Unit 4: MapReduce

Owner	Saugat Tiwari
Tags	BCA Cloud Computing Notes
Creation Date	@December 29, 2024

Introduction to Parallel Computing

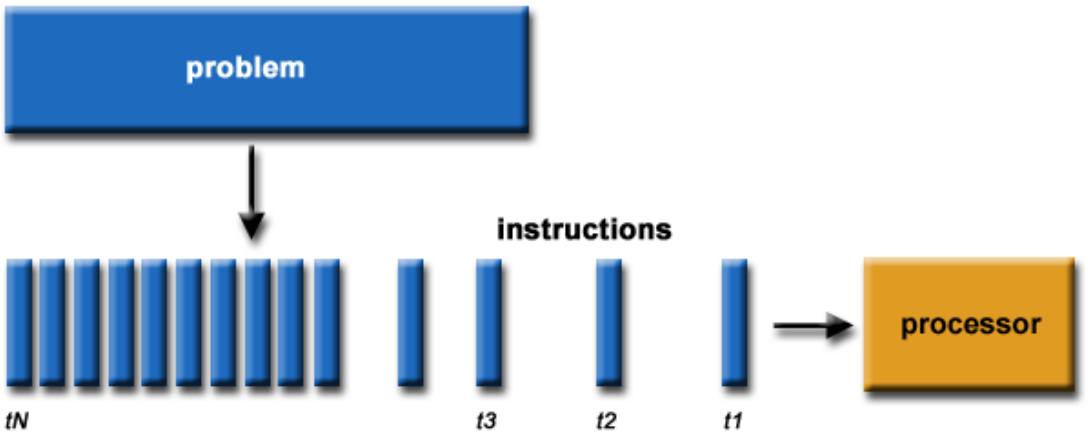
Parallel Computing is a mechanism where two or more process can be executed concurrently on different processors at the same time. In order to handle this overall control/coordination mechanism is employed. The parallel computing will increase the performance of the system i.e. within less time so many processes can be executed simultaneously.

In other word, It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource that has been applied to work is working at the same time.

Parallel computing is a type of computing architecture in which several processors simultaneously execute multiple, smaller calculations broken down from an overall larger, complex problem

Traditionally, software has been written for
serial computation:

- To be run on a single computer having a single Central Processing Unit (CPU);
- A problem is broken into a discrete series of instructions.
- Instructions are executed one after another.
- Only one instruction may execute at any moment in time.

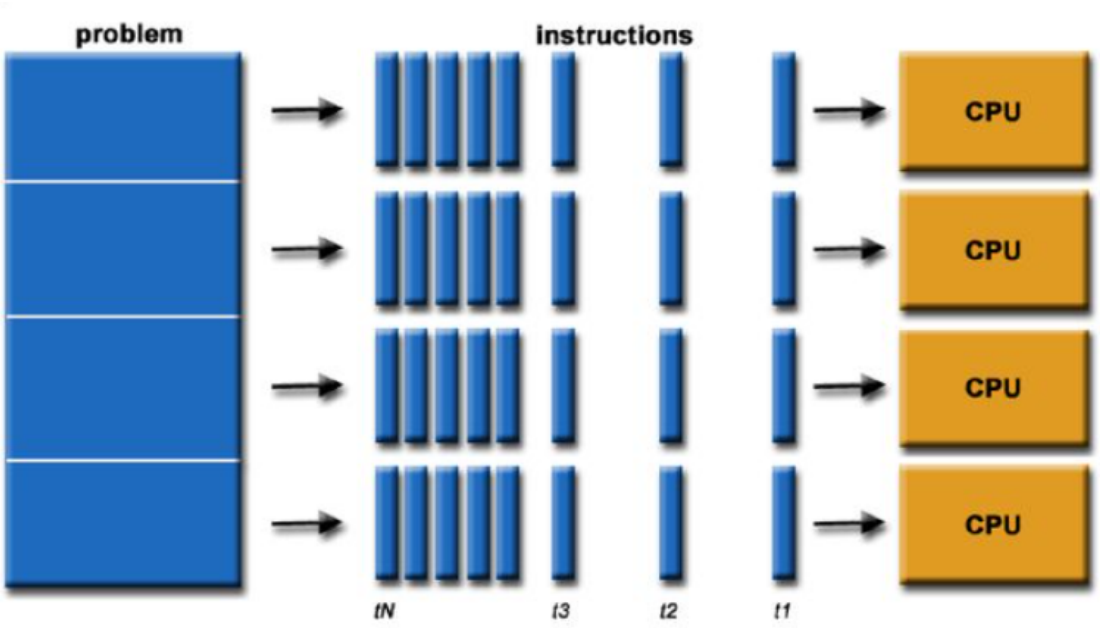


Parallel Computing

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.

- To be run using multiple CPUs
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions

Instructions from each part execute simultaneously on different CPUs



Advantages of Parallel Computing over Serial Computing are as follows:

1. It saves time and money as many resources working together will reduce the time and cut potential costs.
2. It can be impractical to solve larger problems on Serial Computing.
3. It can take advantage of non-local resources when the local resources are finite.
4. Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of the hardware.

Types of Parallelism:

1. **Bit-level parallelism:** It is the form of parallel computing which is based on the increasing processor's size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data.
   For Example: Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.
2. **Instruction-level parallelism:** A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.
3. **Task Parallelism:** Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform the execution of sub-tasks concurrently.
4. **Data-level parallelism (DLP):** Instructions from a single stream operate concurrently on several data – Limited by non-regular data manipulation patterns and by memory bandwidth

Applications of Parallel Computing:

- Databases and Data mining.
- Real-time simulation of systems.
- Science and Engineering.
- Advanced graphics, augmented reality, and virtual reality.

Limitations of Parallel Computing:

- It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.
- The algorithms must be managed in such a way that they can be handled in a parallel mechanism.
- The algorithms or programs must have low coupling and high cohesion. But it's difficult to create such programs.
- More technically skilled and expert programmers can code a parallelism-based program well.

Map Reduce:

Map reduce model MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks.

Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++.

It works on huge volumes of data and enormous scope of computing. This is very useful for performing large-scale data analysis using multiple machines in the cluster.

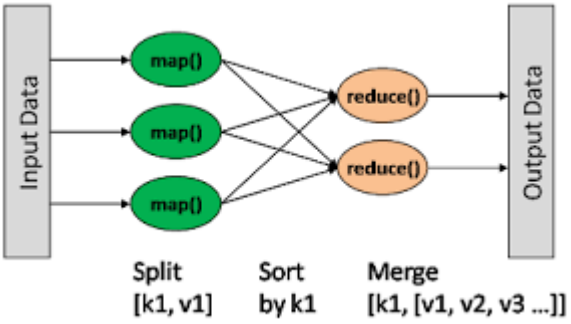
- MapReduce consists of two distinct tasks - Map and Reduce.
- As the name MapReduce suggests, reducer phase takes place after the mapper phase has
  been completed.
- So, the first is the map job, where a block of data is read and processed to produce key-
  value pairs as intermediate outputs.
- The output of a Mapper or map job (key-value pairs) is input to the Reducer.
- The reducer receives the key-value pair from multiple map jobs.
- Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair)
  into a smaller set of tuples or key-value pairs which is the final output.

The MapReduce model is expressed in the form of the two functions, which are defined as follows:

map(k1,v1) → list(k2,v2)

reduce(k2, list(v2)) → list(v2)

MapReduce consists of two phases: Map and Reduce Map generally deals with the
splitting and mapping of data while reducing tasks shuffle and reducing the data.



The whole process of Map reduce goes through four phases of execution:

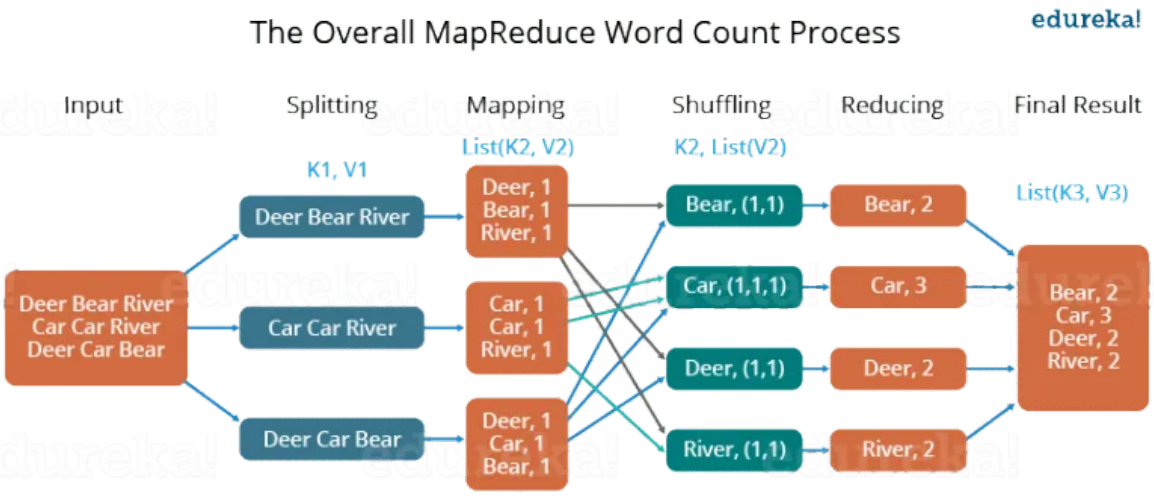
1. **Splitting:** Input divided into fixed-size pieces called input splits.
2. **Mapping:** In this phase data in each split is passed to a mapping function to produce output values.
3. **Shuffling:** The same words are clubbed together along with their respective frequency.
4. **Reducing:** This phase summarizes the complete dataset.

A Word Count Example of MapReduce

Let us understand, how a MapReduce works by taking an example where I 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.



MapReduce Example

- 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.
- 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.
- 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.

Application Of MapReduce

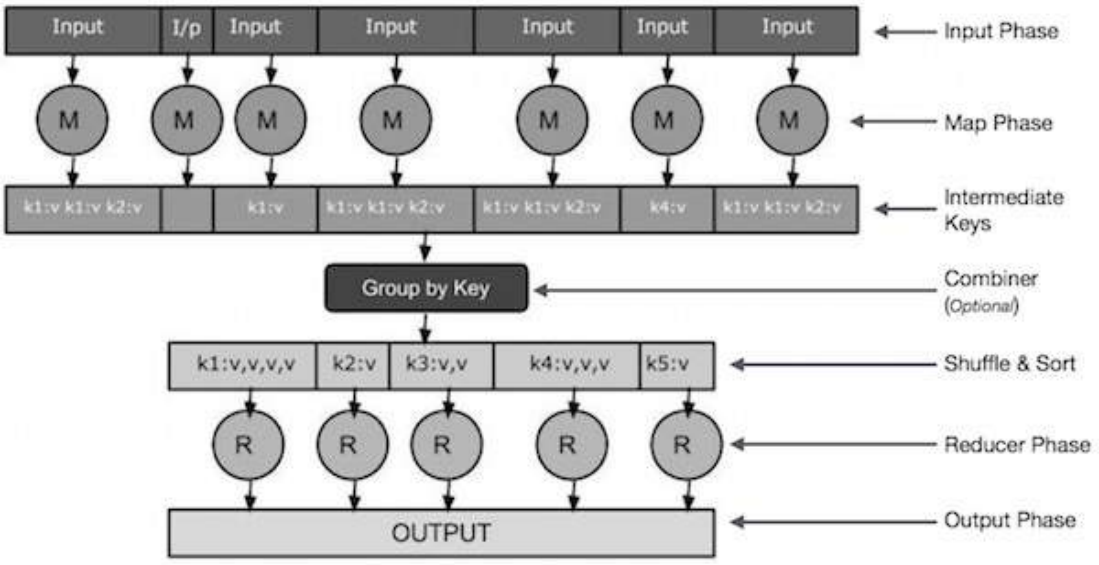
1. **Entertainment**: To discover the most popular movies, based on what you like and what you watched in this case Hadoop MapReduce help you out. It mainly focuses on their logs and clicks.
2. **E-commerce:** Numerous E-commerce suppliers, like Amazon, Walmart, and eBay, utilize the MapReduce programming 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.
3. **Data Warehouse:** We can utilize MapReduce to analyze large data volumes in data warehouses while implementing specific business logic for data insights.
4. **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.

How does MapReduce Works?

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.

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.

Parallel Efficiency of MapReduce

Parallel efficiency in the context of **MapReduce** is a measure of how well the parallelization of a task performs compared to its sequential execution. It is influenced by factors such as data distribution, network overhead, and the time spent on task setup and final aggregation. Here's an explanation with a figure:

Explanation:

1. **Parallel Efficiency:**
    - It is defined as the ratio of speedup achieved by the MapReduce framework to the number of processors (or nodes) used:

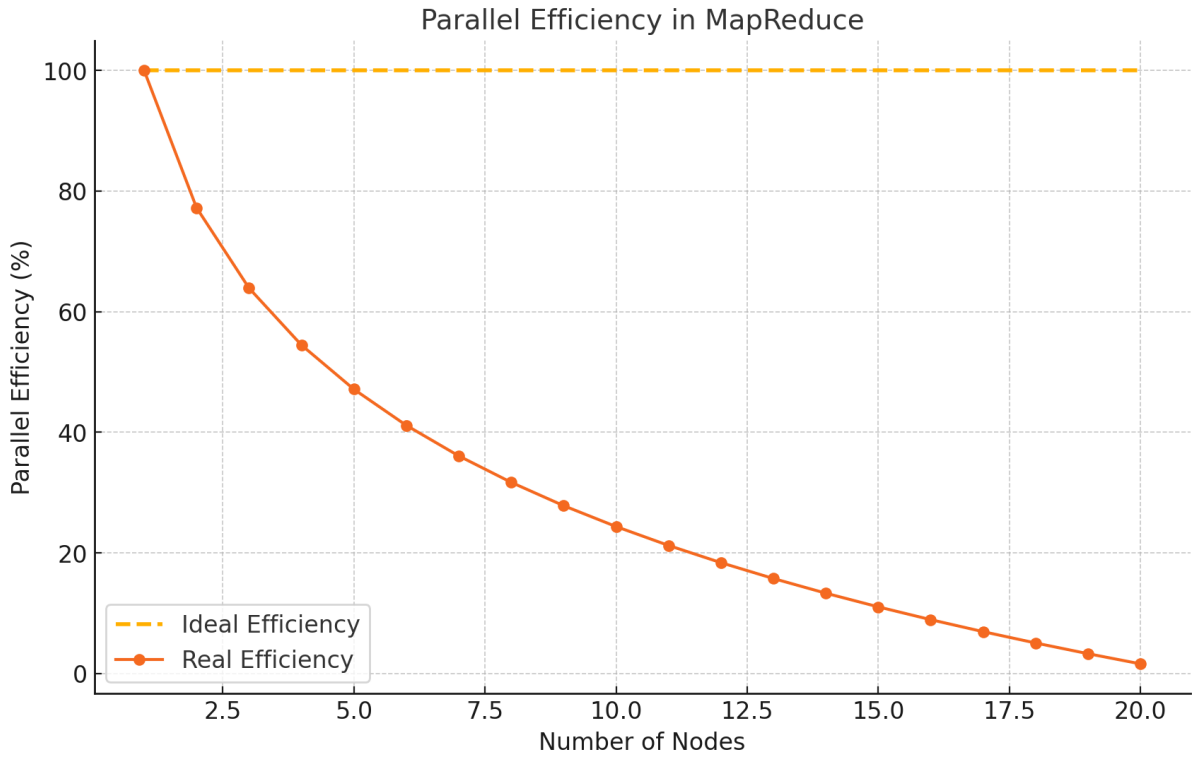
      `Parallel Efficiency = Speedup/ Number of Processors`
    - **Speedup** refers to the improvement in execution time when a task is executed in parallel compared to sequential execution.
2. **Factors Affecting Parallel Efficiency:**

- **Task Splitting:** Uneven splitting of tasks among mappers/reducers can lead to some nodes idling.
- **Communication Overhead:** Data shuffling and inter-node communication reduce efficiency.
- **Fault Tolerance:** Retries or checkpointing mechanisms may impact parallel efficiency.
- **Skewness in Data:** If some tasks take significantly longer than others, it affects parallel efficiency.

Diagram:

The figure below represents how parallel efficiency changes with the number of nodes:

1. The x-axis shows the **number of nodes** used in the MapReduce cluster.
2. The y-axis shows the **parallel efficiency** as a percentage.
3. Two lines indicate:
   - **Ideal Efficiency:** A linear increase without diminishing returns.
   - **Real Efficiency:** The actual observed efficiency, which drops due to overheads and bottlenecks.



The graph above illustrates the **parallel efficiency** of MapReduce:

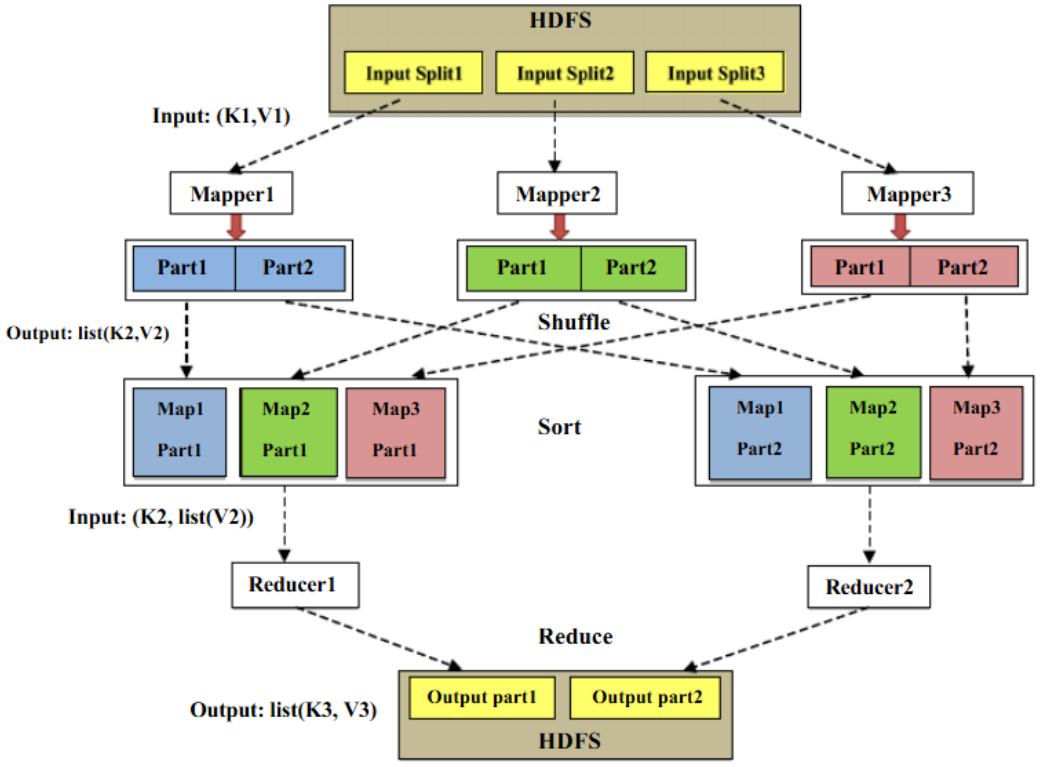
- **Ideal Efficiency (dashed line):** Represents a perfect scenario where adding more nodes leads to 100% efficiency.
- **Real Efficiency (solid line):** Shows diminishing efficiency as more nodes are added due to overhead, communication costs, and other factors.

This highlights the practical limitations of scaling MapReduce and why optimizing task distribution and reducing communication overhead are crucial in real-world implementations.

MapReduce Infrastructure

MapReduce is a programming model and associated infrastructure designed for processing large datasets in a distributed and parallel manner. Its infrastructure comprises various components that work together to execute tasks effectively.

The infrastructure involves **mappers**, **reducers**, and underlying resource management mechanisms.



Components of MapReduce Infrastructure

1. **Input Data**

- The raw data stored in a distributed file system like **HDFS (Hadoop Distributed File System)** is divided into smaller **splits** or **blocks** for processing.

2. **Map Phase**

- The **map function** is applied to each split of the input data.
- The mappers process input data in parallel, converting it into key-value pairs (<key, value>).

3. **Shuffle and Sort**

- Intermediate key-value pairs from the mappers are **shuffled** to ensure all values associated with the same key are grouped together.
- The data is then **sorted** by keys to prepare for reduction.

4. **Reduce Phase**

- The **reduce function** aggregates the key-value pairs from the shuffle phase to produce the final output.

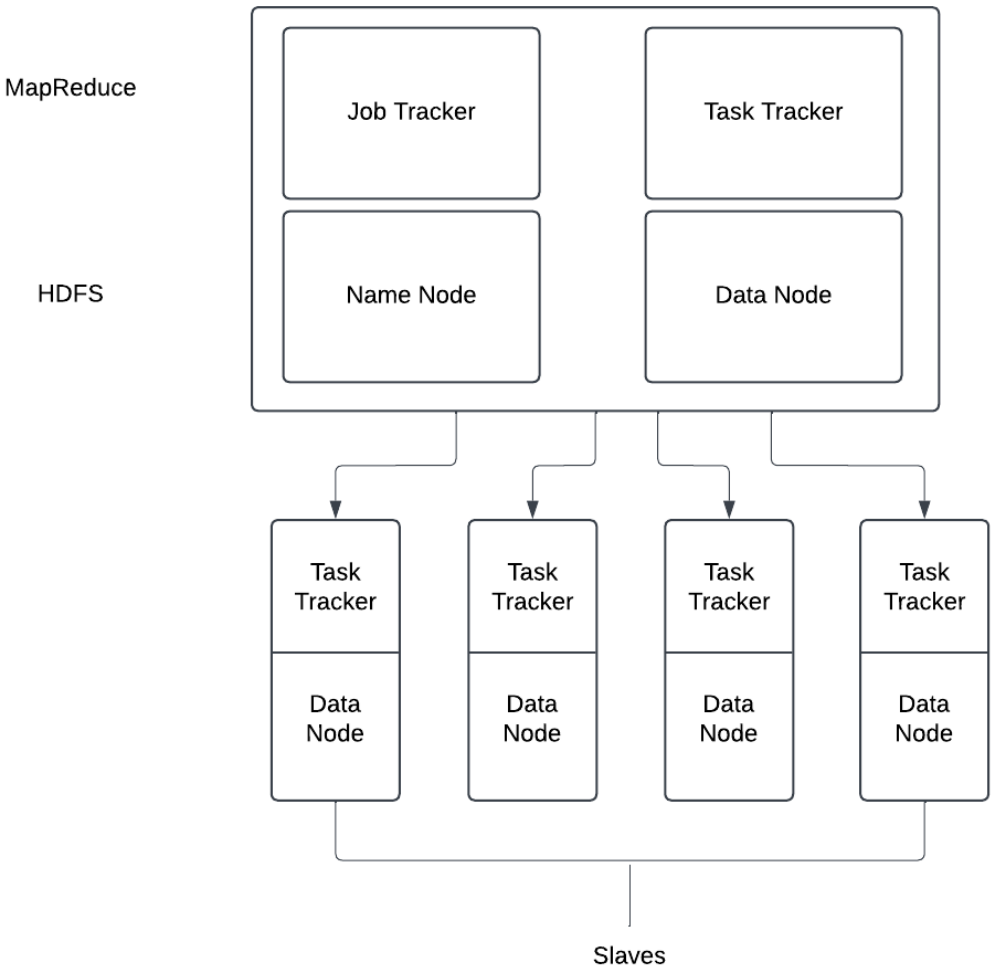
5. **Output Data**

- The results of the reduce phase are written back to a distributed storage system like HDFS.

6. **Job Tracker/Resource Manager**

- Coordinates tasks, monitors execution, and handles failures.
- In modern MapReduce frameworks like Hadoop 2.x, this is managed by **YARN (Yet Another Resource Negotiator).**

MapReduce Architecture:



This diagram represents the **MapReduce architecture** in a Hadoop environment. It illustrates the interaction between the **MapReduce framework** and the **HDFS (Hadoop Distributed File System)**. Here's a breakdown of its components:

Components of the Diagram

1. MapReduce Layer:

- **Job Tracker:**
  - Acts as the **master** in the MapReduce framework.
  - Responsible for:
    - Scheduling tasks across the cluster.
    - Monitoring progress.
    - Handling task failures.

- **Task Tracker:**
  - Runs on each slave node in the cluster.
  - Executes the tasks (map and reduce operations) as instructed by the Job Tracker.
  - Communicates the status of the tasks back to the Job Tracker.

2. HDFS Layer:

- **Name Node:**
  - The master node for HDFS.
  - Responsible for:
    - Managing the filesystem namespace.
    - Keeping metadata about the files (e.g., file location, size, etc.).
    - Coordinating access to files stored in the distributed system.
- **Data Node:**
  - Slave nodes that store the actual data blocks.
  - Communicate with the Name Node to report storage details and handle read/write operations for clients.

3. Slave Nodes (Task Tracker + Data Node):

- The lower portion of the diagram represents slave nodes.
- Each slave node contains:
  - **Task Tracker:** Executes MapReduce tasks (map and reduce operations).
  - **Data Node:** Stores data blocks used during processing.