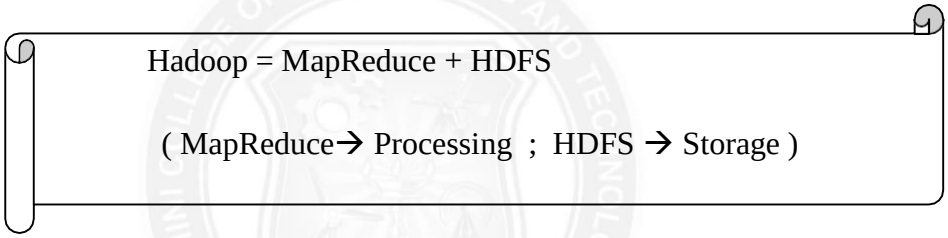
**UNIT V CLOUD TECHNOLOGIES AND ADVANCEMENTS 8**

Hadoop – MapReduce – Virtual Box -- Google App Engine – Programming Environment for Google App Engine — Open Stack – Federation in the Cloud – Four Levels of Federation – Federated Services and Applications – Future of Federation

## Introduction to Hadoop Framework

- Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models.
- Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.
- Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes.




Users of Hadoop:

- Hadoop is running search on some of the Internet's largest sites:
  - Amazon Web Services: Elastic MapReduce
  - AOL: Variety of uses, e.g., behavioral analysis & targeting
  - Ebay: Search optimization (532-node cluster)
  - Facebook: Reporting/analytics, machine learning (1100 m.)
  - LinkedIn: People You May Know (2x50 machines)
  - Twitter: Store + process tweets, log files, other data Yahoo: >36,000 nodes; biggest cluster is 4,000 nodes

Hadoop Architecture

- Hadoop has a Master Slave Architecture for both Storage & Processing
- Hadoop framework includes following four modules:
- Hadoop Common: These are Java libraries and provide file system and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

- ❖ Hadoop YARN: This is a framework for job scheduling and cluster resource management.
- ❖ Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
- ❖ HadoopMapReduce: This is system for parallel processing of large data sets.

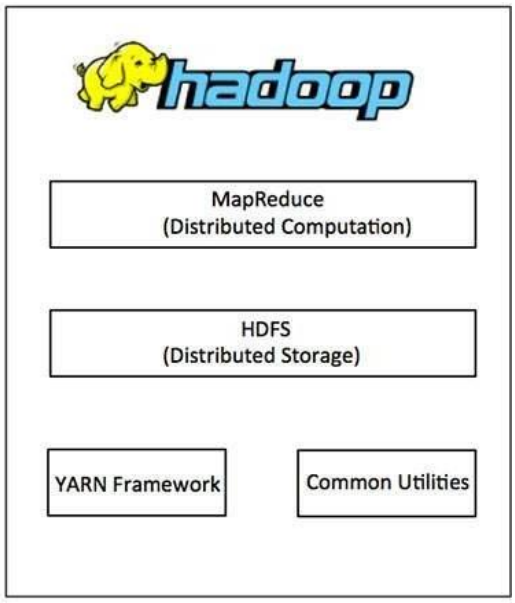



Figure 5.1 Hadoop Architecture

- ☐ The Hadoop core is divided into two fundamental layers:
  - MapReduce engine
  - HDFS
- ☐ The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.
- ☐ HDFS: HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.
- ☐ HDFS Architecture: HDFS has a master/slave architecture containing a single Name Node as the master and a number of Data Nodes as workers (slaves).

**HDFS**

- ☐ To store a file in this architecture,
- ☐ HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (Data Nodes).

- The mapping of blocks to Data Nodes is determined by the Name Node.
- The NameNode (master) also manages the file system's metadata and namespace.
- Namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall management of all files.
- NameNode in the metadata stores all information regarding the location of input splits/blocks in all DataNodes.
- Each DataNode, usually one per node in a cluster, manages the storage attached to the node.
- Each DataNode is responsible for storing and retrieving its file blocks

**HDFS- Features**

Distributed file systems have special requirements

- Performance
- Scalability
- Concurrency Control
- Fault Tolerance
- Security Requirements

**HDFS Fault Tolerance**

**Block replication:**

- To reliably store data in HDFS, file blocks are replicated in this system.
- HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster.
- The replication factor is set by the user and is three by default.
- Replica placement: The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS.
- Storing replicas on different nodes (DataNodes) located in different racks across the whole cluster.
- HDFS stores one replica in the same node the original data is stored.
- One replica on a different node but in the same rack
- One replica on a different node in a different rack.
- Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster.

- Receipt of a Heartbeat implies that the DataNode is functioning properly.
- Each Blockreport contains a list of all blocks on a DataNode .
- The NameNode receives such messages because it is the sole decision maker of all replicas in the system.

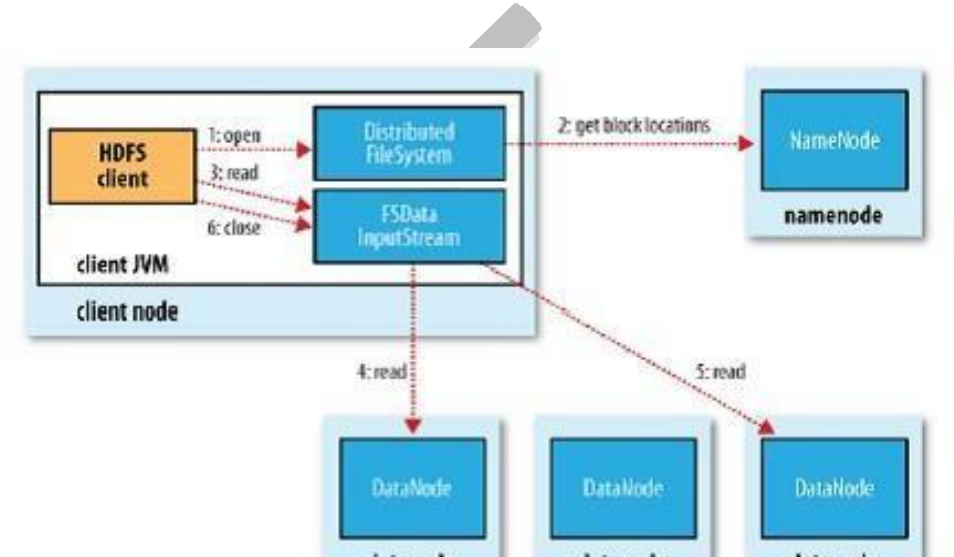
**HDFS High Throughput**

- Applications run on HDFS typically have large data sets.
- Individual files are broken into large blocks to allow HDFS to decrease the amount of metadata storage required per file.
- The list of blocks per file will shrink as the size of individual blocks increases.
- By keeping large amounts of data sequentially within a block, HDFS provides faststreaming reads of data.

**HDFS- Read Operation**

**Reading a file :**

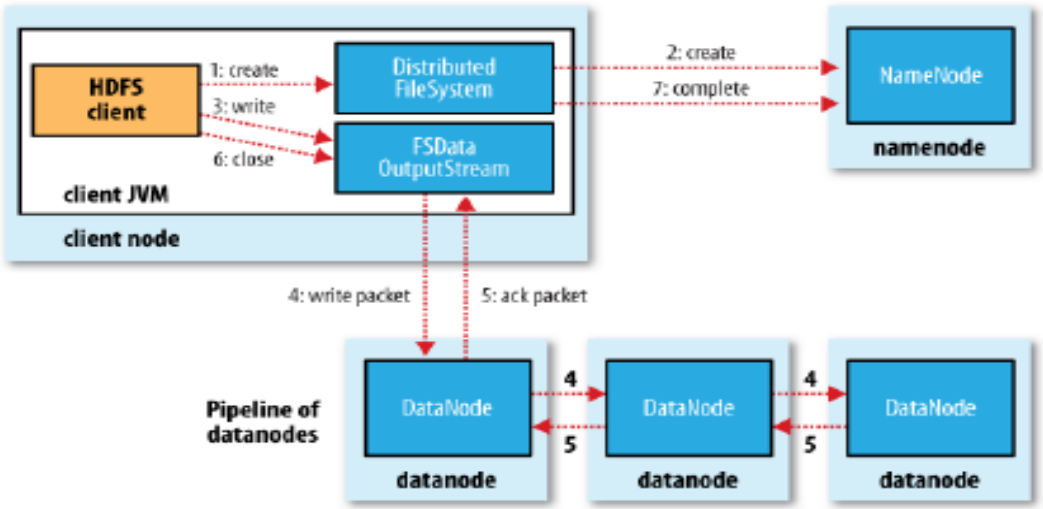
- To read a file in HDFS, a user sends an "open" request to the NameNode to get the location of file blocks.
- For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file.
- The number of addresses depends on the number of block replicas.
- The user calls the read function to connect to the closest DataNode containing the first block of the file.
- Then the first block is streamed from the respective DataNode to the user.
- The established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.



## HDFS-Write Operation

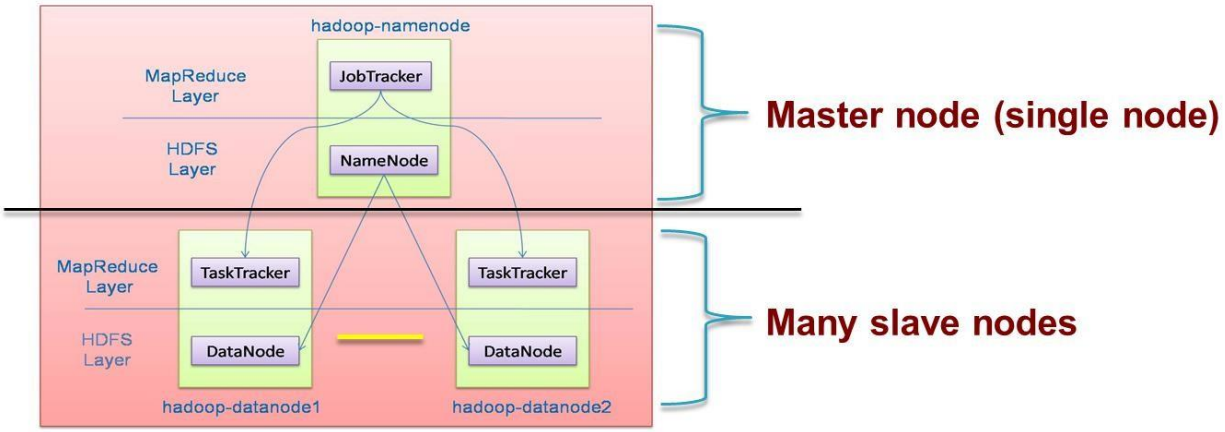
**Writing to a file:**

- To write a file in HDFS, a user sends a "create" request to the NameNode to create a new file in the file system namespace.
- If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the write function.
- The first block of the file is written to an internal queue termed the data queue.
- A data streamer monitors its writing into a DataNode.
- Each file block needs to be replicated by a predefined factor.
- The data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.
- The steamer then stores the block in the first allocated DataNode.
- Afterward, the block is forwarded to the second DataNode by the first DataNode.
- The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode.
- Once this replication process is finalized, the same process starts for the second block.



## 5.1.1 Architecture of Mapreduce in Hadoop

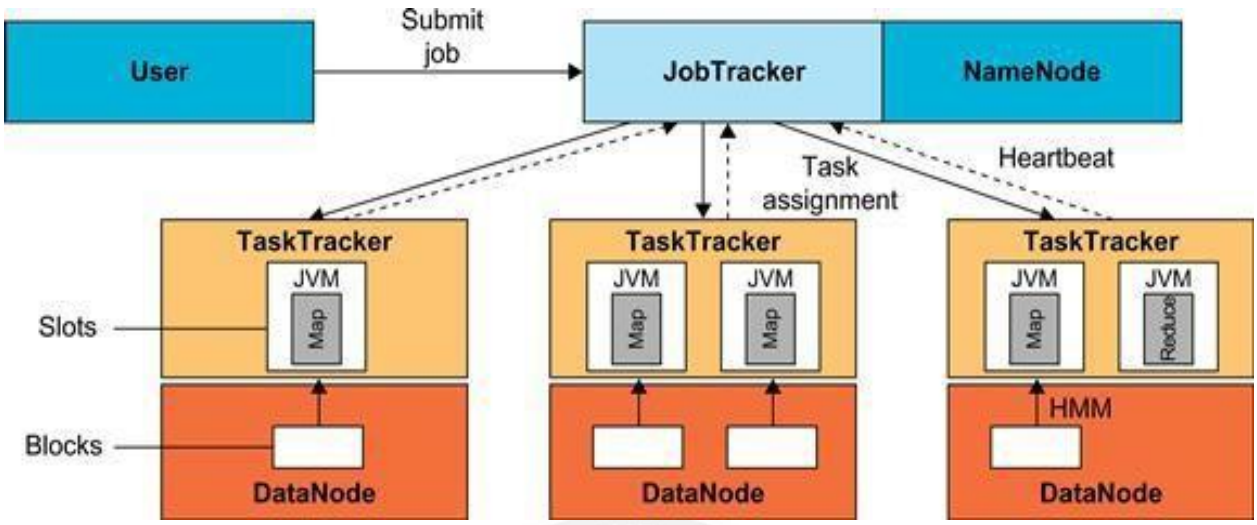
- Distributed file system (HDFS)
- Execution engine (MapReduce)




### Properties of Hadoop Engine

- HDFS has a master/slave architecture containing
- A single NameNode as the master and
- A number of DataNodes as workers (slaves).
- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).
- The NameNode (master) also manages the file system's metadata and namespace.
- Job Tracker is the master node (runs with the namenode)
    - Receives the user's job
    - Decides on how many tasks will run (number of mappers)
    - Decides on where to run each mapper (concept of locality)
- Task Tracker is the slave node (runs on each datanode)
    - Receives the task from Job Tracker
    - Runs the task until completion (either map or reduce task)
    - Always in communication with the Job Tracker reporting progress (heartbeats)

**Running a Job in Hadoop**




- Three components contribute in running a job in this system:
  - a user node,
  - a JobTracker, and
  - severalTaskTrackers.
- The data flow starts by calling the runJob(conf) function inside a user program running on the user node, in which conf is an object containing some tuning parameters for the MapReduce
- Job Submission: Each job is submitted from a user node to the JobTracker node.
- Task assignment : The JobTracker creates one map task for each computed input split
- Task execution : The control flow to execute a task (either map or reduce) starts inside the TaskTracker by copying the job JAR file to its file system.
- Task running check : A task running check is performed by receiving periodic heartbeat messages to the JobTracker from the TaskTrackers.
- Heartbeat:notifies the JobTracker that the sending TaskTracker is alive, and whether the sending TaskTracker is ready to run a new task.

The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing, including:

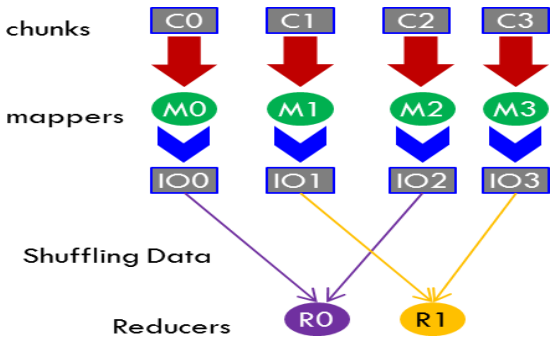
- HadoopCore, our flagship sub-project, provides a distributed filesystem (HDFS) and support for the MapReduce distributed computing metaphor.
- HBase builds on Hadoop Core to provide a scalable, distributed database.
- Pig is a high-level data-flow language and execution framework for parallel

computation.It is built on top of Hadoop Core.

- ZooKeeperis a highly available and reliable coordination system. Distributed applications use ZooKeeper to store and mediate updates for critical shared state.
- Hive is a data warehouse infrastructure built on Hadoop Core that provides data summarization, adhoc querying and analysis of datasets.

MAP REDUCE

- MapReduce is a programming model for data processing.
- MapReduce is designed to efficiently process large volumes of data by connecting many commodity computers together to work in parallel
- Hadoop can run MapReduce programs written in various languages like Java, Ruby, and Python
- MapReduce works by breaking the processing into two phases:
  - The map phase and
  - The reduce phase.
- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.
- The programmer also specifies two functions:
  - The map function and
  - The reduce function.
- In MapReduce, chunks are processed in isolation by tasks called Mappers
- The outputs from the mappers are denoted as intermediate outputs (IOs) and are brought into a second set of tasks called Reducers
- The process of bringing together IOs into a set of Reducers is known as shuffling process
- The Reducers produce the final outputs (FOs)



- Overall, MapReduce breaks the data flow into two phases, map phase and reduce phase

Mapreduce Workflow

Application writer specifies

- A pair of functions called Mapper and Reducer and a set of input files and submits the job
- Input phase generates a number of FileSplits from input files (one per Map task)
- The Map phase executes a user function to transform input key-pairs into a new set of key-pairs
- The framework Sorts & Shuffles the key-pairs to output nodes
- The Reduce phase combines all key-pairs with the same key into new keypairs
- The output phase writes the resulting pairs to files as "parts"

Characteristics of MapReduce is characterized by:

- Its simplified programming model which allows the user to quickly write and test distributed systems
- Its efficient and automatic distribution of data and workload across machines
- Its flat scalability curve. Specifically, after a Mapreduce program is written and functioning on 10 nodes, very little-if any- work is required for making that same program run on 1000 nodes

The core concept of MapReduce in Hadoop is that input may be split into logical chunks, and each chunk may be initially processed independently, by a map task. The results ofthese individual processing chunks can be physically partitioned into distinct sets, which are then sorted. Each sorted chunk is passed to a reduce task.

A map task may run on any compute node in the cluster, and multiple map tasks may berunning in parallel across the cluster. The map task is responsible for transforming the input records into key/value pairs. The output of all of the maps will be partitioned, and each partition will be sorted. There will be one partition for each reduce task. Each partition's sorted keys and thevalues associated with the keys are then processed by the reduce task. There may be multiple reduce tasks running in parallel on the cluster.

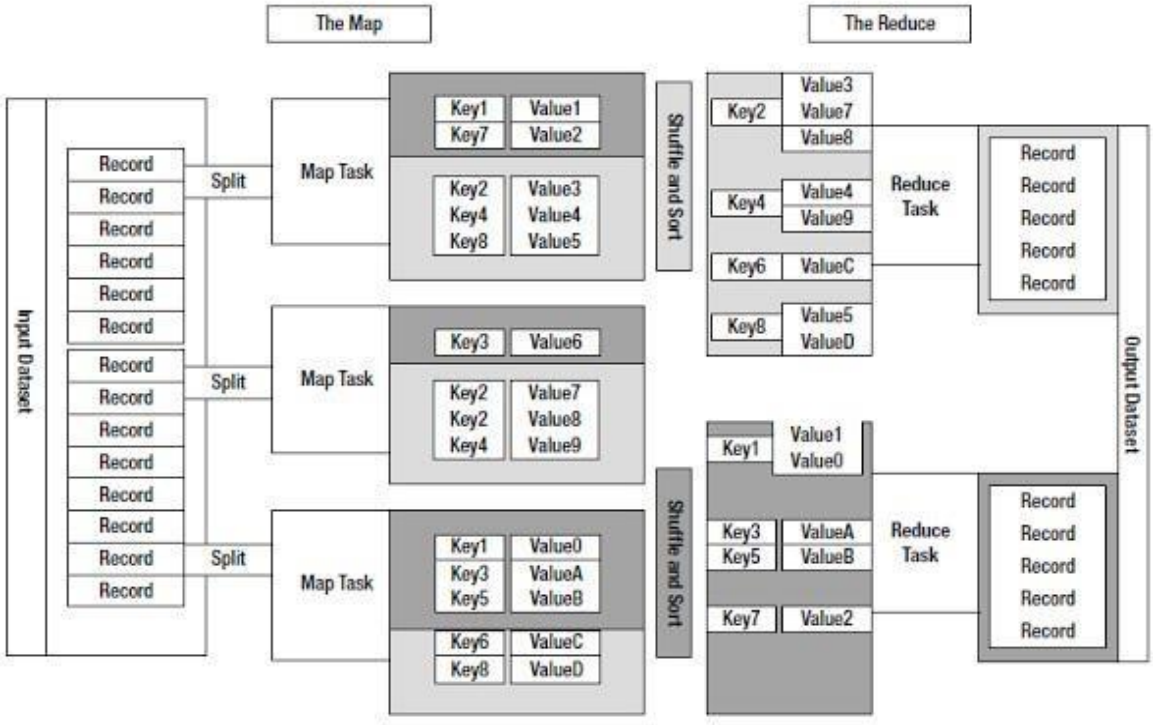
The application developer needs to provide only four items to the Hadoop framework: the class that will read the input records and transform them into one key/value pair per record, a map method, a reduce method, and a class that will transform the key/value pairs that the reduce method outputs into output records.

My first MapReduce application was a specialized web crawler. This crawler received

as input large sets of media URLs that were to have their content fetched and processed. The media items were large, and fetching them had a significant cost in time and resources.

The job had several steps:

1. Ingest the URLs and their associated metadata.
2. Normalize the URLs.
3. Eliminate duplicate URLs.
4. Filter the URLs against a set of exclusion and inclusion filters.
5. Filter the URLs against a do not fetch list.
6. Filter the URLs against a recently seen set.
7. Fetch the URLs.
8. Fingerprint the content items.
9. Update the recently seen set.
10. Prepare the work list for the next application.




Map Reduce Example:

Input File:

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad