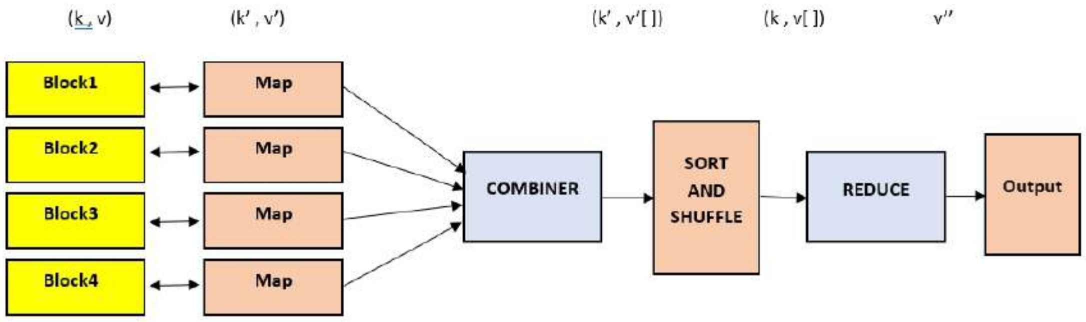
1. What is MapReduce?

MapReduce is a **programming model** or pattern within the Hadoop framework that is used to **access or process big data stored** in the Hadoop File System (**HDFS**).

Here, we can write applications to run huge amounts of data in parallel and large clusters of commodity hardware in a reliable manner.

It is a core component, integral to the functioning of the Hadoop framework.



Different Phases of MapReduce:-

MapReduce model has three major phase.

- **Map**
- Shuffle and Sort
- **Reduce**

Map:- It is the first phase of MapReduce programming. Map Phase accepts key-value pairs as input as (k, v), where the key represents the Key address of each record and the value represents the entire record content. The output of the Map phase will also be in the key-value format (k', v').

Shuffle and Sort :- The output of various mapping parts (k', v'), then goes into Shuffling and Sorting phase. All the different values are grouped together based on same keys. The output of the Shuffling and Sorting phase will be key-value pairs again as key and array of values (k, v[ ]).

Reduce :- The output of the Shuffling and Sorting phase (k, v[]) will be the input of the Reducer phase. In this phase reducer function's logic is executed and all the values are Collected against their corresponding keys. Reducer stabilize outputs of various mappers and computes the final output.

2. Define MapReduce Framework and its functions.

MapReduce is a programming model and associated implementation framework for processing and generating large datasets that can be distributed across a cluster of computers. Google popularized it and is widely used for big data processing tasks. The MapReduce framework consists of two main functions: Map and Reduce.

Map Function:

- The Map function takes a set of data and converts it into another set of data, where individual elements are broken down into key-value pairs.
- Each element is processed independently in parallel.
- It performs filtering, sorting, and transformation of the input data into a format suitable for further processing.
- The output of the Map function is a set of intermediate key-value pairs.

Reduce Function:

- The Reduce function takes the output generated by the Map function and combines the values associated with the same key.
- It performs aggregation and summarization on the intermediate key-value pairs produced by the Map function.
- The Reduce function typically operates on a per-key basis and merges all the values associated with the same key.
- The output of the Reduce function is usually a smaller set of key-value pairs, representing the final result of the MapReduce operation.

3. What is the main difference between Mapper and Reducer?

Aspect	Mapper	Reducer
Input Data	Processes individual input data elements	Processes intermediate key-value pairs
Operation	Transforms input data into intermediate key-value pairs	Aggregates values based on keys
Execution	Operates in parallel on different portions of data	Executes after the Mapper phase
Output	Produces intermediate key-value pairs	Produces final aggregated or summarized key-value pairs

Aspect	Mapper	Reducer
Task Focus	Data transformation, filtering, sorting	Aggregation, summarization, statistical analysis
Independence	Processes data independently and concurrently	Processes data grouped by keys

4. List the main features of MapReduce.

MapReduce, as a programming model and associated framework, offers several key features that make it well-suited for processing large-scale data efficiently and reliably:

1. **Scalability**: MapReduce is highly scalable, allowing it to efficiently process datasets of virtually any size by distributing the workload across multiple nodes in a cluster. As the size of the dataset or the cluster increases, MapReduce can seamlessly scale to accommodate the additional processing requirements.

2. **Fault Tolerance**: MapReduce provides built-in fault tolerance mechanisms that ensure the integrity and reliability of data processing even in the presence of hardware failures, network issues, or other types of system failures. It achieves fault tolerance through mechanisms such as data replication, task re-execution, and automatic recovery.

3. **Parallel Processing**: MapReduce leverages parallel processing techniques to divide data processing tasks into smaller, independent units that can be executed concurrently across multiple nodes in a distributed environment. This parallelism enables efficient utilization of computing resources and accelerates data processing tasks.

5. **Data Locality Optimization**: MapReduce optimizes data processing performance by minimizing data movement across the network and maximizing data locality. It achieves this optimization by scheduling computation tasks to run on nodes where the input data is already stored, reducing the need for data transfer over the network and improving overall processing efficiency.

6. **Flexibility:** MapReduce is a flexible framework that can be applied to a wide range of data processing tasks, including data transformation, filtering, aggregation, sorting, and more. It supports various input and output formats, allowing developers to process structured, semi-structured, and unstructured data from diverse sources.

7. **Compatibility with Distributed File Systems:** MapReduce integrates seamlessly with distributed file systems such as Hadoop Distributed File System (HDFS), enabling efficient storage and retrieval of large datasets across a cluster of machines. This integration facilitates data processing workflows where data is stored and processed in a distributed manner.

Overall, these features make MapReduce a powerful and versatile framework for processing big data, enabling organizations to extract valuable insights from massive datasets efficiently and reliably.

5. What are the limitations of MapReduce?

MapReduce, while a powerful and widely used framework, also has its limitations:

1. **High Latency**: MapReduce jobs typically involve multiple stages (map, shuffle, reduce) which introduces significant overhead and increases job execution time. As a result, MapReduce may not be suitable for real-time or low-latency processing requirements.

2. **Disk I/O Overhead**: MapReduce relies heavily on disk I/O for storing intermediate data between map and reduce phases. This can lead to performance bottlenecks, especially when dealing with large datasets, as excessive disk I/O can degrade overall system performance and increase job execution time.

3. **Complex Programming Model**: While MapReduce provides a simplified programming model compared to traditional distributed computing frameworks, it still requires developers to understand concepts such as key-value pairs, map and reduce functions, and data partitioning. Writing and debugging MapReduce programs can be challenging, especially for users unfamiliar with distributed systems.

6. Describe the working of the MapReduce algorithm.

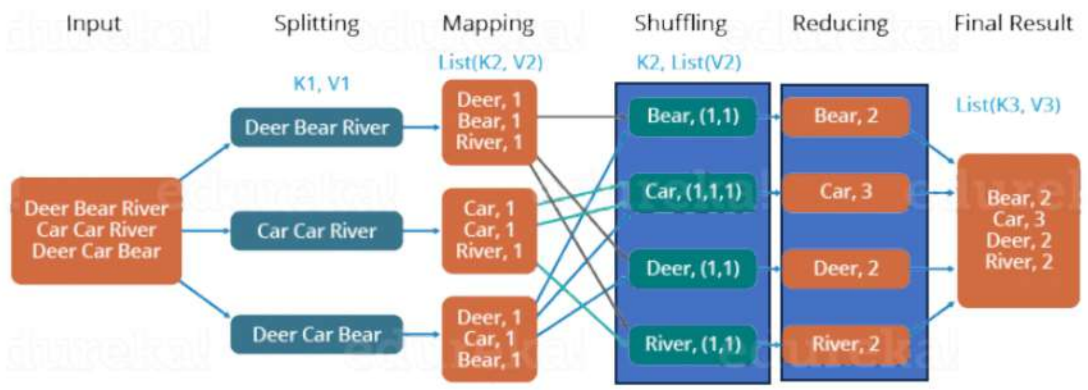
First of all, key-value pairs form the basic data structure in MapReduce. The algorithm receives a set of input key/value pairs and produces a set of key-value pairs as an output. In MapReduce, the designer develops a mapper and a reducer.

Example

Let us understand, how a MapReduce works by taking an example where we have a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding unique words and the number of occurrences of those unique words.



Mapper Phase

- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs — Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.

Shuffling and sorting

- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.

- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.

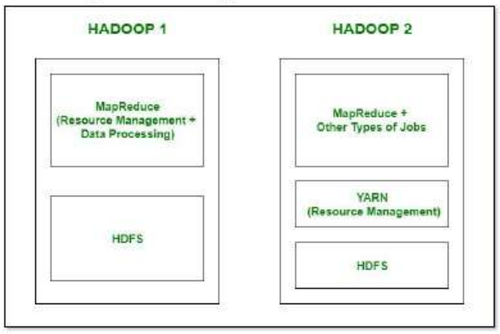
Reduce Phase

- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as — Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

7. Difference between Hadoop 1 and Hadoop 2

Following are the main differences between Hadoop 1 and Hadoop 2.

1. Components: In Hadoop 1 we have MapReduce but Hadoop 2 has YARN(Yet Another Resource Negotiator) and MapReduce version 2.


HADOOP 1
HADOOP 2
MapReduce (Resource Management + Data Processing)
HDFS
MapReduce + Other Types of Jobs
YARN (Resource Management)
HDFS

2. Daemons:

Hadoop 1	Hadoop 2
Namenode	Namenode
Datanode	Datanode
Secondary Namenode	Secondary Namenode
Job Tracker	**Resource Manager**
Task Tracker	**Node Manager**

3. **Working:**

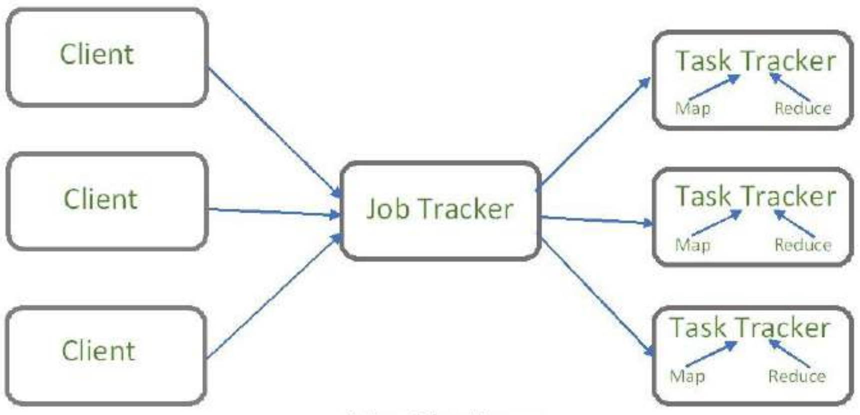
- In Hadoop 1, there is HDFS which is used for storage and top of it, Map Reduce which works as Resource Management as well as Data Processing. Due to this workload on Map Reduce, it will affect the performance.

- In Hadoop 2, there is again HDFS which is again used for storage and on the top of HDFS, there is YARN which works as Resource Management. It basically allocates the resources and keeps all the things going on.

Sr. No.	**Key**	**Hadoop 1**	**Hadoop 2**
1	New Components and API	As Hadoop 1 introduced prior to Hadoop 2 so has some less components and APIs as compare to that of Hadoop 2.	On other hand Hadoop 2 introduced after Hadoop 1 so has more components and APIs as compare to Hadoop 1 such as YARN API,YARN FRAMEWORK, and enhanced Resource Manager.
2	Support	Hadoop 1 only supports MapReduce processing model in its architecture and it does not support non MapReduce tools.	On other hand Hadoop 2 allows to work in MapReducer model as well as other distributed computing models like Spark, Hama, Giraph, Message Passing Interface) MPI & HBase coprocessors.
3	Resource Management	Map reducer in Hadoop 1 is responsible for processing and cluster-resource management.	On other hand in case of Hadoop 2 for cluster resource management YARN is used while processing management is done using different processing models.
4	Scalability	As Hadoop 1 is prior to Hadoop 2 so comparatively less scalable than Hadoop 2 and in context of scaling of nodes it is limited to 4000 nodes per cluster	On other hand Hadoop 2 has better scalability than Hadoop 1 and is scalable up to 10000 nodes per cluster.
5	Implementation	Hadoop 1 is implemented as it follows the concepts of slots which can be used to run a Map task or a Reduce task only.	On other hand Hadoop 2 follows concepts of containers that can be used to run generic tasks.
6	Windows Support	Initially in Hadoop 1 there is no support for Microsoft Windows provided by Apache.	On other hand with an advancement in version of Hadoop Apache provided support for Microsoft windows in Hadoop 2.

8. Explain how JobTracker and TaskTracker function.

JobTracker and TaskTracker are 2 essential process involved in MapReduce execution in MRv1 (or Hadoop version 1). Both processes are now deprecated in MRv2 (or Hadoop version 2) and replaced by Resource Manager, Application Master and Node Manager Daemons.



Hadoop 1.0 architecture

Job Tracker –

1. JobTracker process runs on a separate node and not usually on a DataNode.
2. JobTracker is an essential Daemon for MapReduce execution in MRv1. It is replaced by ResourceManager in MRv2.
3. JobTracker receives the requests for MapReduce execution from the client.
4. JobTracker talks to the NameNode to determine the location of the data.
5. JobTracker finds the best TaskTracker nodes to execute tasks based on the data locality (proximity of the data) and the available slots to execute a task on a given node.
6. JobTracker monitors the individual TaskTrackers and the submits back the overall status of the job back to the client.
7. JobTracker process is critical to the Hadoop cluster in terms of MapReduce execution.
8. When the JobTracker is down, HDFS will still be functional but the MapReduce execution can not be started and the existing MapReduce jobs will be halted.

TaskTracker –

1. TaskTracker runs on DataNode. Mostly on all DataNodes.
2. TaskTracker is replaced by Node Manager in MRv2.
3. Mapper and Reducer tasks are executed on DataNodes administered by TaskTrackers.
4. TaskTrackers will be assigned Mapper and Reducer tasks to execute by JobTracker.
5. TaskTracker will be in constant communication with the JobTracker signalling the progress of the task in execution.
6. TaskTracker failure is not considered fatal. When a TaskTracker becomes unresponsive, JobTracker will assign the task executed by the TaskTracker to another node.