep track of the job's progress, as it will receive progress and completion reports from the tasks.
- It retrieves the input splits computed in the client from the shared filesystem.
- It then creates a map task object for each split, as well as a number of reduce task objects determined by the mapreduce.job.reduces property (set by the setNumReduceTasks() method on Job).

Task Assignment:

- If the job does not qualify for running as an uber task, then the application master requests containers for all the map and reduce tasks in the job from the resource manager .
- Requests for map tasks are made first and with a higher priority than those for reduce tasks, since all the map tasks must complete before the sort phase of the reduce can start.

- Requests for reduce tasks are not made until 5% of map tasks have completed.

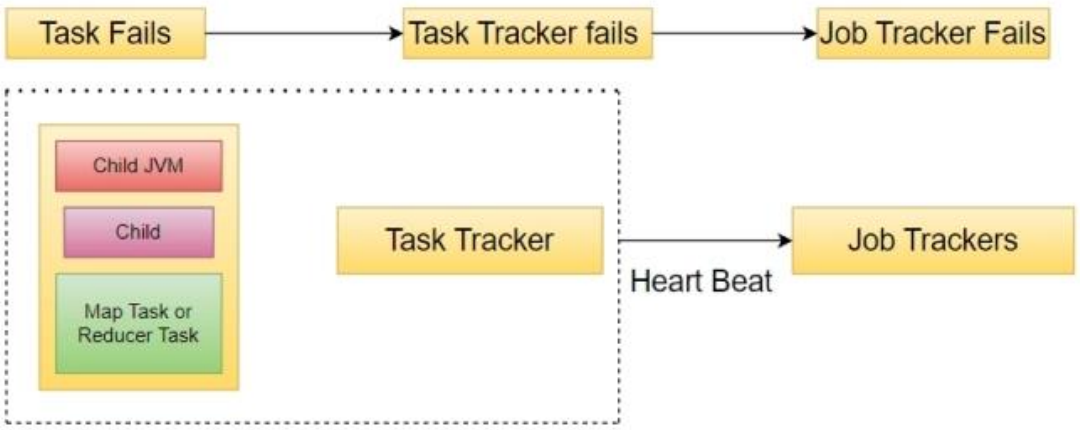
Task Execution:

- Once a task has been assigned resources for a container on a particular node by the resource manager's scheduler, the application master starts the container by contacting the node manager.
- The task is executed by a Java application whose main class is YarnChild. Before it can run the task, it localizes the resources that the task needs, including the job configuration and JAR file, and any files from the distributed cache.
- Finally, it runs the map or reduce task.

3) What are the various types of Failures in MapReduce

There are generally 3 types of failures in MapReduce.

- Task Failure
- TaskTracker Failure
- JobTracker Failure



<u>Task Failure</u>: In Hadoop, **task failure** is similar to an employee making a mistake while doing a task. Consider you are working on a large project that has been broken down into smaller jobs and assigned to different employees in your team. If one of the team members fails to do their task correctly, the entire project may be compromised. Similarly, in Hadoop, if a job fails due to a mistake or issue, it could affect overall data processing, causing delays or faults in the final result.

Reasons for Task Failure

- **Limited memory:** A task can fail if it runs out of memory while processing data.
  Failures of disk: If the disk that stores data or intermediate results fails, tasks that depend on that data may fail.
  Issues with software or hardware: Bugs, mistakes, or faults in software or hardware components can cause task failures.

Steps to Overcome Task Failure

- **Increase memory allocation:** Assign extra memory to jobs to ensure they have the resources to process the data.
- **Implement fault tolerance mechanisms:** Using data replication and checkpointing techniques to defend against disc failures and retrieve lost data.

- **Regularly update software and hardware:** Keep the Hadoop framework and supporting hardware up to date to fix bugs, errors, and performance issues that can lead to task failures.

TaskTracker Failure: A **TaskTracker** in Hadoop is similar to an employee responsible for executing certain tasks in a large project. If a TaskTracker fails, it signifies a problem occurred while an employee worked on their assignment. This can interrupt the entire project, much as when a team member makes a mistake or encounters difficulties with their task, producing delays or problems with the overall project's completion. To avoid TaskTracker failures, ensure the TaskTracker's hardware and software are in excellent working order and have the resources they need to do their jobs successfully.

Reasons for TaskTracker Failure

- **Hardware issues:** Just as your computer's parts can break or stop working properly, the TaskTracker's hardware (such as the processor, memory, or disc) might fail or stop operating properly. This may prohibit it from carrying out its duties.
- **Software problems or errors:** The software operating on the TaskTracker may contain bugs or errors that cause it to cease working properly. It's similar to when an app on your phone fails and stops working properly.
- **Overload or resource exhaustion:** It may struggle to keep up if the TaskTracker becomes overburdened with too many tasks or runs out of resources such as memory or processing power. It's comparable to being overburdened with too many duties or running out of storage space on your gadget.

Steps to Overcome TaskTracker Failure

- **Update software and hardware on a regular basis:** Keep the Hadoop framework and associated hardware up to date to correct bugs, errors, and performance issues that might lead to task failures.
- **Upgrade or replace hardware:** If TaskTracker's hardware is outdated or insufficiently powerful, try upgrading or replacing it with more powerful components. It's equivalent to purchasing a new, upgraded computer to handle jobs more efficiently.
- **Restart or reinstall the program:** If the TaskTracker software is causing problems, a simple restart or reinstall may be all that is required. It's the same as restarting or reinstalling an app to make it work correctly again.

JobTracker Failure: A **JobTracker** in Hadoop is similar to a supervisor or manager that oversees the entire project and assigns tasks to TaskTrackers (employees). If a JobTracker fails, it signifies the supervisor is experiencing a problem or has stopped working properly. This can interrupt the overall project's coordination and development, much as when a supervisor is unable to assign assignments or oversee their completion. To avoid JobTracker failures, it is critical to maintain the JobTracker's hardware and software, ensure adequate resources, and fix any issues or malfunctions as soon as possible to keep the project going smoothly.

Reasons for JobTracker Failure

- **Database connectivity:** The JobTracker stores job metadata and state information in a backend database (usually Apache Derby or MySQL). JobTracker failures can occur if there are database connectivity issues, such as network problems or database server failures.

- **Security problems:** JobTracker failures can be caused by security issues such as authentication or authorization failures, incorrectly configured security settings or key distribution and management issues.

Steps to Overcome JobTracker Failure

- **Avoiding Database Connectivity:** To avoid database connectivity failures in the JobTracker, ensure optimized database configuration, robust network connections, and high availability techniques are implemented. Retrying connections, monitoring, and backups are all useful.

- **To overcome security-related problems:** implement strong authentication and authorization, enable SSL/TLS for secure communication, keep software updated with security patches, follow key management best practices, conduct security audits, and seek expert guidance for vulnerability mitigation and compliance with security standards.

4) What are the Features of MapReduce?

1. Scalability

Apache Hadoop is a highly scalable framework. This is because of its ability to store and distribute huge data across plenty of servers. All these servers were inexpensive and can operate in parallel. We can easily scale the storage and computation power by adding servers to the cluster.

Hadoop MapReduce programming enables organizations to run applications from large sets of nodes which could involve the use of thousands of terabytes of data.

Hadoop MapReduce programming enables business organizations to run applications from large sets of nodes. This can use thousands of terabytes of data.

2. Flexibility

MapReduce programming enables companies to access new sources of data. It enables companies to operate on different types of data. It allows enterprises to access structured as well as unstructured data, and derive significant value by gaining insights from the multiple sources of data.

Additionally, the MapReduce framework also provides support for the multiple languages and data from sources ranging from email, social media, to clickstream.

The MapReduce processes data in simple key-value pairs thus supports data type including meta-data, images, and large files. Hence, MapReduce is flexible to deal with data rather than traditional DBMS.

3. Security and Authentication

The MapReduce programming model uses HBase and HDFS security platform that allows access only to the authenticated users to operate on the data. Thus, it protects unauthorized access to system data and enhances system security.

4. Cost-effective solution

Hadoop’s scalable architecture with the MapReduce programming framework allows the storage and processing of large data sets in a very affordable manner.

5. Fast

Hadoop uses a distributed storage method called as a Hadoop Distributed File System that basically implements a mapping system for locating data in a cluster.

The tools that are used for data processing, such as MapReduce programming, are generally located on the very same servers that allow for the faster processing of data.

So, Even if we are dealing with large volumes of unstructured data, Hadoop MapReduce just takes minutes to process terabytes of data. It can process petabytes of data in just an hour.

6. Simple model of programming

Amongst the various features of Hadoop MapReduce, one of the most important features is that it is based on a simple programming model. Basically, this allows programmers to develop the MapReduce programs which can handle tasks easily and efficiently.

The MapReduce programs can be written in Java, which is not very hard to pick up and is also used widely. So, anyone can easily learn and write MapReduce programs and meet their data processing needs.

7. Parallel Programming

One of the major aspects of the working of MapReduce programming is its parallel processing. It divides the tasks in a manner that allows their execution in parallel.
The parallel processing allows multiple processors to execute these divided tasks. So the entire program is run in less time.

8. Availability and resilient nature

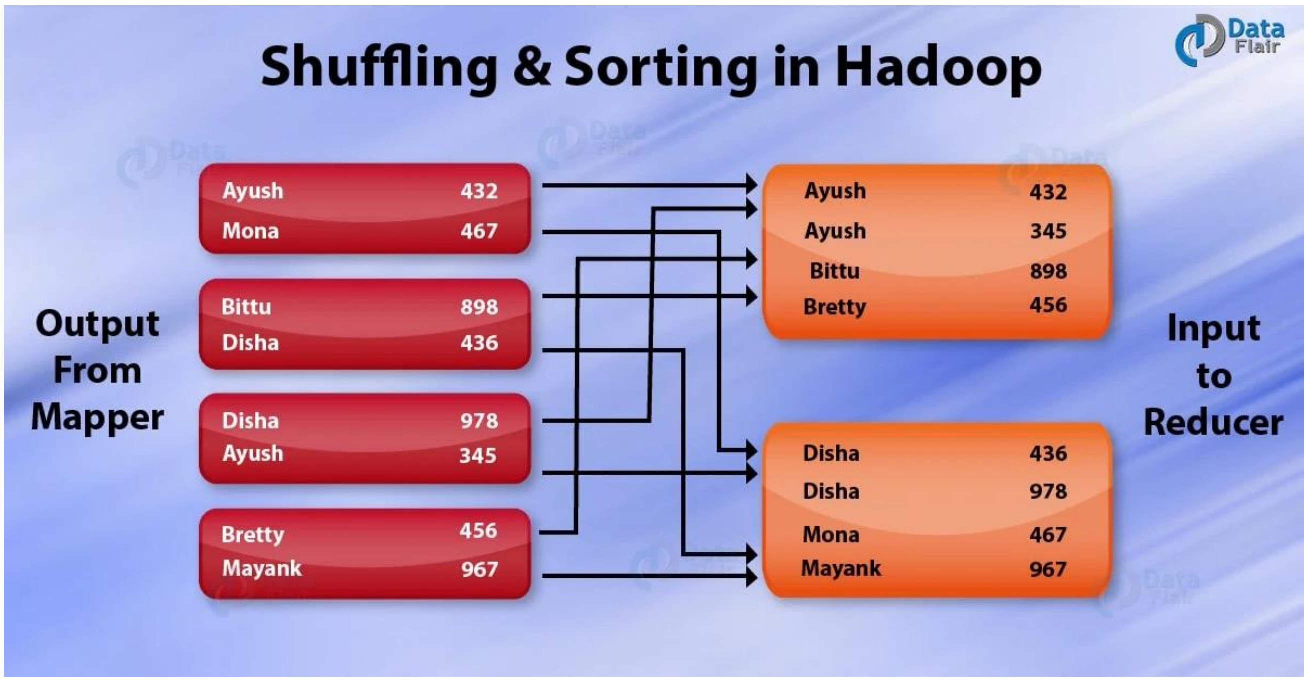
Whenever the data is sent to an individual node, the same set of data is forwarded to some other nodes in a cluster. So, if any particular node suffers from a failure, then there are always other copies present on other nodes that can still be accessed whenever needed. This assures high availability of data.

One of the major features offered by Apache Hadoop is its fault tolerance. The Hadoop MapReduce framework has the ability to quickly recognizing faults that occur.

It then applies a quick and automatic recovery solution. This feature makes it a game-changer in the world of big data processing.

5) What is Shuffling and Sorting in Map Reduce?

Shuffle phase in Hadoop transfers the map output from Mapper to a Reducer in MapReduce. **Sort phase** in MapReduce covers the merging and sorting of map outputs. Data from the mapper are grouped by the key, split among reducers and sorted by the key. Every reducer obtains all values associated with the same key. Shuffle and sort phase in Hadoop occur simultaneously and are done by the MapReduce framework.



Shuffling in MapReduce: The process of transferring data from the mappers to reducers is known as shuffling i.e. the process by which the system performs the sort and transfers the map output to the reducer as input. So, MapReduce shuffle phase is necessary for the reducers, otherwise, they would not have any input (or input from every mapper). As shuffling can start even before the map phase has finished so this saves some time and completes the tasks in lesser time.

Sorting in MapReduce: The keys generated by the mapper are automatically sorted by MapReduce Framework, i.e. Before starting of reducer, all intermediate key-value pairs in MapReduce that are generated by mapper get sorted by key and not by value. Values passed to each reducer are not sorted; they can be in any order.

Sorting in Hadoop helps reducer to easily distinguish when a new reduce task should start. This saves time for the reducer. Reducer starts a new reduce task when the next key in the sorted input data is different than the previous. Each reduce task takes key-value pairs as input and generates key-value pair as output.

Note that shuffling and sorting in Hadoop MapReduce is not performed at all if you specify zero reducers (setNumReduceTasks(0)). Then, the MapReduce job stops at the map phase, and the map phase does not include any kind of sorting (so even the map phase is faster).

Some important key aspects of the Shuffle and Sort in MapReduce

- Transferring data from Mapper to Reducer comes under shuffling.
- The grouping of key-value pairs reduces the operating time because during the Reduce Phase, we can operate the key-value pairs having the same key simultaneously.
- The sorting phase is necessary to ensure that the key-value pairs with the same key are grouped together.
- The sorted key-value pairs are divided into segments so that each segment can be processed independently by the Reduce Phase. These segments are divided in such a way that each segment has the key-value pairs with the same key.
- It enables parallel processing, reduces network traffic and improves the overall performance.

Practices to optimize the Shuffle and Sort in MapReduce

- Using the custom partitioners is beneficial because they select the partition number based on the hash value, which is a constant time operation and improves the performance of the Shuffle and Sort Phase.

- The number of partitions used in the Shuffle and Sort Phase can significantly impact the performance. Choosing the number of partitions as the multiple of the number of nodes in a Reduce Phase is good practice.

- The use of combiners can optimize the Shuffle and Sort Phase by reducing the amount of data that needs to be transferred to the Reducers by aggregating the output of the Mappers.

- If Sorting is optional for any task, it would be beneficial not to Sort the key-value pairs because Sorting is time-consuming.

6) What are the various types and formats of Map Reduce?

MapReduce Types: *Mapping* is the core technique of processing a list of data elements that come in pairs of keys and values. The map function applies to individual elements defined as key-value pairs of a list and produces a new list. The general idea of map and reduce function of Hadoop can be illustrated as follows:

map: (K1, V1) -> list (K2, V2)
reduce: (K2, list(V2)) -> list (K3, V3)

The input parameters of the key and value pair, represented by K1 and V1 respectively, are different from the output pair type: K2 and V2. The reduce function accepts the same format output by the map, but the type of output again of the reduce operation is different: K3 and V3.

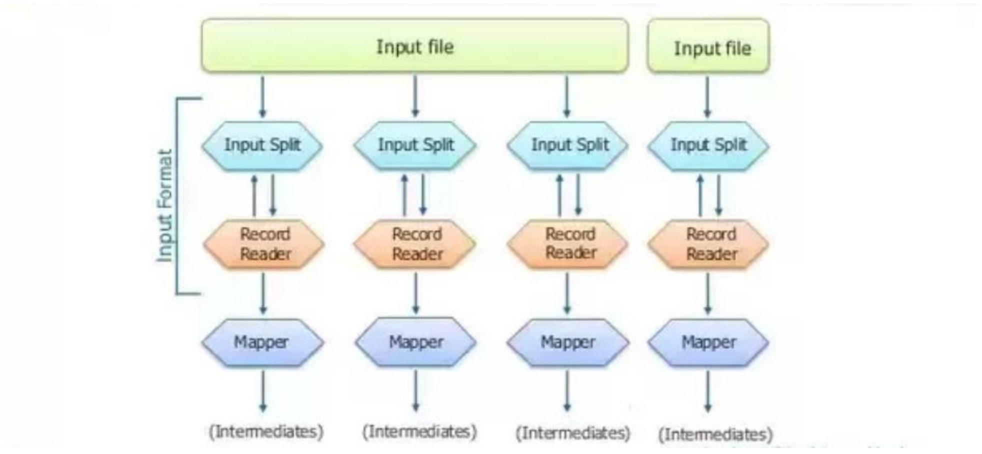
The ***OutputCollector*** is the generalized interface of the Map-Reduce framework to facilitate collection of data output either by the *Mapper* or the *Reducer*. These outputs are nothing but intermediate output of the job. Therefore, they must be parameterized with their types. The *Reporter* facilitates the Map-Reduce application to report progress and update counters and status information. If, however, the combine function is used, it has the same form as the reduce function and the output is fed to the reduce function. This may be illustrated as follows:

map: (K1, V1) -> list (K2, V2)
combine: (K2, list(V2)) -> list (K2, V2)
reduce: (K2, list(V2)) -> list (K3, V3)

The partition function operates on the intermediate key-value types. It controls the partitioning of the keys of the intermediate map outputs. The key derives the partition using a typical hash function. The total number of partitions is the same as the number of reduce tasks for the job. The partition is determined only by the key ignoring the value.

Hadoop InputFormat: Hadoop InputFormat provides the input specification for Map-Reduce job execution. InputFormat specifies how to divide and read input files. InputFormat is the initial phase in MapReduce job execution. It is also in charge of generating input splits and separating them into records. One of the core classes in MapReduce is InputFormat, which provides the following functionality:

- InputFormat determines which files or other objects to accept as input.
- It also specifies data splits. It specifies the size of each Map task as well as the potential execution server.
- The RecordReader is defined by Hadoop InputFormat. It is also in charge of reading actual records from input files.



Types of InputFormat in MapReduce

In Hadoop, there are various MapReduce types for InputFormat that are used for various purposes. Let us now look at the MapReduce types of InputFormat:

1.FileInputFormat: It serves as the foundation for all file-based InputFormats. FileInputFormat also provides the input directory, which contains the location of the data files. When we start a MapReduce task, FileInputFormat returns a path with files to read. This InpuFormat will read all files. Then it divides these files into one or more InputSplits.

2. TextInputFormat: It is the standard InputFormat. Each line of each input file is treated as a separate record by this InputFormat. It does not parse anything. TextInputFormat is suitable for raw data or line-based records, such as log files. Hence:

1. **Key**: It is the byte offset of the first line within the file (not the entire file split). As a result, when paired with the file name, it will be unique.
2. **Value**: It is the line's substance. It does not include line terminators.

***3.* KeyValueTextInputFormat:** It is comparable to TextInputFormat. Each line of input is also treated as a separate record by this InputFormat. While TextInputFormat treats the entire line as the value, KeyValueTextInputFormat divides the line into key and value by a tab character ('/t'). Hence:

- Key: Everything up to and including the tab character.
- Value: It is the remaining part of the line after the tab character.

4. SequenceFileInputFormat: It's an input format for reading sequence files. Binary files are sequence files. These files also store binary key-value pair sequences. These are block-compressed and support direct serialization and deserialization of a variety of data types. Hence Key & Value are both user-defined.

5. SequenceFileAsTextInputFormat: It is a subtype of SequenceFileInputFormat. The sequence file key values are converted to Text objects using this format. As a result, it converts the keys and values by running 'tostring()' on them. As a result, SequenceFileAsTextInputFormat converts sequence files into text-based input for streaming.

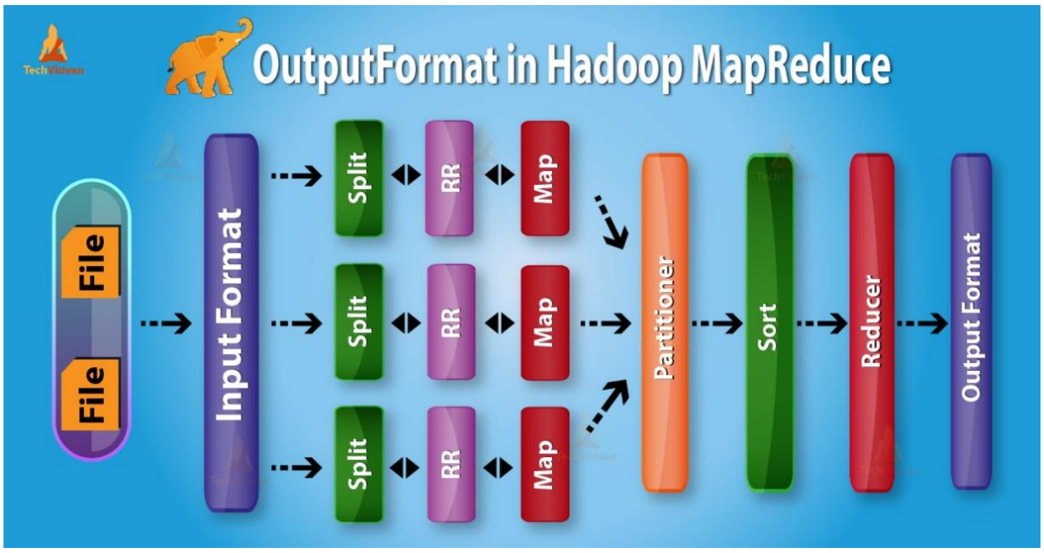
6. NlineInputFormat: It is a variant of TextInputFormat in which the keys are the line's byte offset. And values are the line's contents. As a result, each mapper receives a configurable number of lines of TextInputFormat and KeyValueTextInputFormat input. The number is determined by the magnitude of the split. It is also dependent on the length of the lines. So, if we want our mapper to accept a specific amount of lines of input, we use NLineInputFormat.

- N- It is the number of lines of input received by each mapper.
- Each mapper receives exactly one line of input by default (N=1).
- Assuming N=2, each split has two lines. As a result, the first two Key-Value pairs are distributed to one mapper. The second two key-value pairs are given to another mapper.

6. **DBInputFormat:** Using JDBC, this InputFormat reads data from a relational Database. It also loads small datasets, which might be used to connect with huge datasets from HDFS using multiple inputs. Hence:
    - Key: LongWritables
    - Value: DBWritables.

Output Format in MapReduce

The output format classes work in the opposite direction as their corresponding input format classes. **OutputFormat** check the output specification for execution of the Map-Reduce job.



The OutputFormat and InputFormat functions are similar. OutputFormat instances are used to write to files on the local disk or in **HDFS.** In MapReduce job execution on the basis of output specification;

- Hadoop MapReduce job checks that the output directory does not already present.
- OutputFormat in MapReduce job provides the RecordWriter implementation to be used to write the output files of the job. Then the output files are stored in a FileSystem.

Types of OutputFormat in MapReduce

There are various types of OutputFormat which are as follows:

1. TextOutputFormat

The default OutputFormat is TextOutputFormat. It writes (key, value) pairs on individual lines of text files. Its keys and values can be of any type. The reason behind is that TextOutputFormat turns them to string by calling **toString()** on them. It separates key-value pair by a tab character. By using **MapReduce.output.textoutputformat.separator** property we can also change it. KeyValueTextOutputFormat is also used for reading these output text files.

2. SequenceFileOutputFormat

This OutputFormat writes sequences files for its output. SequenceFileInputFormat is also intermediate format use between MapReduce jobs. It serializes arbitrary data types to the file.

And the corresponding SequenceFileInputFormat will deserialize the file into the same types. It presents the data to the next **mapper** in the same manner as it was emitted by the previous reducer. Static methods also control the compression.

3. SequenceFileAsBinaryOutputFormat

It is another variant of SequenceFileInputFormat. It also writes keys and values to sequence file in binary format.

4. MapFileOutputFormat

It is another form of FileOutputFormat. It also writes output as map files. The framework adds a key in a MapFile in order. So we need to ensure that reducer emits keys in sorted order.

5. MultipleOutputs

This format allows writing data to files whose names are derived from the output keys and values.

6. LazyOutputFormat

In MapReduce job execution, FileOutputFormat sometimes create output files, even if they are empty. LazyOutputFormat is also a wrapper OutputFormat.

7. DBOutputFormat

It is the OutputFormat for writing to relational databases and HBase. This format also sends the reduce output to a SQL table. It also accepts key-value pairs. In this, the key has a type extending DBwritable.