4 Map Reduce

4.1 Introduction to parallel Computing

- The simultaneous execution of many tasks or processes by utilizing various computing resources, such as multiple processors or computer nodes, to solve a computational problem is referred to as parallel computing. It is a technique for enhancing computation performance and efficiency by splitting a difficult operation into smaller sub-tasks that may be completed concurrently.
- Tasks are broken down into smaller components in parallel computing, with each component running simultaneously on a different computer resource. These resources may consist of separate processing cores in a single computer, a network of computers, or specialized high-performance computing platforms.
- Various Methods to Enable Parallel Computing:
  Different frameworks and programming models have been created to support parallel computing. The design and implementation of parallel algorithms are made easier by these models' abstractions and tools. Programming models that are often utilized include:
  - **Message Passing Interface (MPI):** The Message Passing Interface (MPI) is a popular approach for developing parallel computing systems, particularly in situations with distributed memory. Through message passing, it allows communication as well as collaboration between various processes.
  - **CUDA:** NVIDIA designed CUDA, a platform for parallel computing and a programming language. It gives programmers the ability to use general-purpose parallel computing to its full potential using NVIDIA GPUs.
  - OpenMP: For shared memory parallel programming, OpenMP is a well-liked approach. It enables programmers to define parallel portions in their code, which are then processed by several threads running on various processors.

❖ **Why parallel computing?**

- The whole real-world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.
- Real-world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.
- Parallel computing provides concurrency and saves time and money.
- Complex, large datasets, and their management can be organized only and only using parallel computing's approach.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of the hardware was used and the rest rendered idle.
- Also, it is impractical to implement real-time systems using serial computing.

❖ **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.

❖ **Advantages of Parallel Computing**

- **Cost Efficiency:** Parallel computing can help you save money by utilizing commodity hardware with multiple processors or cores rather than expensive specialized hardware. This makes parallel computing more accessible and cost-effective for a variety of applications.
- **Fault Tolerance:** Systems for parallel computing can frequently be built to be fault-tolerant. The system can continue to function and be reliable even if a processor or core fails because it can continue to be computed on the other processors.
- **Resource Efficiency:** Parallel computing utilizes resources more effectively by dividing the workload among several processors or cores. Parallel computing can maximize resource utilization and minimize idle time instead of relying solely on a single processor, which may remain underutilized for some tasks.
- **Solving Large-scale Problems:** Large-scale problems that cannot be effectively handled on a single machine are best solved using parallel computing. It makes it possible to divide the issue into smaller chunks, distribute those chunks across several processors, and then combine the results to find a solution.
- **Scalability:** By adding more processors or cores, parallel computing systems can increase their computational power. This scalability makes it possible to handle bigger and more complex problems successfully. Parallel computing can offer the resources required to effectively address the problem as its size grows.

❖ **Disadvantages of Parallel Computing**

- **Increased Memory Requirements:** The replication of data across several processors, which occurs frequently in parallel computing, can lead to higher memory requirements. The amount of memory required by large-scale parallel systems to store and manage replicated data may have an impact on the cost and resource usage.
- **Debugging and Testing:** Debugging parallel programs can be more difficult than debugging sequential ones. Race conditions, deadlocks, and improper synchronization problems can be difficult and time-consuming to identify and fix. It is also more difficult to thoroughly test parallel programs to ensure reliability and accuracy.
- **Complexity:** Programming parallel systems as well as developing parallel algorithms can be much more difficult than sequential programming. Data dependencies, load balancing, synchronization, and communication between processors must all be carefully taken into account when using parallel algorithms.

4.1.1 Types of Parallel Computing

- **Bit-level parallelism:**
  - The simultaneous execution of operations on multiple bits or binary digits of a data element is referred to as bit-level parallelism in parallel computing. It is a type of parallelism that uses hardware architectures' parallel processing abilities to operate on multiple bits concurrently.
  - Bit-level parallelism is very effective for operations on binary data such as addition, subtraction, multiplication, and logical operations. The execution time may be considerably decreased by executing these actions on several bits at the same time, resulting in enhanced performance.
  - For example, consider the addition of two binary numbers: 1101 and 1010. As part of sequential processing, the addition would be carried out bit by bit, beginning with the least significant bit (LSB) and moving any carry bits to the following bit. The addition can be carried out concurrently for each pair of related bits when bit-level parallelism is used, taking advantage of the capabilities of parallel processing. Faster execution is possible as a result, and performance is enhanced overall.
  - Specialized hardware elements that can operate on several bits at once, such as parallel adders, multipliers, or logic gates, are frequently used to implement bit-level parallelism. Modern processors may also have SIMD (Single Instruction, Multiple Data) instructions or vector processing units, which allow operations on multiple data components, including multiple bits, to be executed in parallel.

- **Instruction-level parallelism:**
  - ILP, or instruction-level parallelism, is a parallel computing concept that focuses on running several instructions concurrently on a single processor. Instead of relying on numerous processors or computing resources, it seeks to utilize the natural parallelism present in a program at the instruction level.
  - Instructions are carried out consecutively by traditional processors, one after the other. Nevertheless, many programs contain independent instructions that can be carried out concurrently without interfering with one another's output. To increase performance, instruction-level parallelism seeks to recognize and take advantage of these separate instructions.
  - Instruction-level parallelism can be achieved via a variety of methods:
    - **Pipelining:** Pipelining divides the process of executing instructions into several steps, each of which may carry out more than one command at once. This enables the execution of many instructions to overlap while they are in different stages of execution. Each step carries out a distinct task, such as fetching, decoding, executing, and writing back instructions.
    - **Out-of-Order Execution:** According to the availability of input data and execution resources, the processor dynamically rearranges instructions during out-of-order execution. This enhances the utilization of execution units and decreases idle time by enabling independent instructions to be executed out of the order they were originally coded.
- **Task Parallelism**
  - The idea of task parallelism in parallel computing refers to the division of a program or computation into many tasks that can be carried out concurrently. Each task is autonomous and can run on a different processing unit, such as several cores in a multicore CPU or nodes in a distributed computing system.
  - The division of the work into separate tasks rather than the division of the data is the main focus of task parallelism. When conducted concurrently, the jobs can make use of the parallel processing capabilities available and often operate on various subsets of the input data. This strategy is especially helpful when the tasks are autonomous or just loosely dependent on one another.
  - Task parallelism's primary objective is to maximize the use of available computational resources and enhance the program's or computations overall performance. In comparison to sequential execution, the execution time can be greatly decreased by running numerous processes concurrently.
  - Task parallelism can be carried out in various ways few of which are explained below
    - **Thread-based parallelism:** This involves breaking up a single program into several threads of execution. When running simultaneously on various cores or processors, each thread stands for a distinct task. Commonly, shared-memory systems employ thread-based parallelism.
    - **Task-based parallelism:** Tasks are explicitly defined and scheduled for execution in this model. A task scheduler dynamically assigns tasks to available processing resources, taking dependencies and load balance into consideration. Task-based parallelism is a versatile and effective method of expressing parallelism that may be used with other parallel programming paradigms.
    - **Process-based parallelism:** This method involves splitting the program into many processes, each of which represents a separate task. In a distributed computing system, processes can operate on different compute nodes concurrently. In distributed-memory systems, process-based parallelism is often used.
- **Superword-level parallelism**
  - Superword-level parallelism is a parallel computing concept that concentrates on utilizing parallelism at the word or vector level to enhance computation performance. Architectures that enable SIMD (Single Instruction, Multiple Data) or vector operations are particularly suited for their use.
  - Finding and classifying data activities into vector or array operations is the core concept of superword-level parallelism. The parallelism built within the data may be fully utilized by conducting computations on several data pieces in a single instruction.

- Superword-level parallelism is particularly beneficial for applications with predictable data access patterns and easily parallelizable calculations. In applications where a lot of data may be handled concurrently, such as scientific simulations, picture and video processing, signal processing, and data analytics, it is frequently employed.

4.2 MapReduce Model

- The MapReduce model is a programming model and data processing technique designed for large-scale data processing.
- It was popularized by Google and later implemented in open-source frameworks like Apache Hadoop.
- MapReduce simplifies the development of scalable and distributed data processing applications by breaking down the processing into two main phases: the Map phase and the Reduce phase.
- **MapReduce Overview:**

- **Map Phase:**
  - **Mapping Function:** The input data is divided into smaller chunks, and a Map function is applied to each chunk independently. The Map function takes key-value pairs as input and produces a set of intermediate key-value pairs.
  - **Parallel Execution:** Each Map task can be distributed across multiple nodes, allowing for parallel execution. The goal is to process different portions of the input data concurrently, taking advantage of the available computational resources.
  - **Output:** The Map phase produces a set of intermediate key-value pairs, where the keys are typically used to group related data together.
- **Shuffling and Sorting:**
  - **Intermediate Data:** The key-value pairs generated by the Map tasks are shuffled and sorted based on their keys. This step ensures that all values associated with the same key are grouped together.
  - **Data Exchange:** The shuffling and sorting phase involves the transfer of intermediate data between different nodes in the computing environment. This prepares the data for the subsequent Reduce phase.
- **Reduce Phase:**
  - **Reducing Function:** The Reduce function is applied to the sorted and shuffled key-value pairs. The goal is to aggregate and process the data based on the common keys.
  - **Parallel Execution:** Similar to the Map phase, the Reduce phase is designed for parallel execution. Each Reduce task handles a subset of the key-value pairs, and these tasks can be distributed across multiple nodes.
  - **Output:** The output of the Reduce phase provides the final results of the computation. The results are typically aggregated and represent the desired output of the MapReduce job.

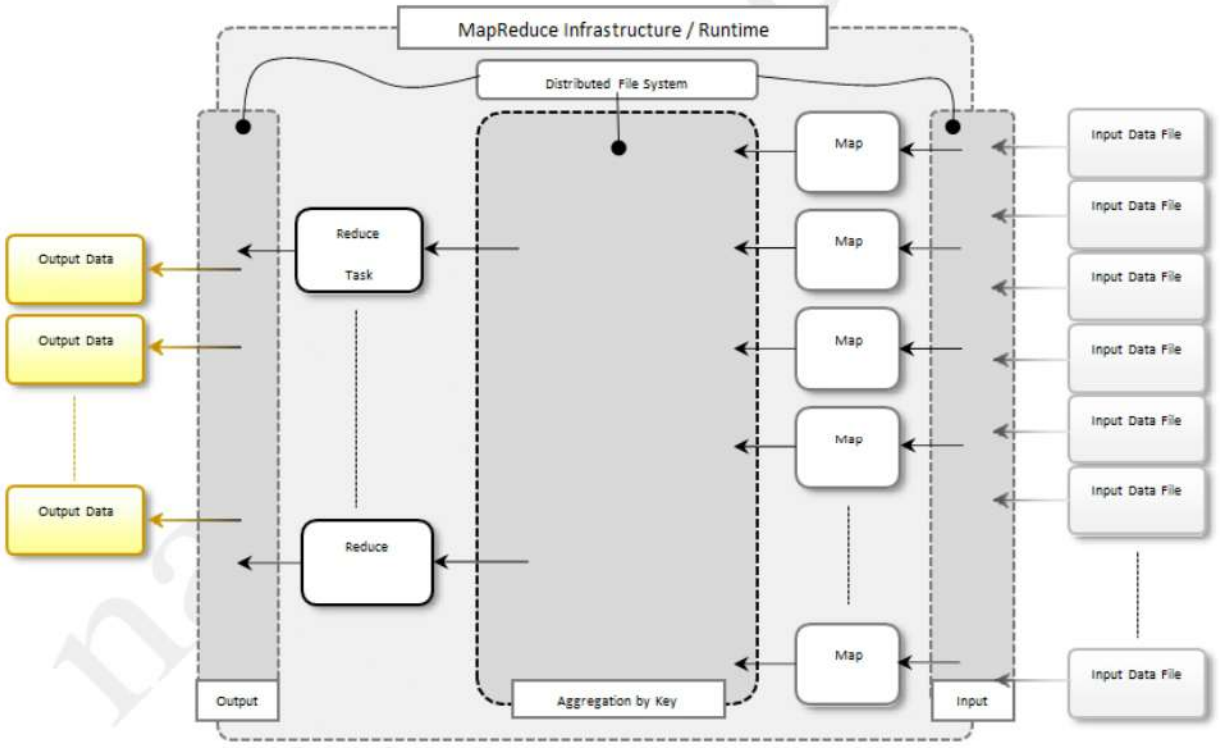
4.3 Map-Reduce Programming

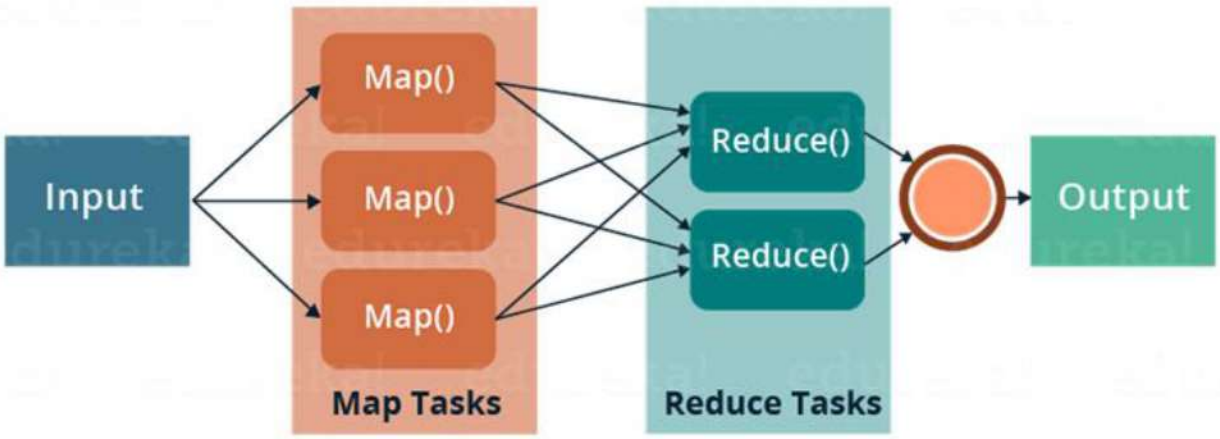
- MapReduce is a programming platform introduced by Google for processing large quantities of data.
- It expresses the computation logic of an application into two simple functions: map and reduce.
- Data transfer and management is completely handled by the distributed storage infrastructure (i.e. the Google File System), which is in charge of providing access to data, replicate files, and eventually move them where needed.
- Therefore, developers do not have to handle anymore these issues and are provided with an interface that presents data at a higher level: as a collection of key-value pairs.
- The computation of MapReduce applications is then organized in a workflow of map and reduce operations that is entirely controlled by the runtime system and developers have only to specify how the map and reduce functions operate on the key value pairs.
- More precisely, the model is expressed in the form of two functions, which are defined as follows:

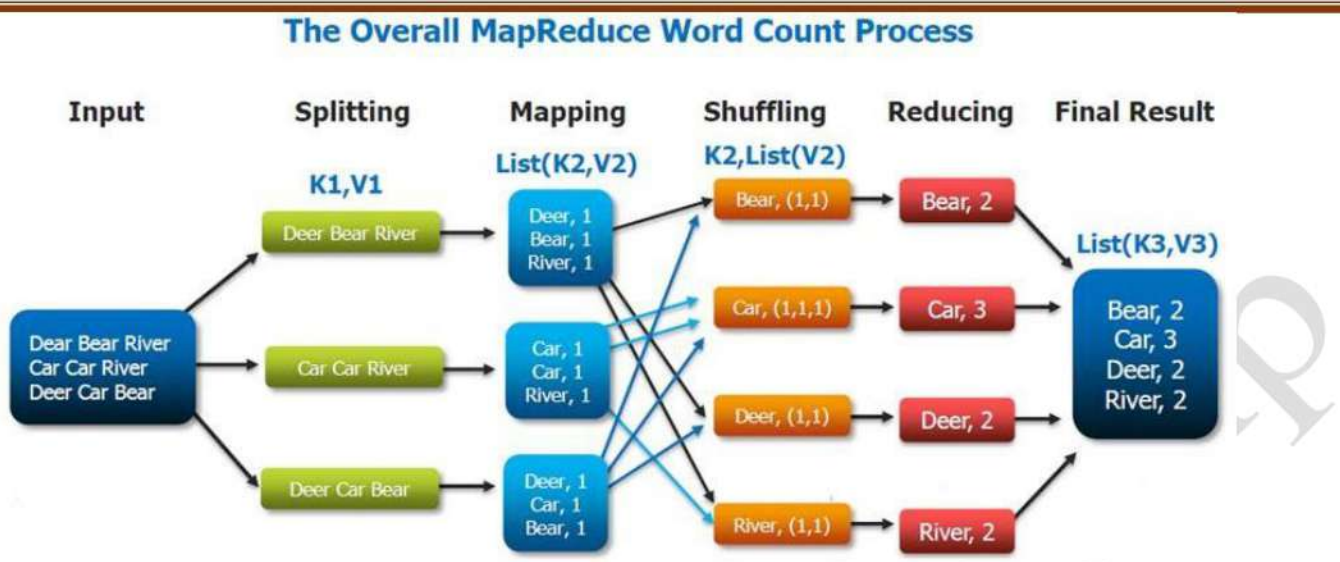
- map (k1,v1) → list(k2,v2)
- reduce(k2,list(v2)) → list(v2)

- The map function reads a key-value pair and produces a list of key-value pairs of different types.
- The reduce function reads a pair composed by a key and a list of values and produces a list of values of the same type.
- The types (k1,v1,k2,kv2) used in the expression of the two functions provide hints on how these two functions are connected and are executed to carry out the computation of a MapReduce job: the output of map tasks is aggregated together by grouping the values according to their corresponding keys and constitute the input of reduce tasks that, for each of the keys found, reduces the list of attached values to a single value.
- Therefore, the input of a MapReduce computation is expressed as a collection of key-value pairs <k1,v1> and the final output is represented by a list values: list(v2).

4.4 MapReduce Computation Workflow







4.5 Benefits of MapReduce

- It is fault-tolerant; during the middle of a map-reduce job, if a machine carrying a few data blocks fails, the architecture handles the failure.
- By splitting the jobs, MapReduce tasks may process several portions of the same dataset concurrently. This has the advantage of completing tasks in less time.
- Multiple copies of same data are delivered to various network nodes. As a result, in the event of a failure, alternative copies are quickly available for processing with no loss.
- Each node sends a status update to the master node regularly. If a slave node fails to transmit its notice, the master node reassigns the slave node's current job to another available node in the cluster.

4.6 Application of MapReduce

- It has found widespread application in cloud computing platforms due to its ability to process massive datasets in parallel across a large number of nodes. Here are the key details about the application of MapReduce in cloud computing:
  - **Distributed Computing Paradigm:**
    - MapReduce is designed to work in a distributed computing environment, which aligns well with the architecture of cloud computing platforms.
    - Cloud computing provides a scalable and elastic infrastructure, allowing MapReduce to efficiently process large datasets by distributing the computation across multiple nodes.
  - **Scalability:**
    - One of the main advantages of MapReduce in cloud computing is its scalability. Cloud platforms allow users to easily scale up or down based on the computational requirements.
    - MapReduce applications can be scaled horizontally by adding more nodes to the cluster, enabling the processing of larger datasets and reducing the overall processing time.
  - **Cost Efficiency:**
    - Cloud computing follows a pay-as-you-go model, allowing users to pay only for the resources they consume.
    - MapReduce applications benefit from this model by utilizing resources on-demand, leading to cost efficiency as users don't need to invest in and maintain a fixed infrastructure.
  - **Fault Tolerance:**
    - Cloud computing environments, including popular platforms like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), offer robust fault-tolerant features.

- MapReduce takes advantage of cloud-based fault tolerance mechanisms to ensure the reliability of distributed processing, even in the presence of node failures.

- **Data Storage Integration:**
  - Cloud platforms provide various storage services, such as Amazon S3, Google Cloud Storage, and Azure Blob Storage, which can seamlessly integrate with MapReduce jobs.
  - MapReduce applications can easily access and process data stored in cloud-based storage services, facilitating efficient data handling.
- **Parallel Processing:**
  - MapReduce breaks down large-scale data processing tasks into smaller sub-tasks, which are processed in parallel across multiple nodes.
  - Cloud computing platforms provide the necessary infrastructure for parallel processing, enabling MapReduce to achieve high performance in handling massive datasets.
- **Flexibility and Customization:**
  - Cloud-based MapReduce implementations offer flexibility in configuring the computing resources according to the specific requirements of the application.
  - Users can customize the number of nodes, memory, and processing power, optimizing the performance of MapReduce jobs.
- **Data Locality:**
  - Cloud platforms often provide the concept of data locality, where data is processed on the same physical node where it resides.
  - MapReduce algorithms can take advantage of data locality, minimizing data transfer overhead and improving overall processing efficiency.
- **Big Data Analytics:**
  - MapReduce is a foundational technology for big data analytics, and cloud computing platforms are well-suited for running analytics workloads on large datasets.
  - Organizations use MapReduce in the cloud for tasks such as log processing, machine learning, and data mining, where the ability to process vast amounts of data quickly is crucial.

4.7 Parallel efficiency of Map-Reduce

The parallel efficiency of MapReduce is a measure of how well a distributed computing system using the MapReduce programming model can utilize its resources when scaling the workload across multiple nodes. Parallel efficiency is an essential metric in evaluating the performance of parallel algorithms, and in the case of MapReduce, it plays a crucial role in assessing how effectively large-scale data processing tasks can be distributed and executed in parallel. Here are key factors influencing the parallel efficiency of MapReduce:

- **Map and Reduce Tasks:**
  - MapReduce divides a processing task into map and reduce phases, which can be executed concurrently on different nodes.
  - The efficiency depends on the ability to balance the workload evenly across all available nodes, ensuring that each node is actively contributing to the overall computation.
- **Data Distribution:**
  - The parallel efficiency of MapReduce is influenced by how well the data is distributed across the nodes. Ideally, data should be evenly distributed to prevent certain nodes from becoming bottlenecks.
  - Skewed data distribution, where a small subset of nodes has significantly more data to process, can lead to reduced parallel efficiency.
- **Communication Overhead:**
  - Efficient communication between nodes is crucial for parallel efficiency. MapReduce relies on shuffling intermediate data between map and reduce tasks, and the efficiency of this data transfer impacts overall performance.

- Minimizing communication overhead, such as by optimizing network communication and reducing unnecessary data transfer, enhances parallel efficiency.

- **Task Scheduling and Coordination:**
  - The scheduling and coordination of map and reduce tasks across nodes play a vital role in parallel efficiency.
  - Efficient task scheduling algorithms ensure that nodes are actively processing data, minimizing idle time and maximizing parallelism.
- **Data Locality:**
  - Data locality refers to the concept of processing data on the same node where it resides. MapReduce frameworks aim to maximize data locality to reduce the need for data transfer between nodes.
  - Utilizing data locality effectively enhances parallel efficiency by minimizing the time spent on data movement.
- **Fault Tolerance Mechanisms:**
  - MapReduce frameworks in cloud computing environments implement fault tolerance mechanisms to handle node failures.
  - The efficiency of these fault tolerance mechanisms impacts parallel efficiency, as they should quickly recover from failures without significantly affecting the overall computation.
- **Scaling:**
  - MapReduce is designed to scale horizontally by adding more nodes to the cluster. The parallel efficiency should ideally improve as the system scales.
  - However, there may be diminishing returns in parallel efficiency as the system scales, and factors like communication overhead become more critical.
- **Resource Utilization:**
  - Parallel efficiency is influenced by how well the computational resources (CPU, memory) are utilized across the nodes.
  - Balancing resource utilization ensures that the entire system is efficiently contributing to the parallel processing task.
- **Algorithm Design:**
  - The design of the MapReduce algorithm itself impacts parallel efficiency. Well-designed algorithms can exploit parallelism effectively, distributing the workload optimally across nodes.
- **Dynamic Workload Balancing:**
  - MapReduce systems that dynamically adjust the workload distribution based on the progress of tasks can improve parallel efficiency.
  - Dynamic workload balancing helps prevent situations where some nodes finish their tasks quickly while others are still processing, reducing idle time.

4.8 Introduction to Hadoop

- Hadoop is an open source software programming framework for storing a large amount of data and performing the computation. Its framework is based on Java programming with some native code in C and shell scripts.
- Hadoop is an open-source software framework that is used for storing and processing large amounts of data in a distributed computing environment. It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets,

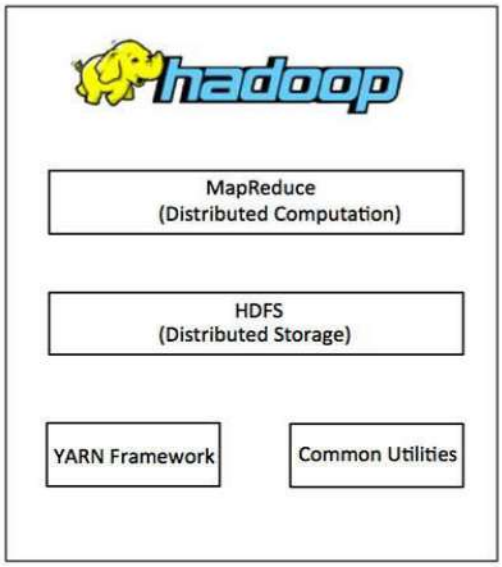
4.8.1 Modules of Hadoop

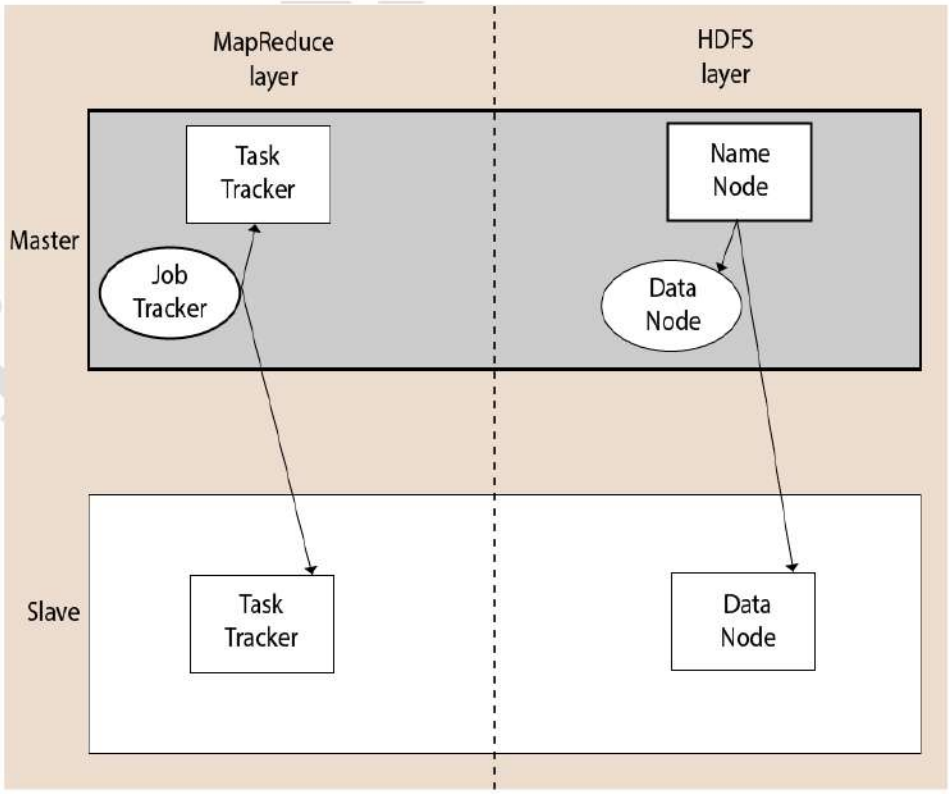
- **HDFS**: Hadoop Distributed File System. Google published its paper GFS and on the basis of that HDFS was developed. It states that the files will be broken into blocks and stored in nodes over the distributed architecture.
- **Yarn**: Yet another Resource Negotiator is used for job scheduling and manages the cluster.

- **Map Reduce**: This is a framework which helps Java programs to do the parallel computation on data using key value pair. The Map task takes input data and converts it into a data set which can be computed in Key value pair. The output of Map task is consumed by reduce task and then the out of reducer gives the desired result.
- **Hadoop Common:** These Java libraries are used to start Hadoop and are used by other Hadoop modules.

4.8.2 Hadoop Architecture

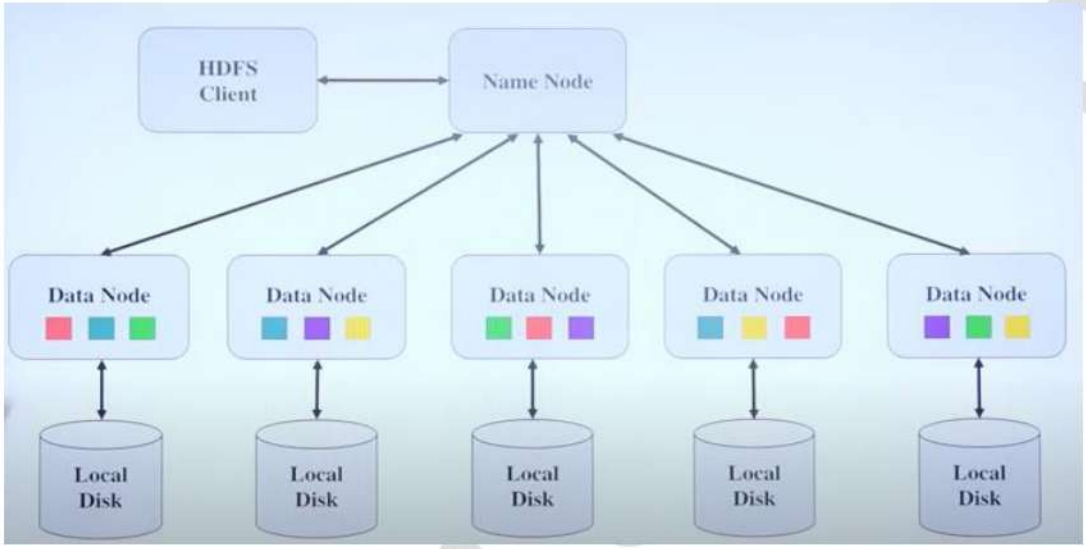
- At its core, Hadoop has two major layers namely –
  - Processing/Computation layer (MapReduce), and
  - Storage layer (Hadoop Distributed File System).





4.8.3 Hadoop Distributed File System

- The Hadoop Distributed File System (HDFS) is a distributed file system for Hadoop. It contains a master/slave architecture. This architecture consist of a single NameNode performs the role of master, and multiple DataNodes performs the role of a slave.
- Both NameNode and DataNode are capable enough to run on commodity machines. The Java language is used to develop HDFS. So any machine that supports Java language can easily run the NameNode and DataNode software.



❖ **NameNode**
- It is a single master server exist in the HDFS cluster.
- As it is a single node, it may become the reason of single point failure.
- It manages the file system namespace by executing an operation like the opening, renaming and closing the files.
- It simplifies the architecture of the system.

❖ **DataNode**
- The HDFS cluster contains multiple DataNodes.
- Each DataNode contains multiple data blocks.
- These data blocks are used to store data.
- It is the responsibility of DataNode to read and write requests from the file system's clients.
- It performs block creation, deletion, and replication upon instruction from the NameNode.

❖ **Job Tracker**
- The role of Job Tracker is to accept the MapReduce jobs from client and process the data by using NameNode.
- In response, NameNode provides metadata to Job Tracker.

❖ **Task Tracker**
- It works as a slave node for Job Tracker.
- It receives task and code from Job Tracker and applies that code on the file. This process can also be called as a Mapper.