19CSE357- Big Data Analytics(3-0-0-3)

Lecture #24

HADOOP

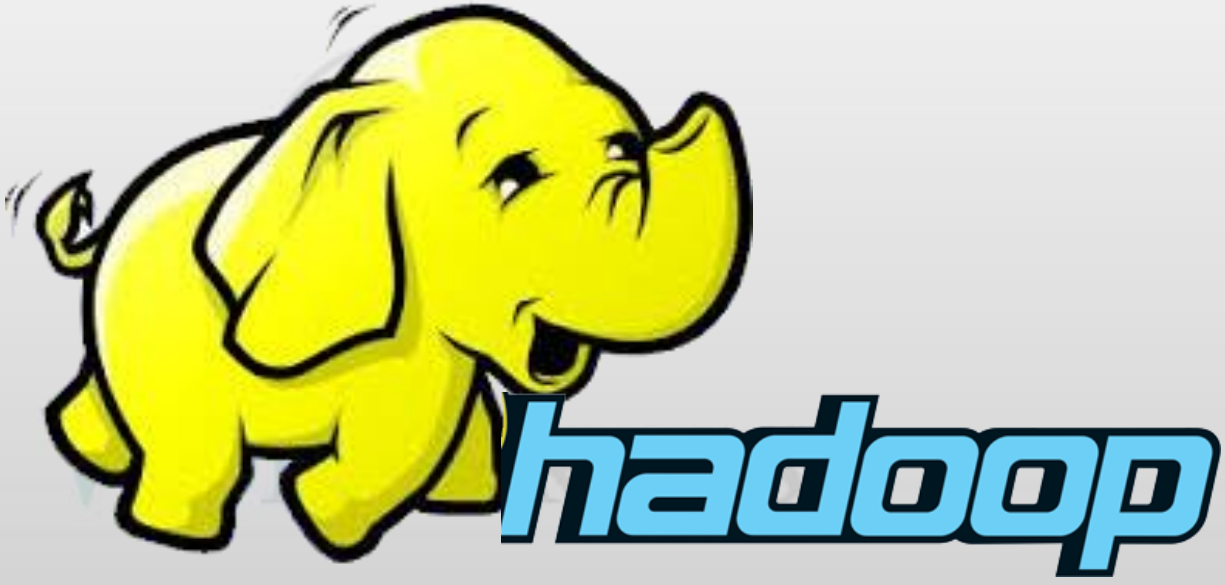
Chapter 5

Sangita Khare

Department of Computer Science & Engineering

Hadoop: A Software Framework for Data Intensive Computing Applications

Hadoop, a distributed framework for Big Data

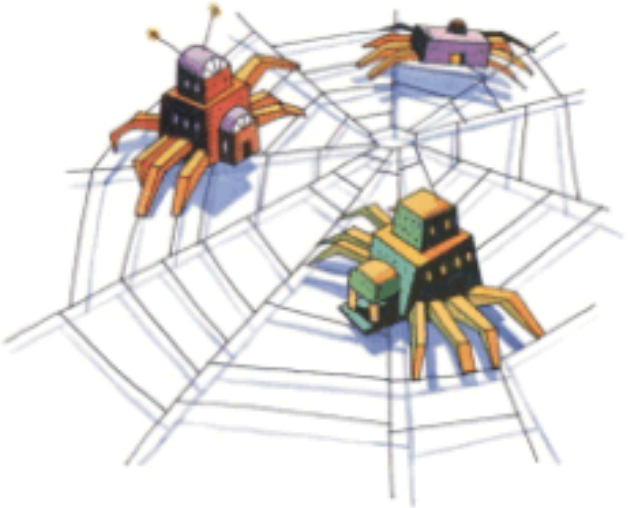


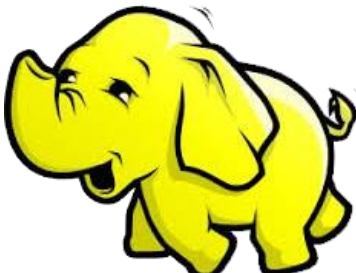
What is Hadoop?

- Apache's top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.
- It is a flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.

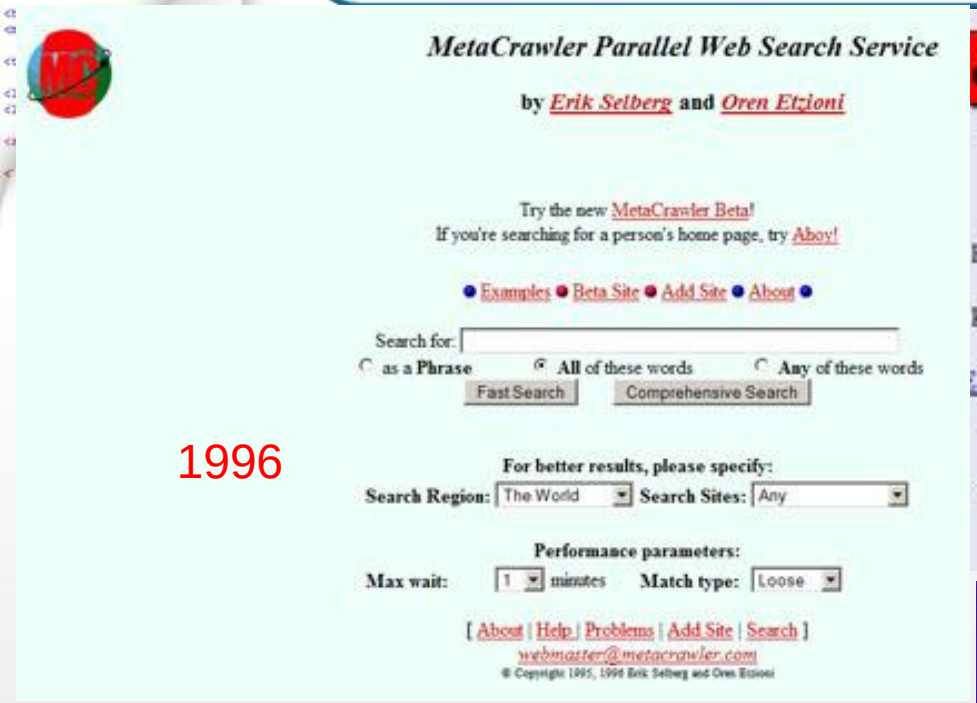
Brief History of Hadoop

Designed to answer the question: **“How to process big data with reasonable cost and time?”**





Search engines in 1990s



1996

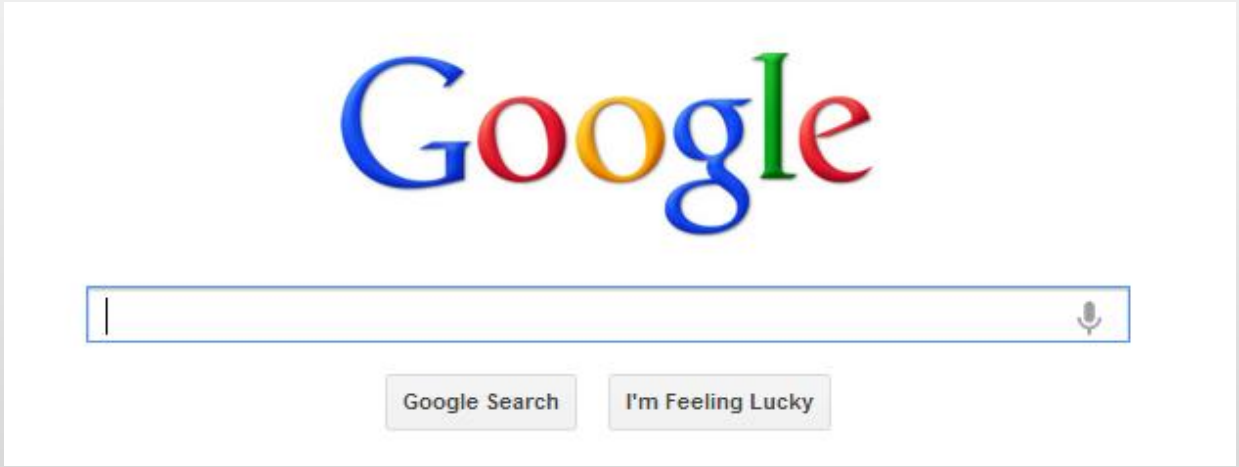
1996

1996

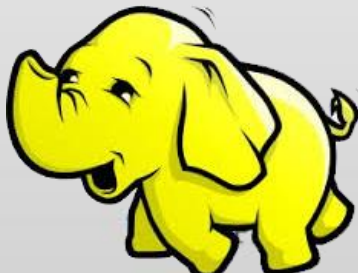
1997

Google search engines

1998



2013



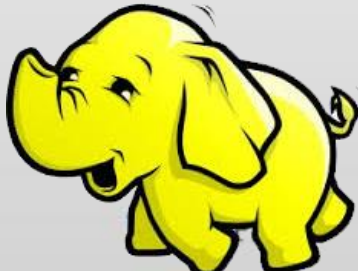
Hadoop's Developers

Doug Cutting

2005: Doug Cutting and Michael J. Cafarella developed Hadoop to support distribution for the Nutch search engine project.

The project was funded by Yahoo.

2006: Yahoo gave the project to Apache Software Foundation.



Google Origins

2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com
Google, Inc.

2006

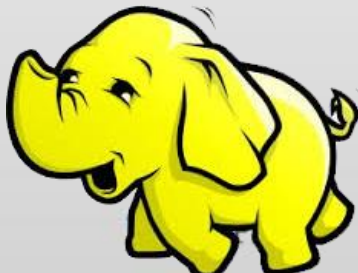
Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

Abstract

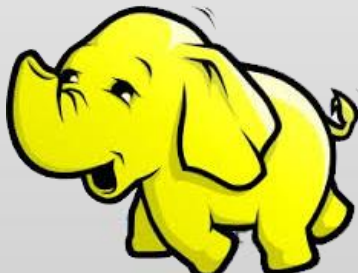
gtable is a distributed storage system for managing tured data that is designed to scale to a very large petabytes of data across thousands of commodity rs. Many projects at Google store data in Bigtable, ding web indexing, Google Earth, and Google Fi- e. These applications place very different demands igtable, both in terms of data size (from URLs to

achieved scalability and high performance, but Big provides a different interface than such systems. Big does not support a full relational data model; inste provides clients with a simple data model that sup dynamic control over data layout and format, an lows clients to reason about the locality properties o data represented in the underlying storage. Data i dexed using row and column names that can be arbi strings. Bigtable also treats data as uninterpreted str

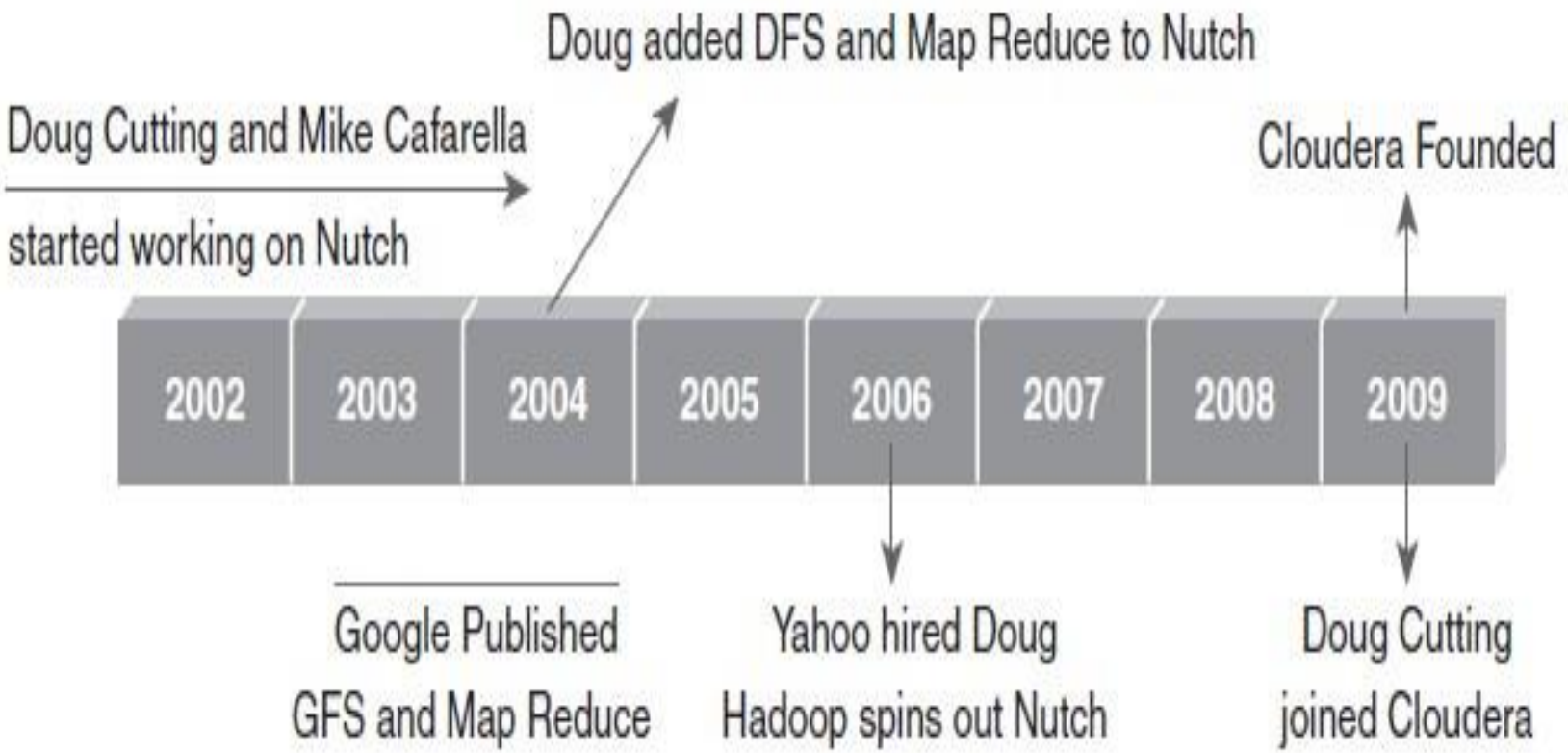


Some Hadoop Milestones

- **2008 - Hadoop Wins Terabyte Sort Benchmark (**sorted 1 terabyte of data in 209 seconds, compared to previous record of 297 seconds)
- 2009 - Avro and Chukwa became new members of Hadoop Framework family
- 2010 - Hadoop's Hbase, Hive and Pig subprojects completed, adding more computational power to Hadoop framework
- **2011 - ZooKeeper Completed**
- **2013 - Hadoop 1.1.2 and Hadoop 2.0.3 alpha.**
  - Ambari, Cassandra, Mahout have been added



History of Hadoop



Apache Hadoop Wins Terabyte Sort Benchmark (July 2008)

- One of Yahoo's Hadoop clusters sorted 1 terabyte of data in **209 seconds**, which beat the previous record of 297 seconds in the annual general purpose (daytona) terabyte sort benchmark. The sort benchmark specifies the input data (10 billion 100 byte records), which must be completely sorted and written to disk.
- The sort used 1800 maps and 1800 reduces and allocated enough memory to buffers to hold the intermediate data in memory.
- The cluster had 910 nodes; 2 quad core Xeons @ 2.0ghz per node; 4 SATA disks per node; 8G RAM per a node; 1 gigabit ethernet on each node; 40 nodes per a rack; 8 gigabit ethernet uplinks from each rack to the core; Red Hat Enterprise Linux Server Release 5.1 (kernel 2.6.18); Sun Java JDK 1.6.0_05-b13

Example Applications and Organizations using Hadoop

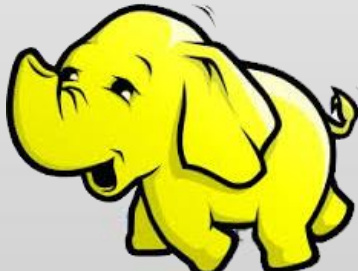
- A9.com – Amazon: To build Amazon's product search indices; process millions of sessions daily for analytics, using both the Java and streaming APIs; clusters vary from 1 to 100 nodes.
- Yahoo! : More than 100,000 CPUs in ~20,000 computers running Hadoop; biggest cluster: 2000 nodes (2*4cpu boxes with 4TB disk each); used to support research for Ad Systems and Web Search
- AOL : Used for a variety of things ranging from statistics generation to running advanced algorithms for doing behavioral analysis and targeting; cluster size is 50 machines, Intel Xeon, dual processors, dual core, each with 16GB Ram and 800 GB hard-disk giving us a total of 37 TB HDFS capacity.
- Facebook: To store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning; 320 machine cluster with 2,560 cores and about 1.3 PB raw storage;
- FOX Interactive Media : 3 X 20 machine cluster (8 cores/machine, 2TB/machine storage) ; 10 machine cluster (8 cores/machine, 1TB/machine storage); Used for log analysis, data mining and machine learning
- University of Nebraska Lincoln: one medium-sized Hadoop cluster (200TB) to store and serve physics data;

More Hadoop Applications

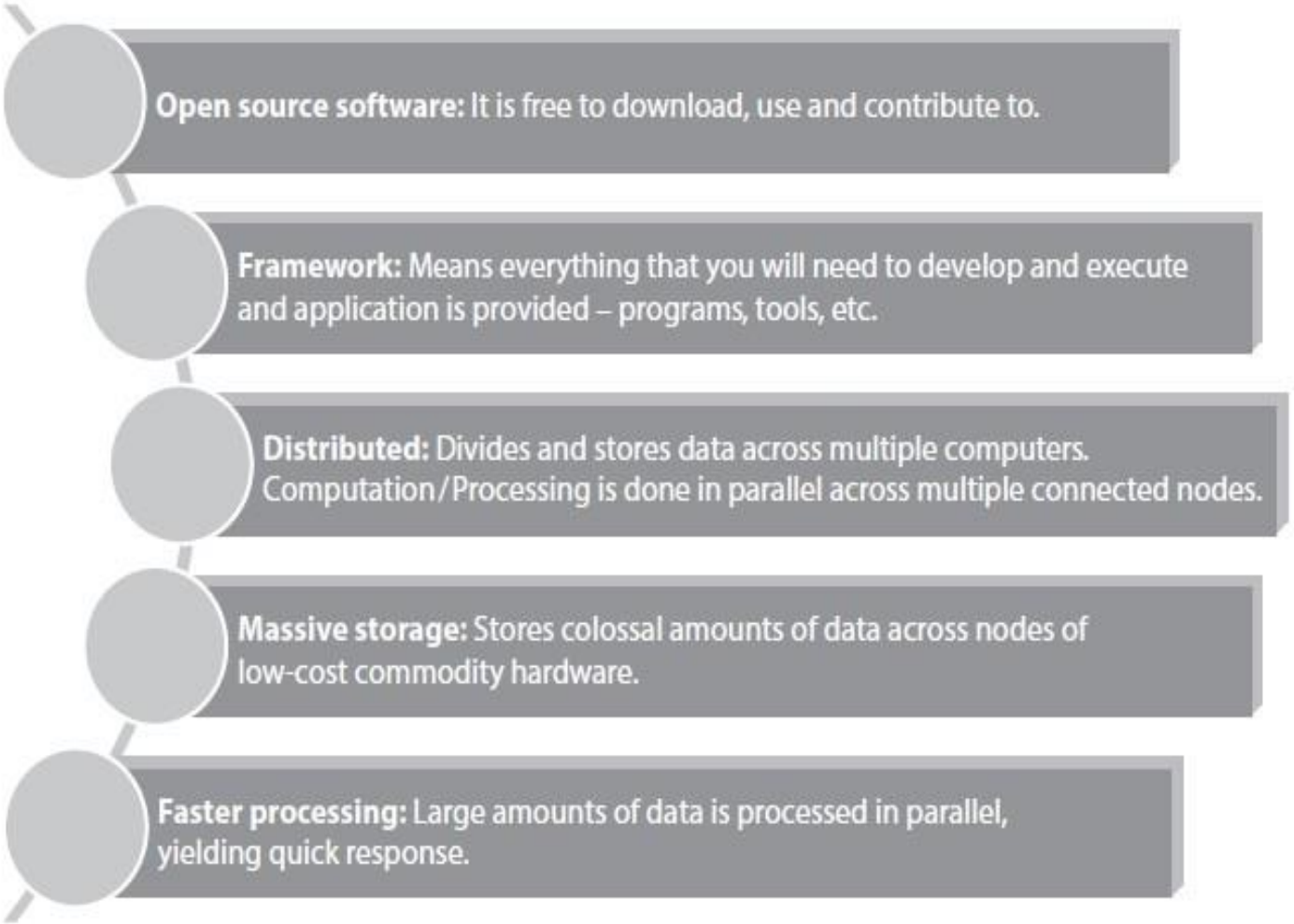
- Adknowledge - to build the recommender system for behavioral targeting, plus other clickstream analytics; clusters vary from 50 to 200 nodes, mostly on EC2.
- Contextweb - to store ad serving log and use it as a source for Ad optimizations/ Analytics/reporting/machine learning; 23 machine cluster with 184 cores and about 35TB raw storage. Each (commodity) node has 8 cores, 8GB RAM and 1.7 TB of storage.
- Cornell University Web Lab: Generating web graphs on 100 nodes (dual 2.4GHz Xeon Processor, 2 GB RAM, 72GB Hard Drive)
- NetSeer - Up to 1000 instances on Amazon EC2 ; Data storage in Amazon S3; Used for crawling, processing, serving and log analysis
- The New York Times : Large scale image conversions ; EC2 to run hadoop on a large virtual cluster
- Powerset / Microsoft - Natural Language Search; up to 400 instances on Amazon EC2 ; data storage in Amazon S3

What is Hadoop?

- **Hadoop:**
  - an open-source software framework that supports data-intensive distributed applications, licensed under the Apache v2 license.
- **Goals / Requirements:**
  - Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
    - Structured and non-structured data
    - Simple programming models
  - High scalability and availability
  - Use commodity (cheap!) hardware with little redundancy
  - Fault-tolerance
  - Move computation rather than data



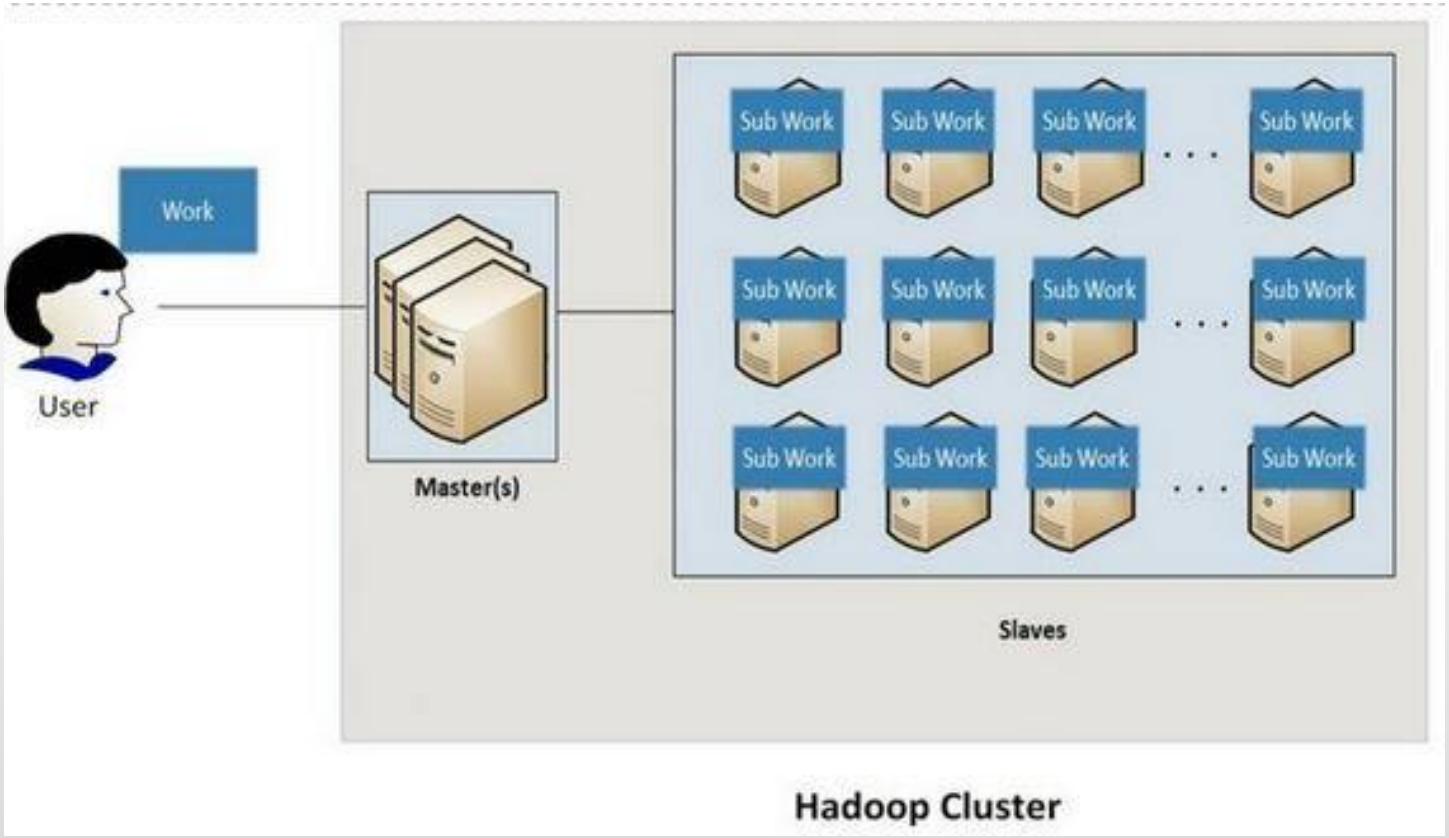
Key Aspects of Hadoop



Other Key Aspects

- **Fault Tolerance**
- **Data Reliability/Data Availability**
- **Scalability**
- **Economic**
- **Easy to use**
- **Data Locality** Hadoop works on data locality principle which states that move computation to data instead of data to computation. When a client submits the MapReduce algorithm, this algorithm is moved to data in the cluster rather than bringing data to the location where the algorithm is submitted and then processing it.

Basic HADOOP Architecture- Distributed Master-Slave Architecture



Hadoop Distributors

RDBMS versus HADOOP

RDBMS versus HADOOP

PARAMETERS	RDBMS	HADOOP
System	Relational Database Management System.	Node Based Flat Structure.
Data	Suitable for structured data.	Suitable for structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file formats, etc.
Processing	OLTP	Analytical, Big Data Processing
Choice	When the data needs consistent relationship.	Big Data processing, which does not require any consistent relationships between data.
Processor	Needs expensive hardware or high-end processors to store huge volumes of data.	In a Hadoop Cluster, a node requires only a processor, a network card, and few hard drives.
Cost	Cost around $10,000 to $14,000 per terabytes of storage.	Cost around $4,000 per terabytes of storage.

Why Hadoop??

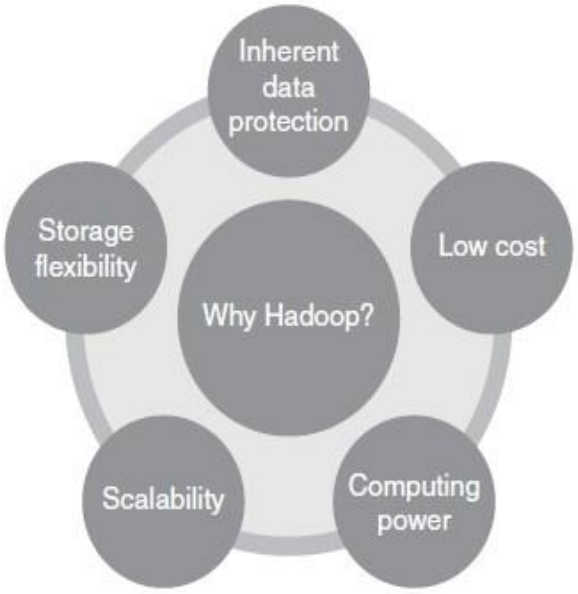
Hadoop is:

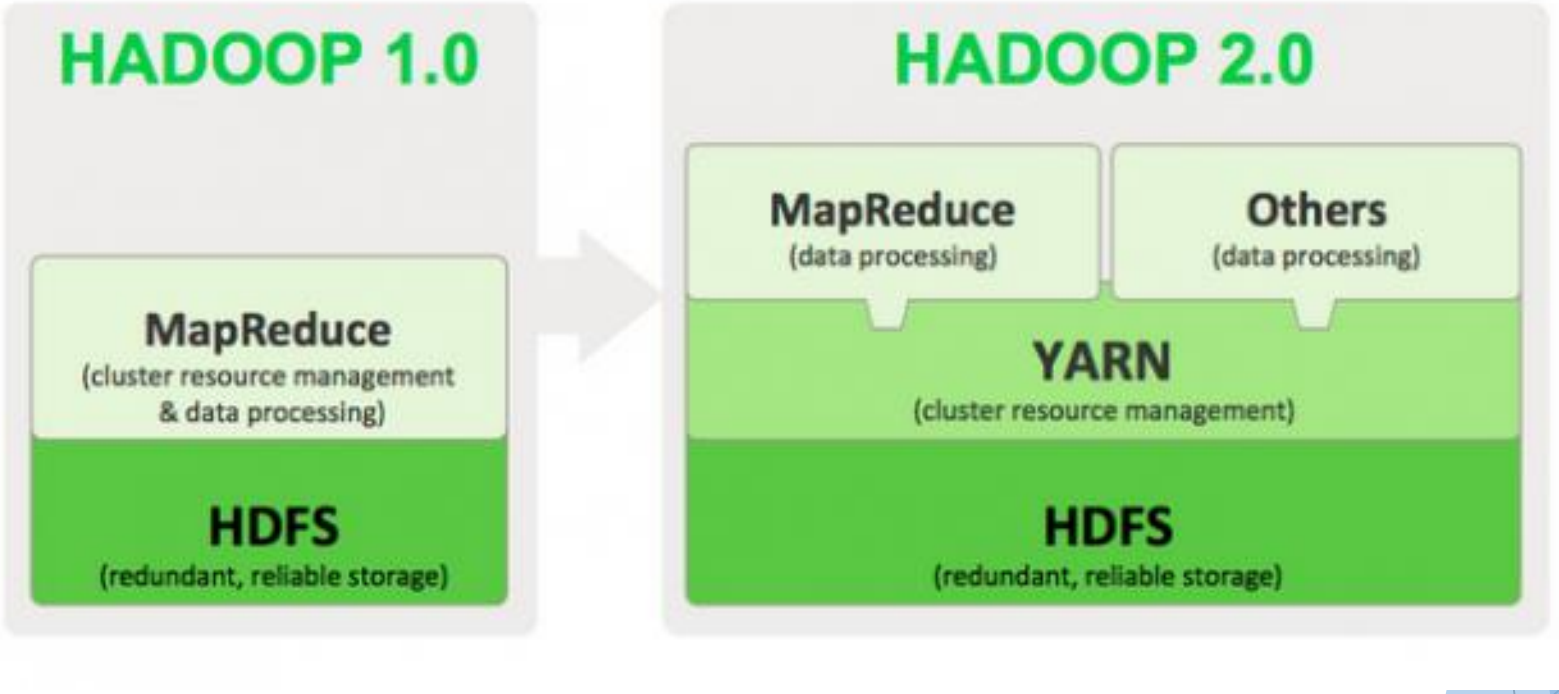
Ever wondered why Hadoop has been and is one of the most wanted technologies!!

The key consideration (the rationale behind its huge popularity) is:

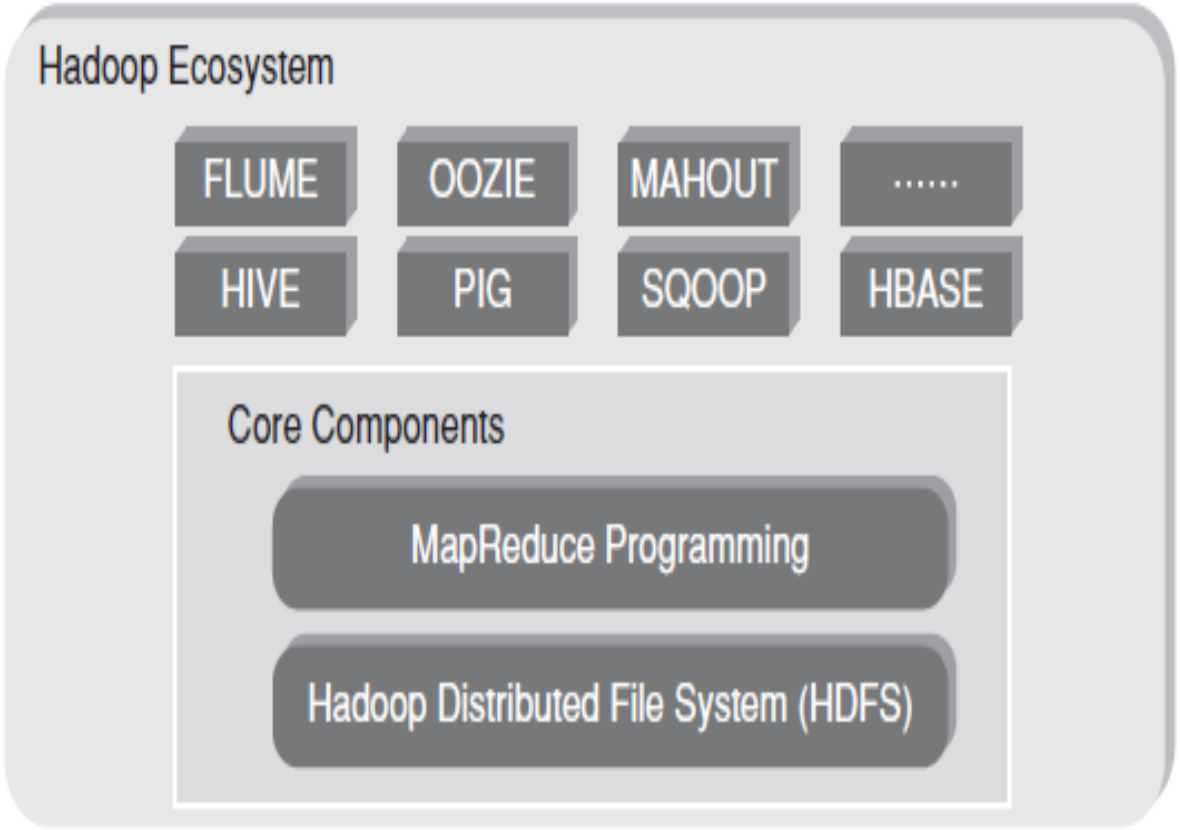
Its capability to handle massive am categories of data - fairly quickly.

The other considerations are :
Apache Hadoop is not only a storage system but is a platform for data storage as well as processing.
It is scalable (as we can add more nodes on the fly), Fault tolerant (Even if nodes go down, data processed by another node) and open source (we can change its source code as per the requirements).




HADOOP 1.0
MapReduce
(cluster resource management & data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MapReduce
(data processing)
Others
(data processing)
YARN
(cluster resource management)
HDFS
(redundant, reliable storage)

Hadoop Components



Hadoop Components

Hadoop Core Components:

HDFS: Data Storage Layer

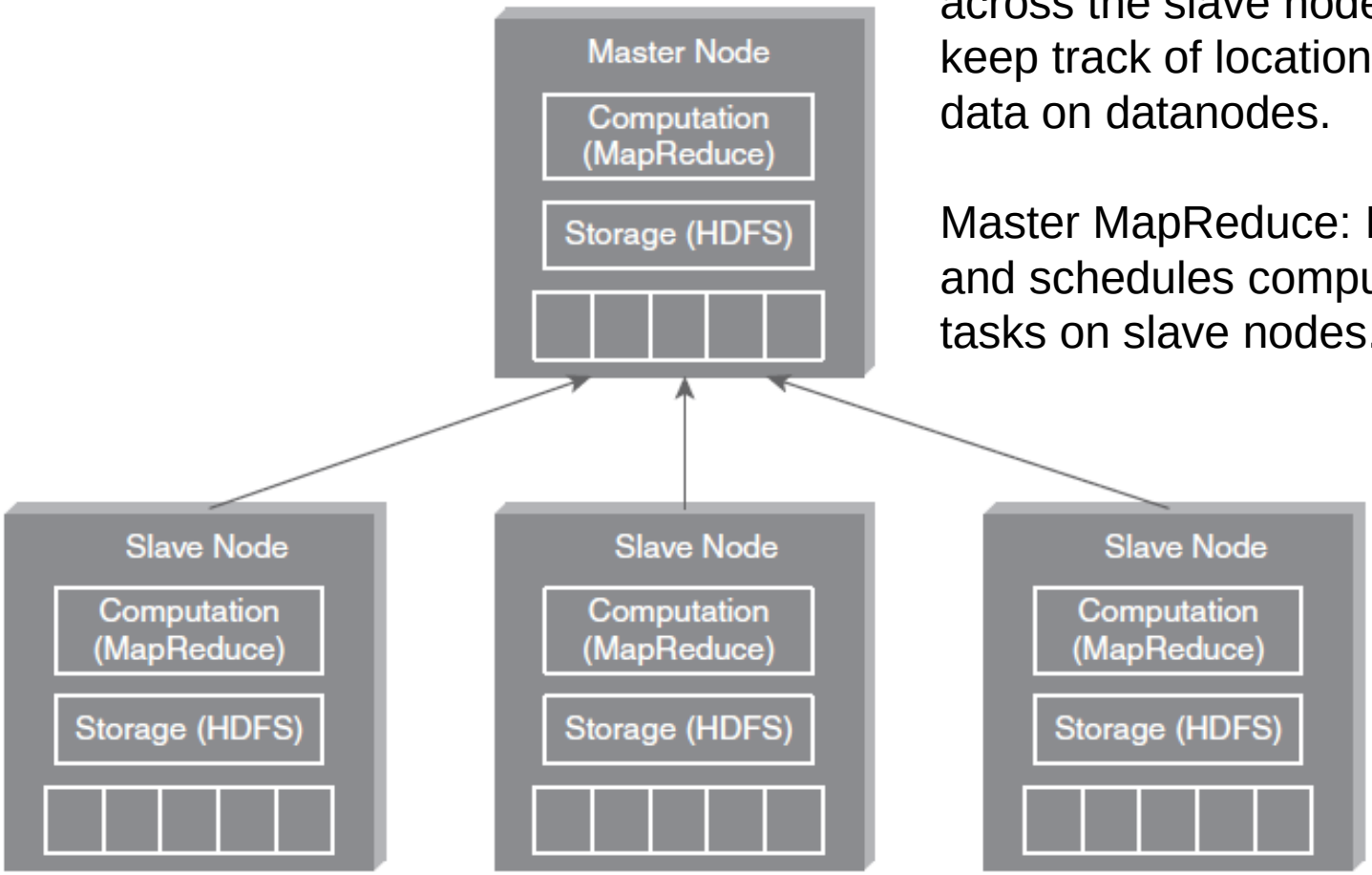
(a)Storage component.
(b)Distributes data across several nodes.
(c)Natively redundant.

MapReduce: Data Processing Layer

(a)Computational framework.
(b)Splits a task across multiple nodes.
(c)Processes data in parallel.

Conceptually Hadoop is divided into Data Storage Layer and Data Processing Layer

Hadoop High Level Architecture

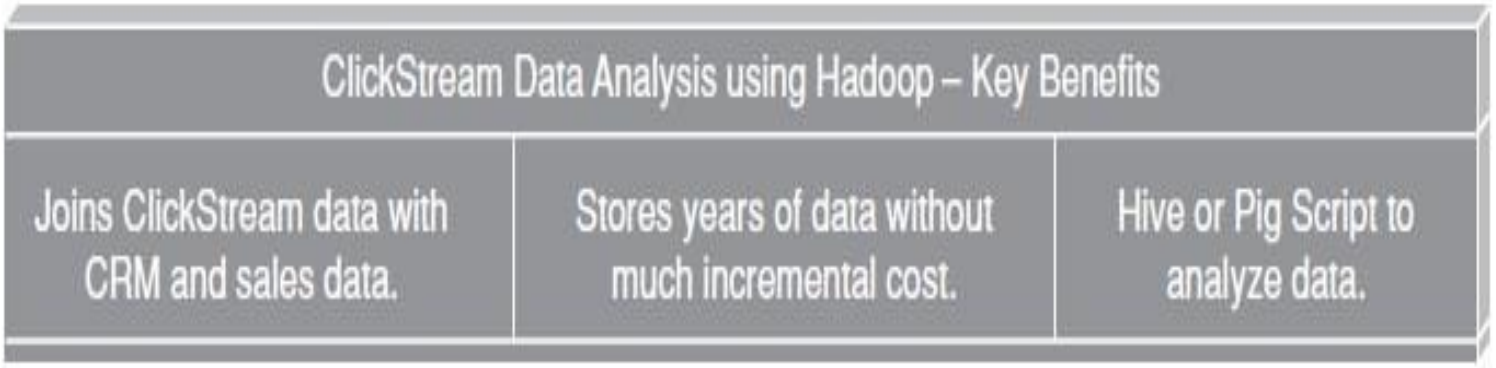


Master HDFS-responsible for partitioning the data storage across the slave nodes, also keep track of locations of data on datanodes.

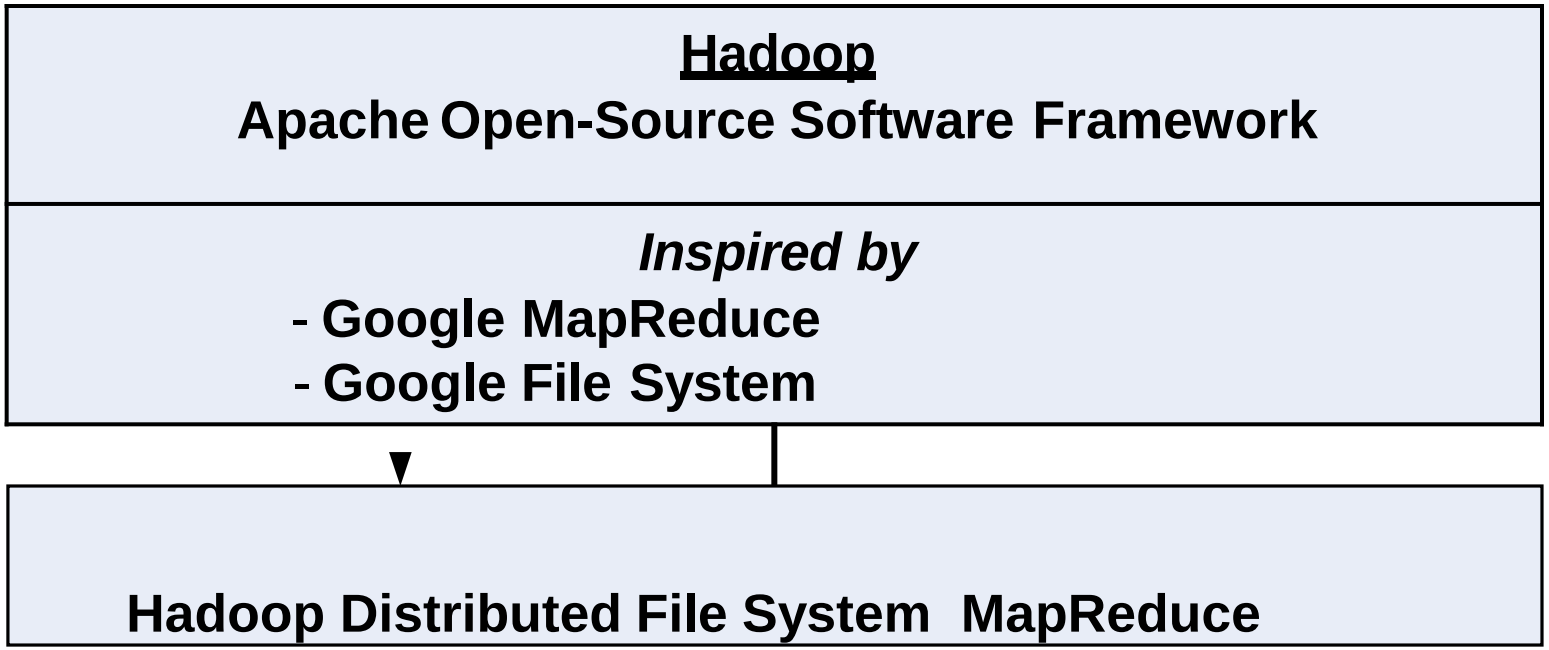
Master MapReduce: Decides and schedules computation tasks on slave nodes.

Use Case of Hadoop: ClickStream Data Analysis

ClickStream data (mouse clicks) helps you to understand the purchasing behavior of customers. ClickStream analysis helps online marketers to optimize their product web pages, promotional content, etc. to improve their business.



Hadoop


Hadoop
Apache Open-Source Software Framework
Inspired by
- Google MapReduce
- Google File System
Hadoop Distributed File System MapReduce

Hadoop: Assumptions

It is written with large clusters of computers in mind and is built around the following assumptions:

- Hardware *will* fail.
- Processing will be run in batches. Thus there is an emphasis on high throughput as opposed to low latency.
- Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size.
- It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.
- Applications need a **write-once-read-many** access model.
- Moving Computation is Cheaper than Moving Data[Fast].
- Portability is important.

Hadoop Core Components

HDFS – Hadoop Distributed File System (Storage)

Map Reduce (Processing)

HDFS
(HADOOP DISTRIBUTED FILE SYSTEM)

Hadoop Distributed File System

1. Storage component of Hadoop.

2. Distributed File System.

3. Modeled after Google File System.

4. Optimized for high throughput (HDFS leverages large block size and moves computation where data is stored).

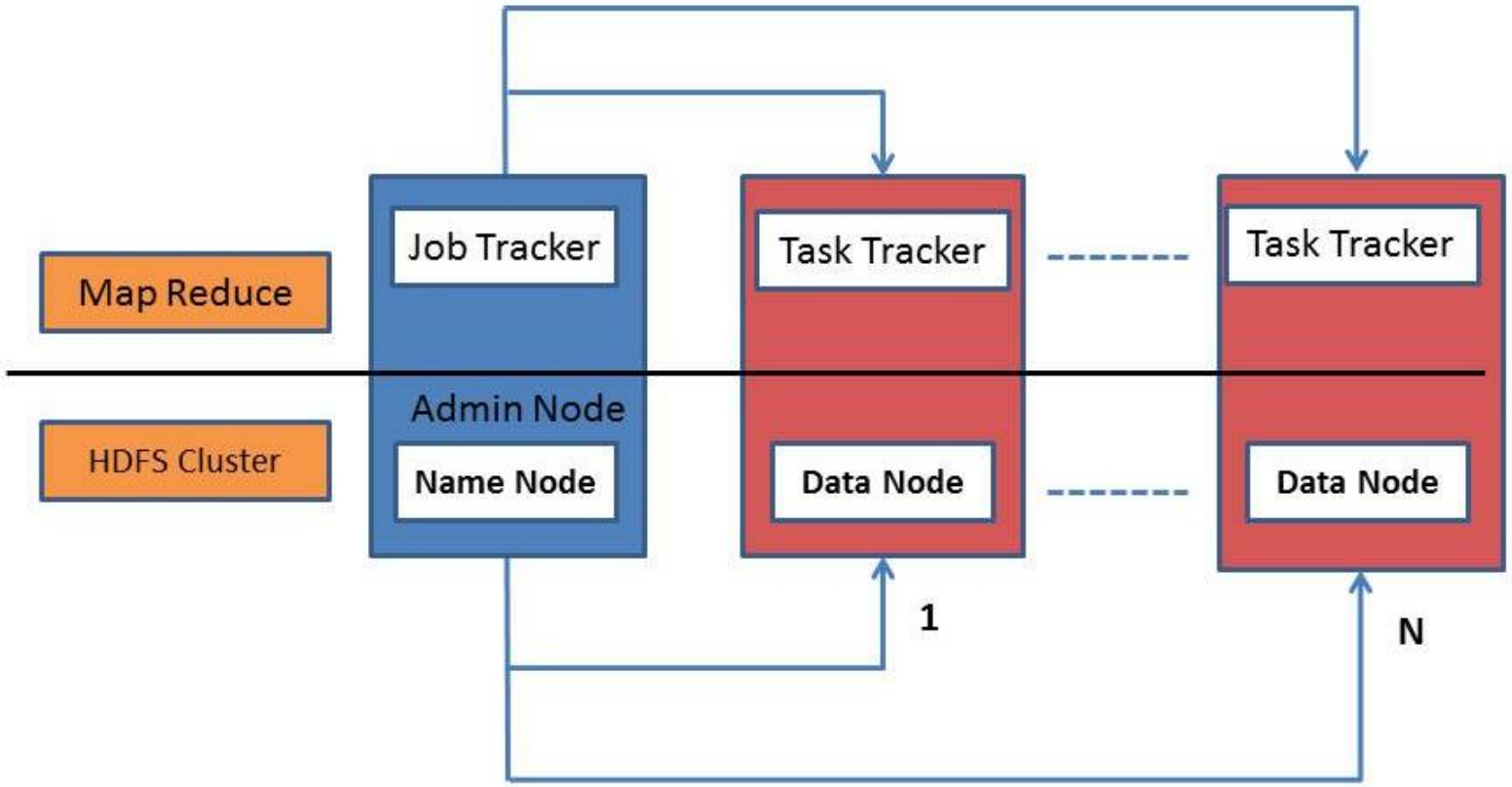
5. You can replicate a file for a configured number of times, which is tolerant in terms of both software and hardware.

6. Re-replicates data blocks automatically on nodes that have failed.

7. You can realize the power of HDFS when you perform read or write on large files (gigabytes and larger).

8. Sits on top of native file system such as ext3 and ext4(linux file system)

Hadoop Core Components



19CSE357-
Big Data Analytics(3-0-0-3)

Lecture #25

HADOOP

Chapter 5

Sangita Khare

Department of Computer Science & Engineering

HDFS-Key POints

1. Hadoop Distributed File System (HDFS) is designed to reliably store very large files across machines in a large cluster. It is inspired by the GoogleFileSystem.
2. Distribute large data file into blocks.(Block Sructured File)
3. Blocks are managed by different nodes in the cluster
4. Each block is replicated on multiple nodes. Default Replication factor: 3
5. Default Block Size:64MB/128MB

HDFS Daemons

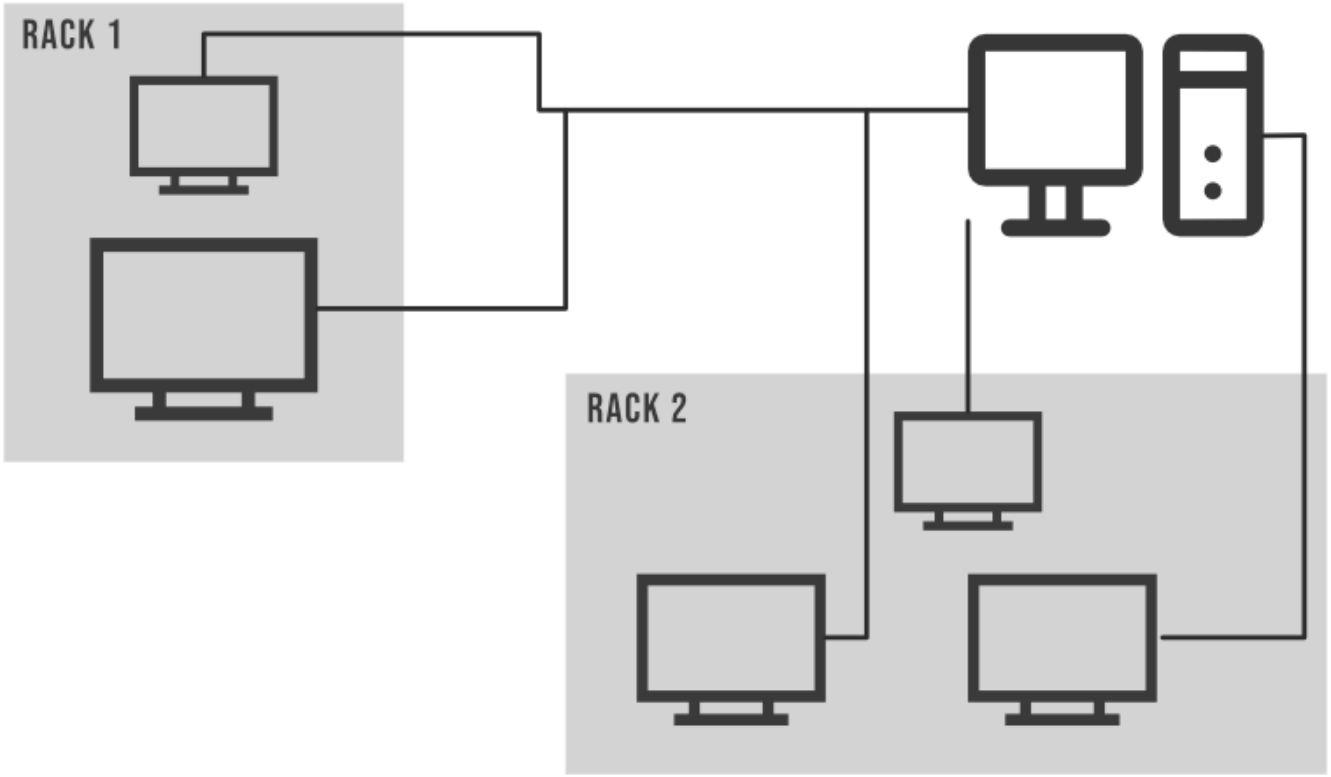
NameNode:

- Master of the system
- Maintains and manages the blocks which are present on the DataNodes

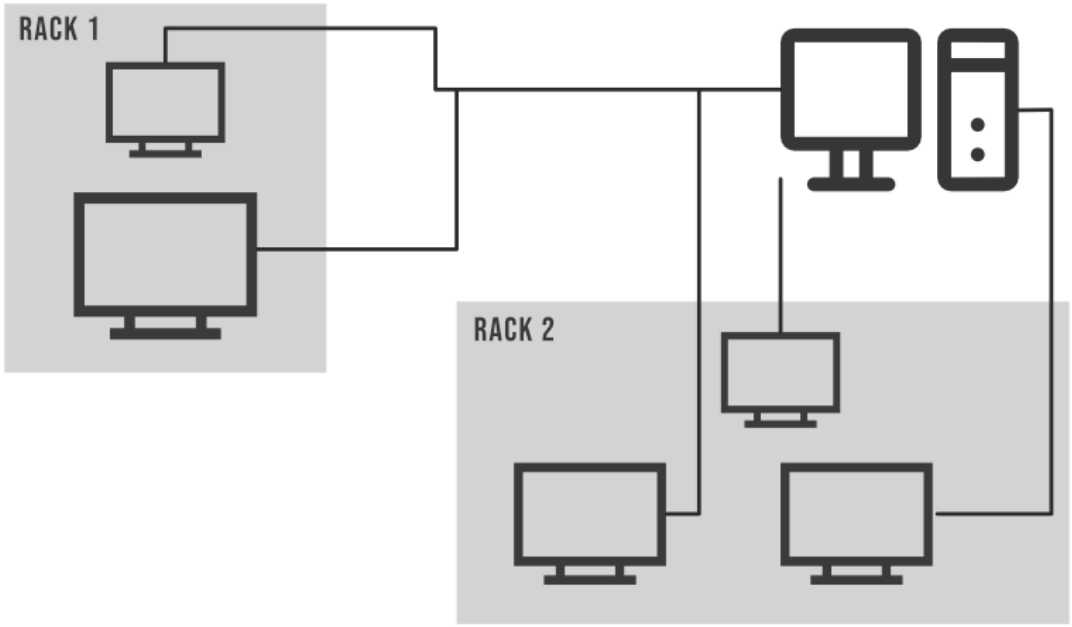
DataNodes:

- Slaves which are deployed on each machine and provide the actual storage
- Responsible for serving read and write requests for the clients

Racks in a cluster



The rack is a physical collection of nodes in our Hadoop cluster (maybe 30 to 40). A large Hadoop cluster is consists of many Racks. With the help of this Racks information, Namenode chooses the closest Datanode to achieve maximum performance while performing the read/write information which reduces the Network Traffic.



Communication between the Datanodes that are present on the same rack is quite much faster than the communication between the data node present at the 2 different racks.

The name node has the feature of finding the closest data node for faster performance for that Name node holds the ids of all the Racks present in the Hadoop cluster. This concept of choosing the closest data node for serving a purpose is **Rack Awareness**.

Replica Placement via Rack Awareness:

==> Block 1 , Block 2 , Block 3

Here, we have 3 different Racks in our Hadoop cluster each Rack contains 4 Datanode. Now suppose you have 3 file blocks(Block 1, Block 2, Block 3) that you want to put in this data node.

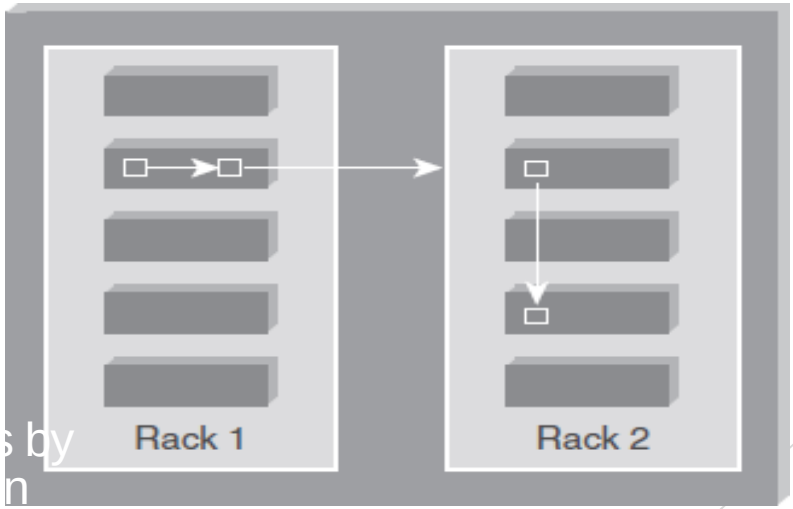
As we all know Hadoop has a Feature to make Replica's of the file blocks to provide the high availability and fault tolerance. By default, the Replication Factor is 3 so Hadoop is so smart that it will place the replica's of Blocks in Racks in such a way that we can achieve a good network bandwidth.

Source:https://www.geeksforgeeks.org/hadoop-rack-and-rack-awareness/

Replica Placement Strategy

As per the Hadoop Replica Placement Strategy, first replica is placed on the same node as the client. Then it places second replica on a node that is present on different rack. It places the third replica on the same rack as second, but on a different node in the rack. Once replica locations have been set, a pipeline is built. This strategy provides good reliability, availability and fault tolerance

The current default HDFS block placement policy guarantees that a block's 3 replicas will be placed on at least 2 racks. Specifically one replica is placed on one rack and the other two replicas are placed on another rack during write pipeline.



How Hadoop works

Step 1: Input data is broken into blocks of size 64MB or 128MB and then blocks are moved to different nodes.
Step 2: Once all the blocks of the are stored on data-nodes, user can process the data.
Step 3: Master, then schedules the program (submitted by user) on individual nodes.
Step 4: Once all the nodes process the data, output is written back on HDFS

NAMENODE

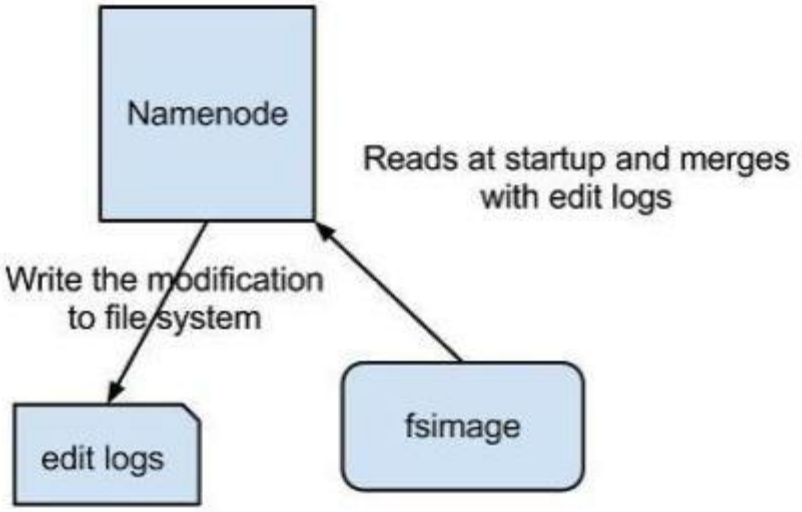
1. NameNode is the main central component of HDFS architecture framework. NameNode is also known as Master node.
2. HDFS Namenode stores meta-data i.e. number of data blocks, file name, path, Block IDs, Block location, no. of replicas, and also Slave related configuration. This meta-data is available in memory in the master for faster retrieval of data.
3. **NameNode keeps metadata related to the file system namespace in memory, for quicker response time. It manages file related operations such as read, write, create and delete. File system namespace is collection of files in the cluster. NameNode stores HDFS namespace. File System namespace includes mapping of blocks to file, file properties etc stored in Fsimage file
4. NameNode maintains and manages the slave nodes, and assigns tasks to them. NameNode has knowledge of all the DataNodes containing data blocks for a given file.
5. It uses RACK-ID to identify DataNodes in the Rack
6. NameNode coordinates with hundreds or thousands of data nodes and serves the requests coming from client applications.

Two files ‘FSImage’ and the ‘EditLog’ are used to store metadata information in NameNode.

1. **FsImage: It is the snapshot the file system when Name Node is started. It is an “Image file”. FsImage contains the entire filesystem namespace and stored as a file in the NameNode’s local file system. It also contains a serialized form of all the directories and file inodes in the filesystem. Each inode is an internal representation of file or directory’s metadata.**
2. **EditLogs-Transaction Log: It records every transaction that occurs to file system metadata. NameNode receives a create/update/delete request from the client. After that this request is first recorded to edits file.**

Functions of NameNode

1. It is the master daemon that maintains and manages the DataNodes (slave nodes).
2. When NameNode starts up ,it reads FsImage and EditLog from disk and applies all transactions from Editlog to in-memory representation of the FsImage.Then it flushes out the new version of FsImage on the disk and truncates the old Editlog because the change are updated in the FsImage.



The image shows how Name Node stores information in disk.
Two different files are
1. fsimage – It is the snapshot of the filesystem ,in which entire filesystem is stored.
2. Edit logs – It is the sequence of changes made to the filesystem after NameNode started.

Functions of NameNode

3. It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. FsImage and EditLogs files associated with the metadata

4. It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.

5. It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.

6. The NameNode is also responsible to take care of the replication factor of all the blocks.

7. In case of the DataNode failure, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

Why Secondary NameNode?

When the NameNode is in the active state the edit logs size grows continuously. Only in the restart of NameNode , edit logs are applied to fsimage to get the latest snapshot of the file system. But NameNode restart are rare in production clusters which means edit logs can grow very large for the clusters where NameNode runs for a long period of time. The following issues we will encounter in this situation.

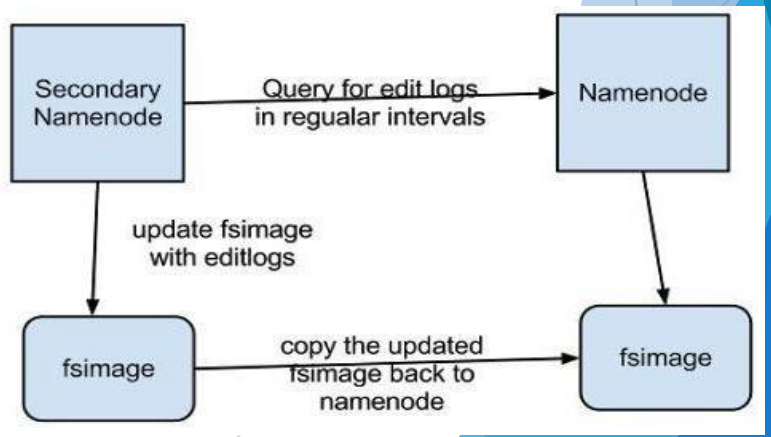
- Editlog become very large, which will be challenging to manage it.
- Namenode restart takes long time because lot of changes has to be merged.
- In the case of crash, we will have lost huge amount of metadata since fsimage is very old.

So to overcome this issues we need a mechanism which will help us reduce the edit log size which is manageable and have up to date fsimage, so that load on NameNode reduces.

It’s very similar to Windows Restore point, which will allow us to take snapshot of the OS so that if something goes wrong, we can fall back to the last restore point.

Secondary NameNode

1. Secondary NameNode helps to overcome the above issues by taking over responsibility of merging editlogs with fsimage from the NameNode.
2. Secondary NameNode also contains a namespace image and edit logs like NameNode. Now after every certain interval of time (which is one hour by default) it copies the namespace image from NameNode and merge this namespace image with the edit log and copy it back to the NameNode so that NameNode will have the fresh copy of namespace image.
3. Now let's suppose at any instance of time the NameNode goes down and becomes corrupt then we can restart other machine with the namespace image and the edit log that's what we have with the Secondary NameNode and hence can be prevented from a total failure.
4. Secondary Name node takes almost the same amount of memory and CPU for its working as the NameNode. So, it is also kept in a separate machine like that of a NameNode.



HDFS DataNode

1. DataNodes are the slave nodes in HDFS. Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability..
2. In Hadoop HDFS Architecture, DataNode stores actual data in HDFS.
3. DataNodes are responsible for serving, read and write requests for the clients.
4. DataNodes can deploy on commodity hardware.
5. DataNodes sends information to the NameNode about the files and blocks stored in that node and responds to the NameNode for all filesystem operations.
6. When a DataNode starts up it announce itself to the NameNode along with the list of blocks it is responsible for.
7. DataNode is usually configured with a lot of hard disk space. Because the actual data is stored in the DataNode.

Functions of DataNode in HDFS

1. These are slave daemons or process which runs on each slave machine.
2. The actual data is stored on DataNodes.
3. The DataNodes perform the low-level read and write requests from the file system's clients.
4. Every DataNode sends a heartbeat message to the Name Node every 3 seconds and conveys that it is alive. In the scenario when Name Node does not receive a heartbeat from a Data Node for 10 minutes, the Name Node considers that particular Data Node as dead and starts the process of Block replication on some other Data Node.
5. All Data Nodes are synchronized in the Hadoop cluster in a way that they can communicate with one another and make sure of
    a. Balancing the data in the system
    b. Move data for keeping high replication
    c. Copy Data when required.

HDFS Daemons

NameNode:

- Single NameNode per cluster.
- Keeps the metadata details

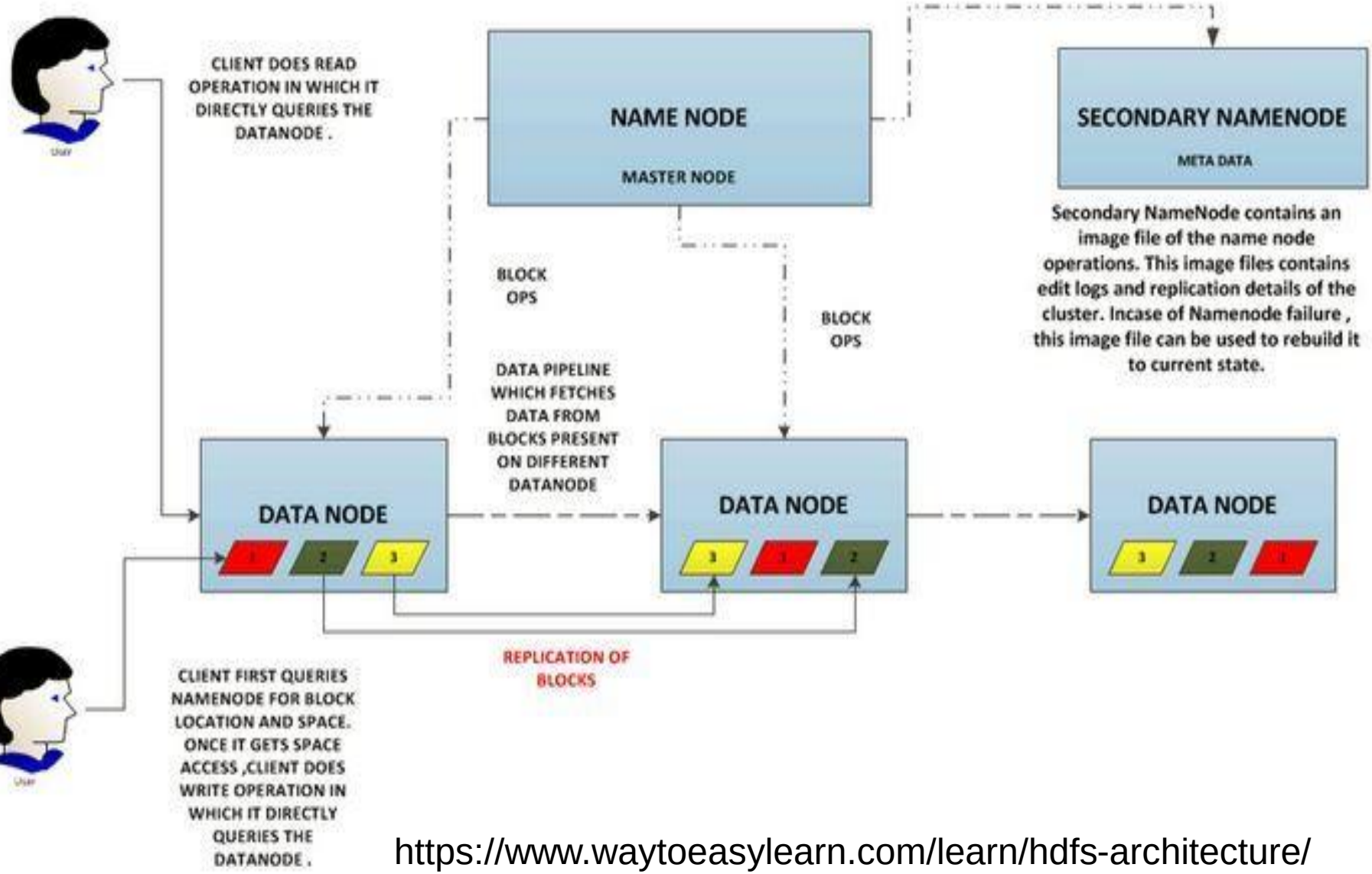
DataNode:

- Multiple DataNode per cluster
- Read/Write operations

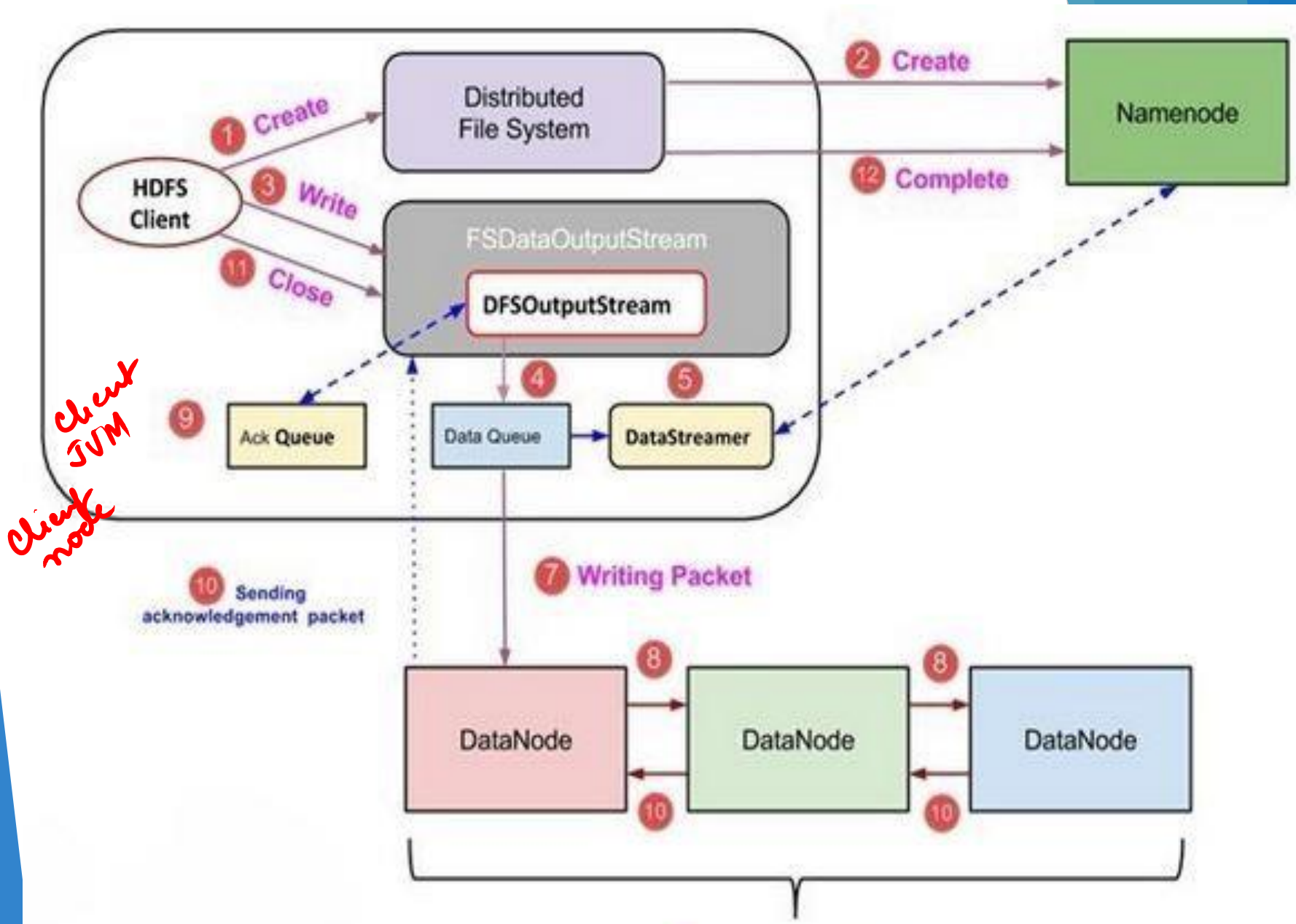
SecondaryNameNode:

- Housekeeping Daemon

HDFS ARCHITECTURE

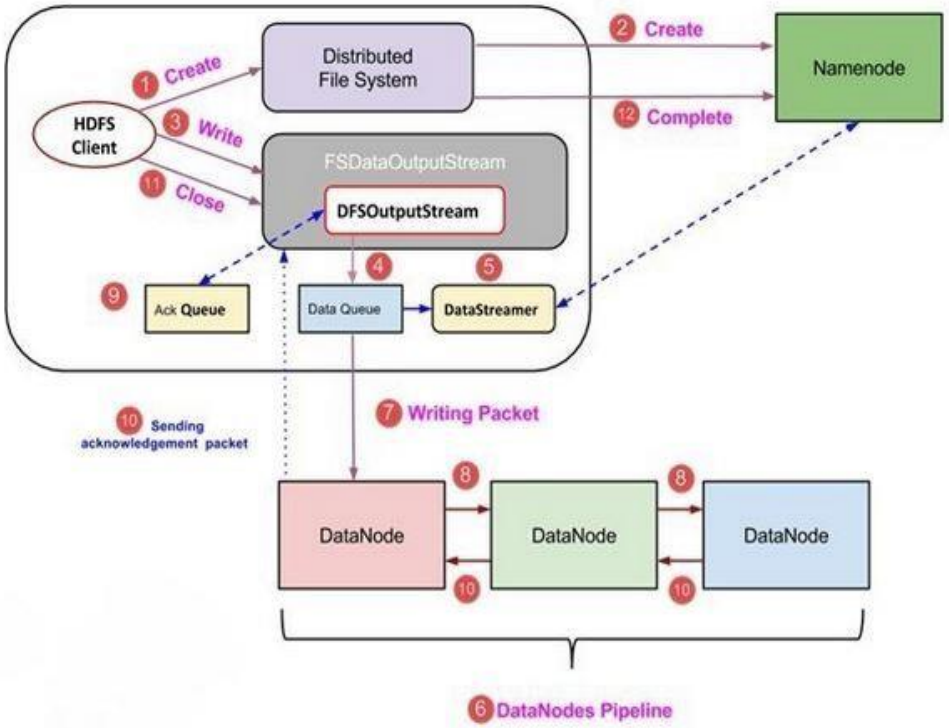


https://www.waytoeasylearn.com/learn/hdfs-architecture/



Anatomy of File Write

1. The HDFS client sends a WRITE request on DistributedFileSystem API using create().-1,2
2. DistributedFileSystem issue a RPC call to the NameNode to create a new file in FS namespace. After various checks, client gets the permission or an IOException.
3. The DistributedFileSystem return FSDataOutputStream to the client for writing data. As the client writes data, DFSOutputStream splits it into packets, which it writes to an internal queue, called the data queue. The data queue is consumed by the DataStreamer, that requests the NameNode to allocate new blocks by picking a list of suitable data nodes to store the replicas.(3,4,5)



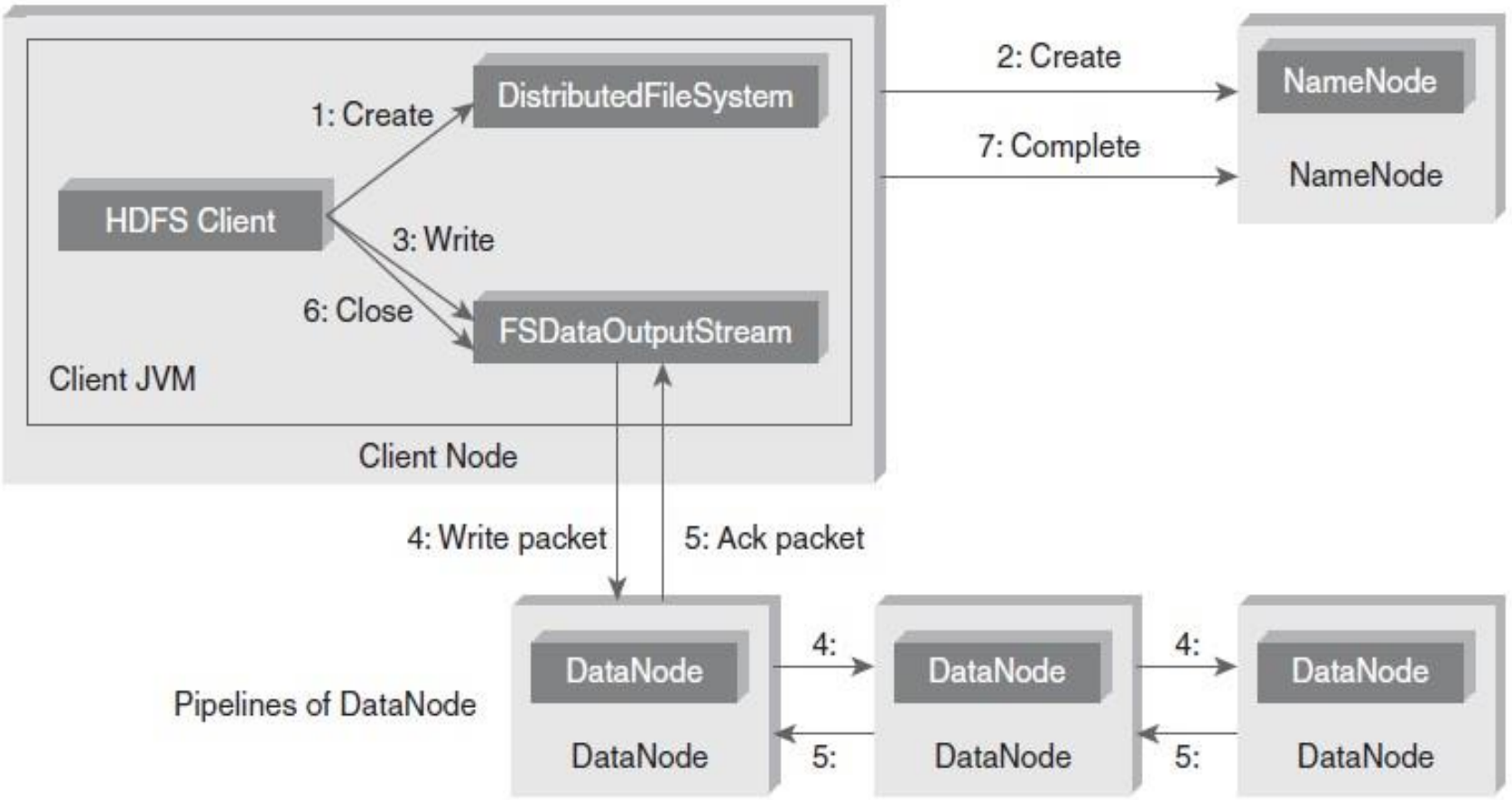
Anatomy of File Write

4. The list of data nodes forms a pipeline based on the replication Level. The default is 3. The DataStreamer streams the packets to the first data node in the pipeline, which stores the packet a forwards it to the second data node in the pipeline. so is the second node does and send it to the data node.(6,7,8)

5. DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowled by data nodes, called the "ack queue". A packet gets removed as soon as it has been acknowledg all the data nodes in the pipeline. Datanode sends the acknowledgment once required replicas are created.(9,10)

6. The client calls close() on the stream when done which flushes all the remaining packets to the node pipeline and waits for acknowledgments before contacting the NameNode to signal that the complete. The NameNode already knows the blocks the file is made up of, so it only has to wait fo blocks to be minimally replicated before returning successfully.(11)

Anatomy of File Write



19CSE357-
Big Data Analytics(3-0-0-3)

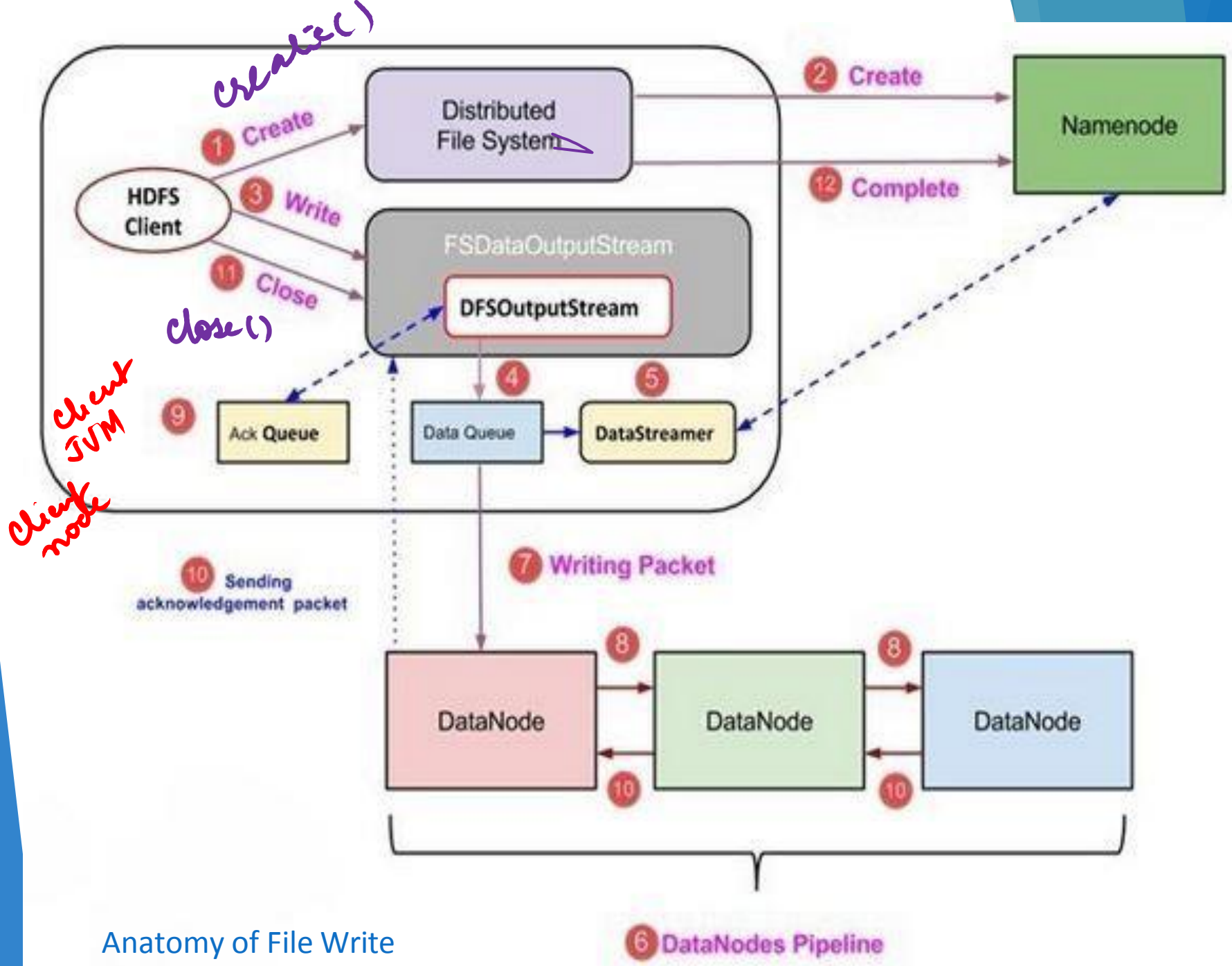
Lecture #26,27

HADOOP

Chapter 5

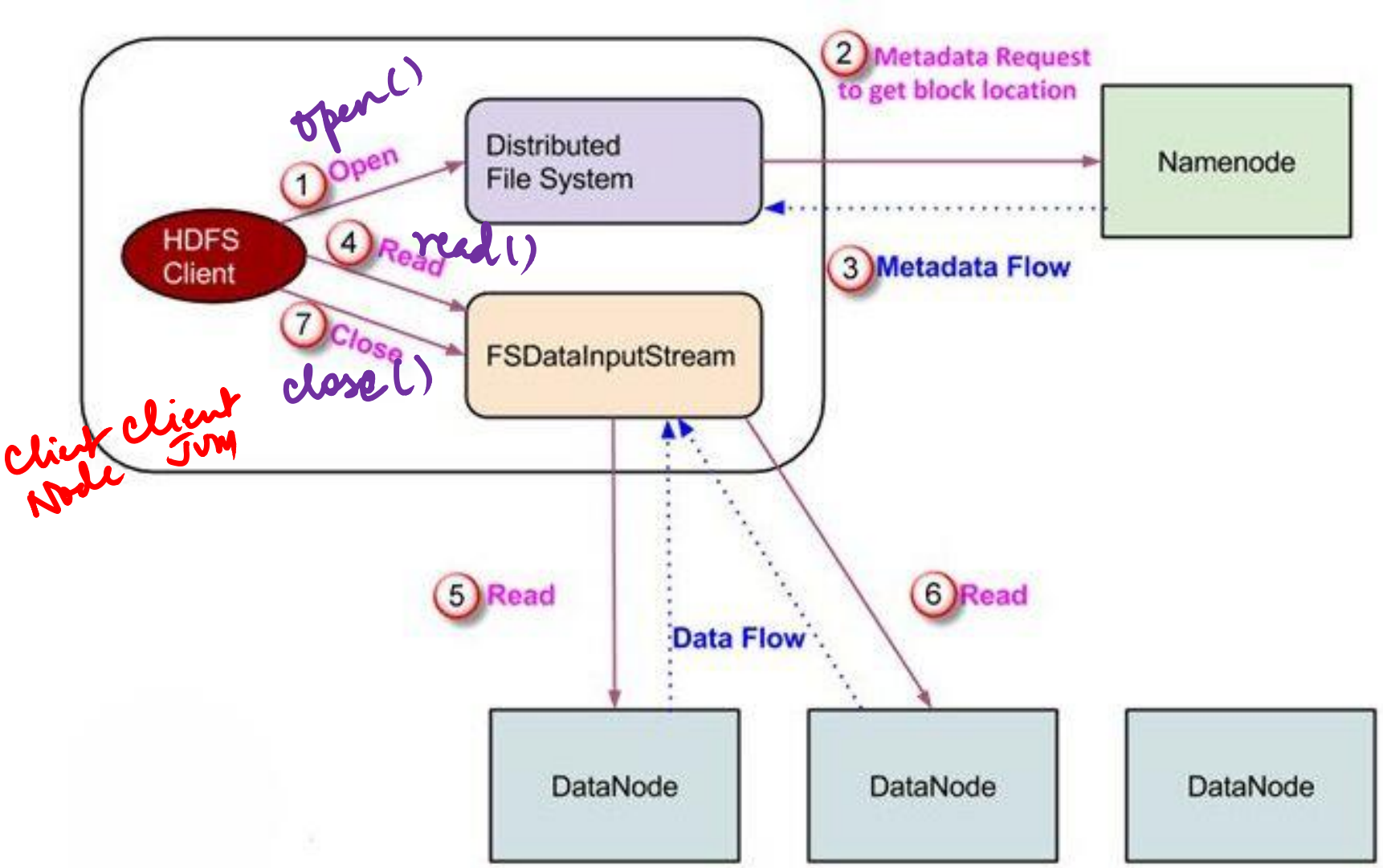
Sangita Khare

Department of Computer Science & Engineering

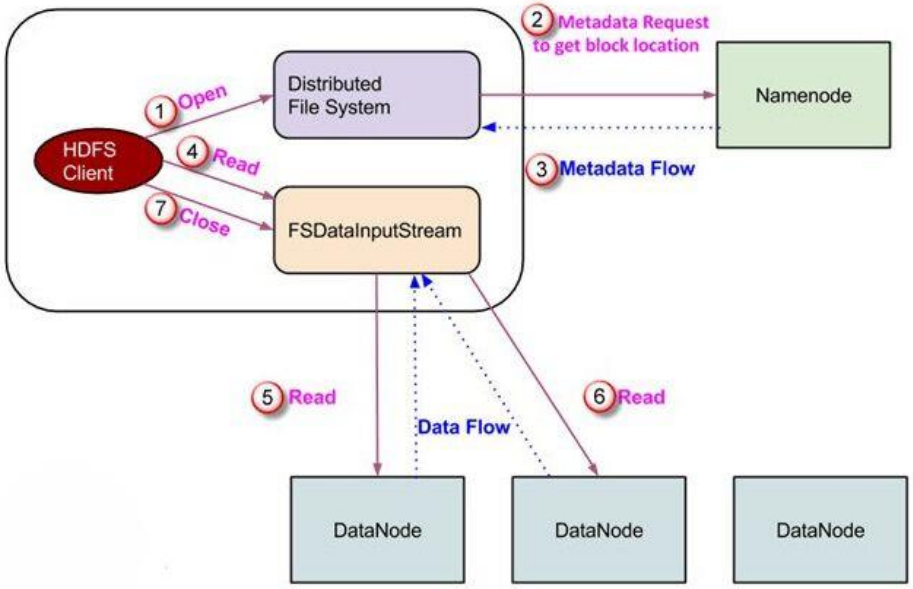


Anatomy of File Write

Anatomy of File Read



Anatomy of File Read



1. To access the blocks stored in HDFS, a client initiates a request 'open()' on DistributedFileSystem, which in turn will interact with the NameNode using RPC (Remote Procedure Call), to get the location of the slaves which contains the blocks of the file requested by the client.(1,2)
2. At this level, NameNode will check whether the client is authorized to access that file, if yes then it sends the information of the block locations. Also, it gives a security token to the client which they need to show to the slaves for authentication.(3)

Client interact with DataNode

3. Now after receiving the address of slaves containing blocks, the client will directly interact with Slave to read the blocks. The data will flow directly from the slaves to the client (it will not flow via NameNode). The client reads the data block from multiple slaves in parallel.

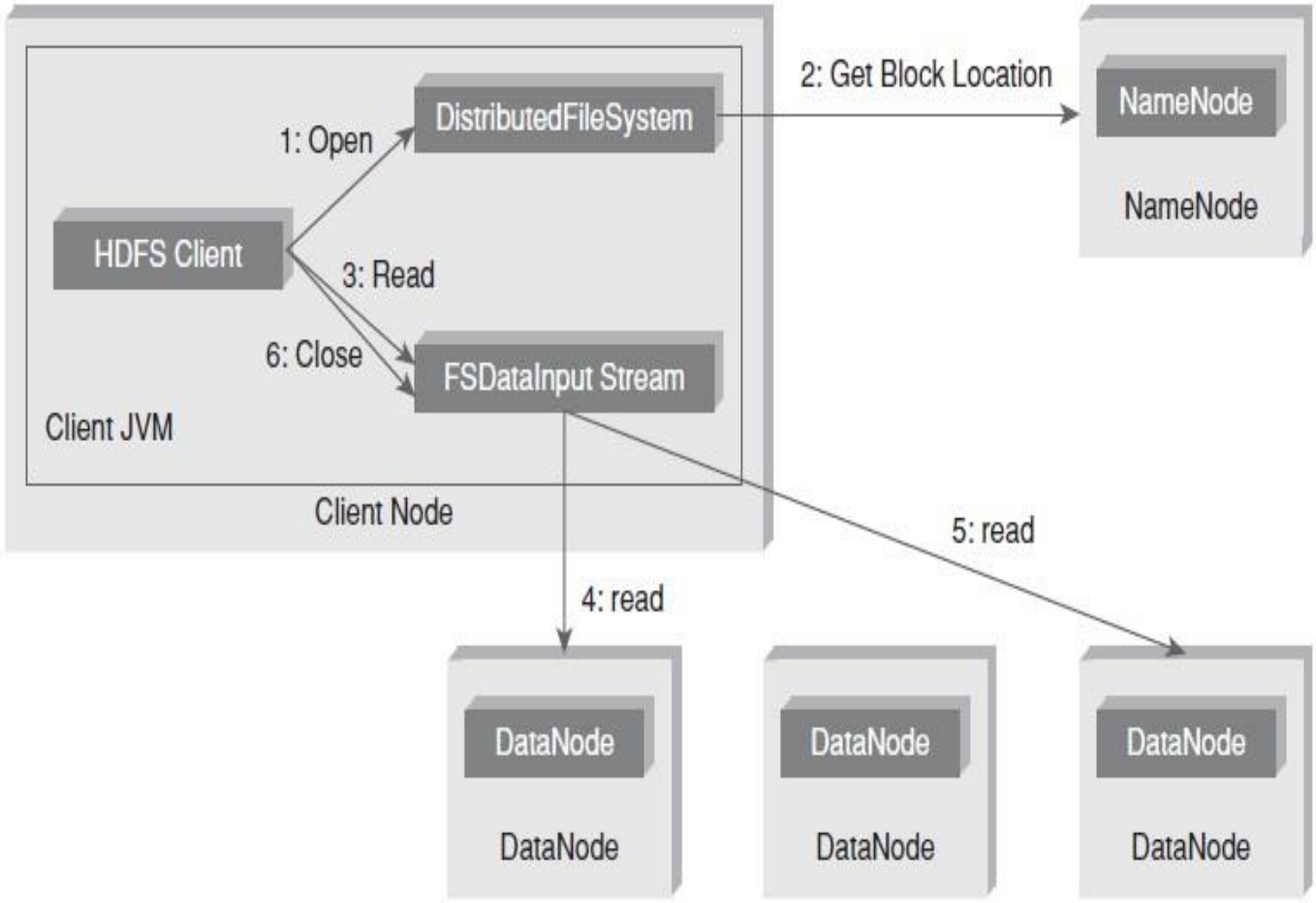
4. Let's understand client and NameNode interaction in more details. The client will send a read request to slaves via using the FSDataInputStream object. All the interactions between client and DataNode are managed by the DFSInputStream which is a part of client APIs. DFSInputStream has the addresses of Datanodes for blocks of file.The client will show authentication token to slaves which were provided by NameNode. Now client will start reading the blocks using read(), data will be transmitted continuously from closest DataNode to client.
Client calls read() repeatedly to stream the data from the DataNode.(4,5,6)

5. After reaching an end of a block, the connection is closed to the DataNode by the DFSInputStream.It repeats the steps to find the best DataNode for the next block and subsequent blocks.

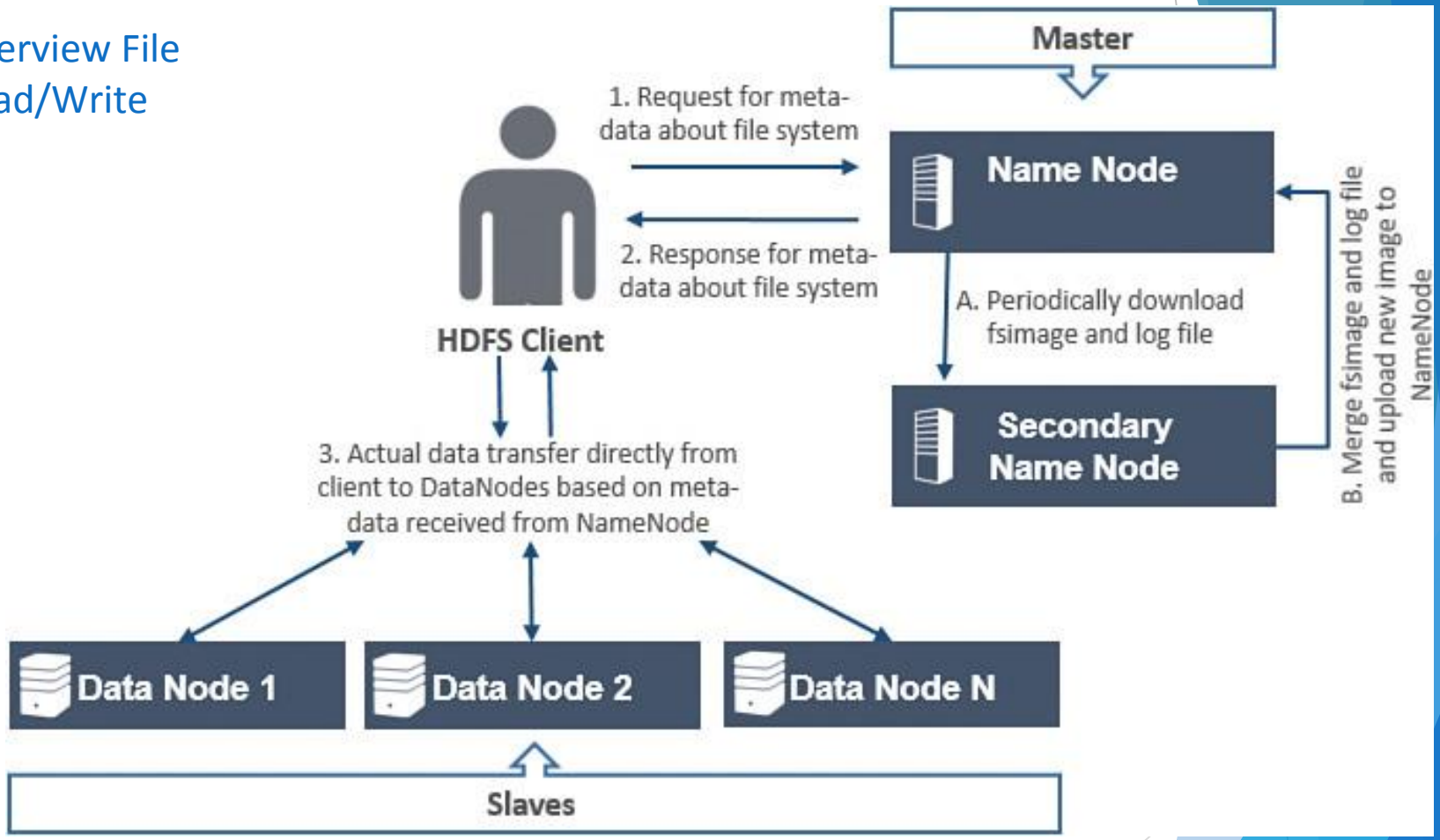
6.. When client completes reading a file,it calls close() on FSDataInputStream to close the connection(7)

Read operation is highly optimized, as it does not involve NameNode for actual data read, otherwise NameNode would have become the bottleneck. Due to distributed parallel read mechanism, thousands of clients can read the data directly from DataNodes very efficiently.
Data is read from the DataNode that is closest to the client.

Anatomy of File Read



Overview File Read/Write

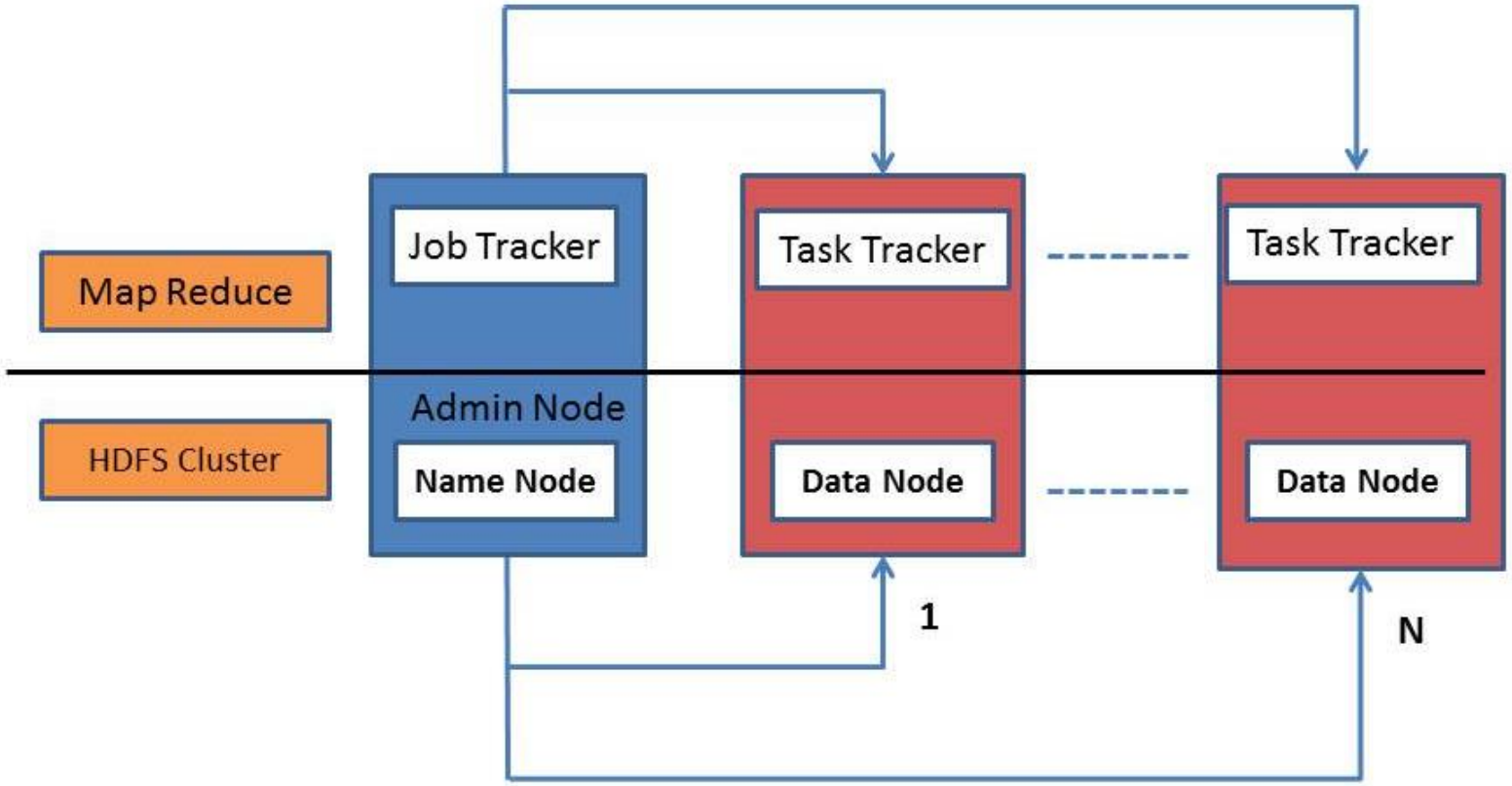


Special Features of HDFS

Data Replication: There is absolutely no need for a client application to track all blocks. It directs the client to the nearest replica to ensure high performance.

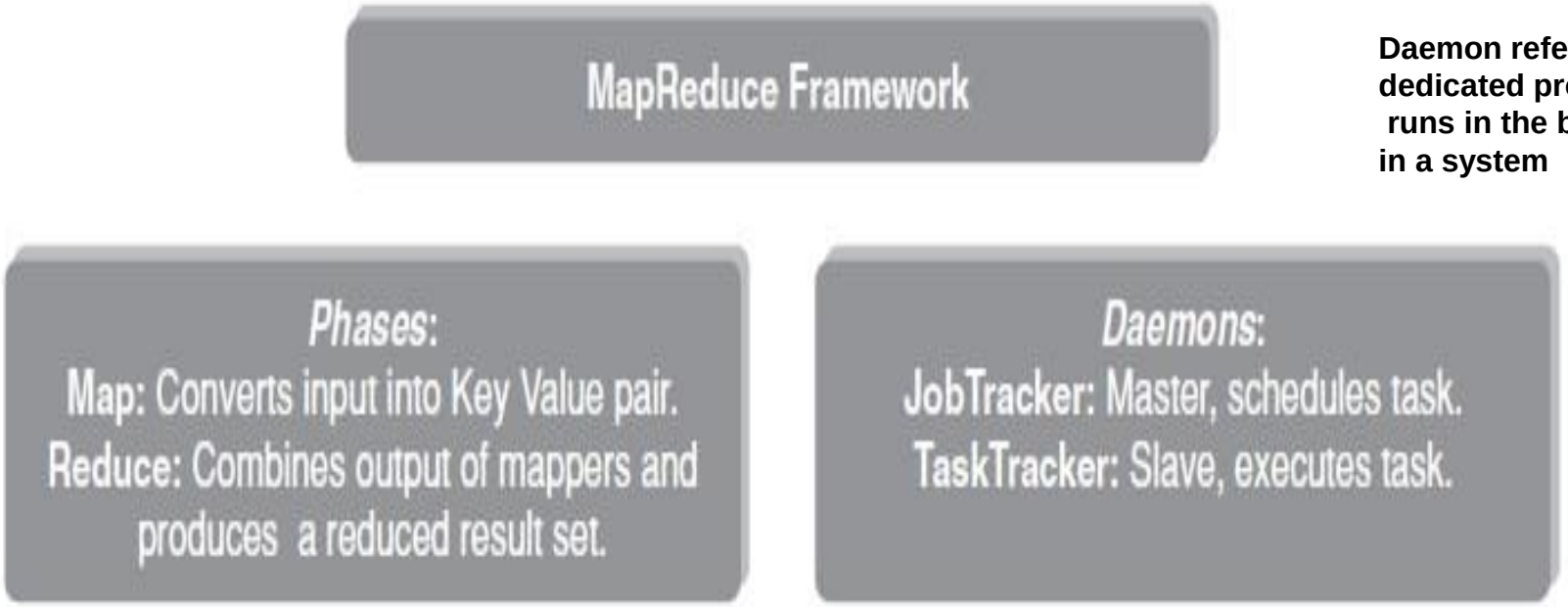
Data Pipeline: A client application writes a block to the first DataNode in the pipeline. Then this DataNode takes over and forwards the data to the next node in the pipeline. This process continues for all the data blocks, and subsequently all the replicas are written to the disk.

Hadoop Core Components

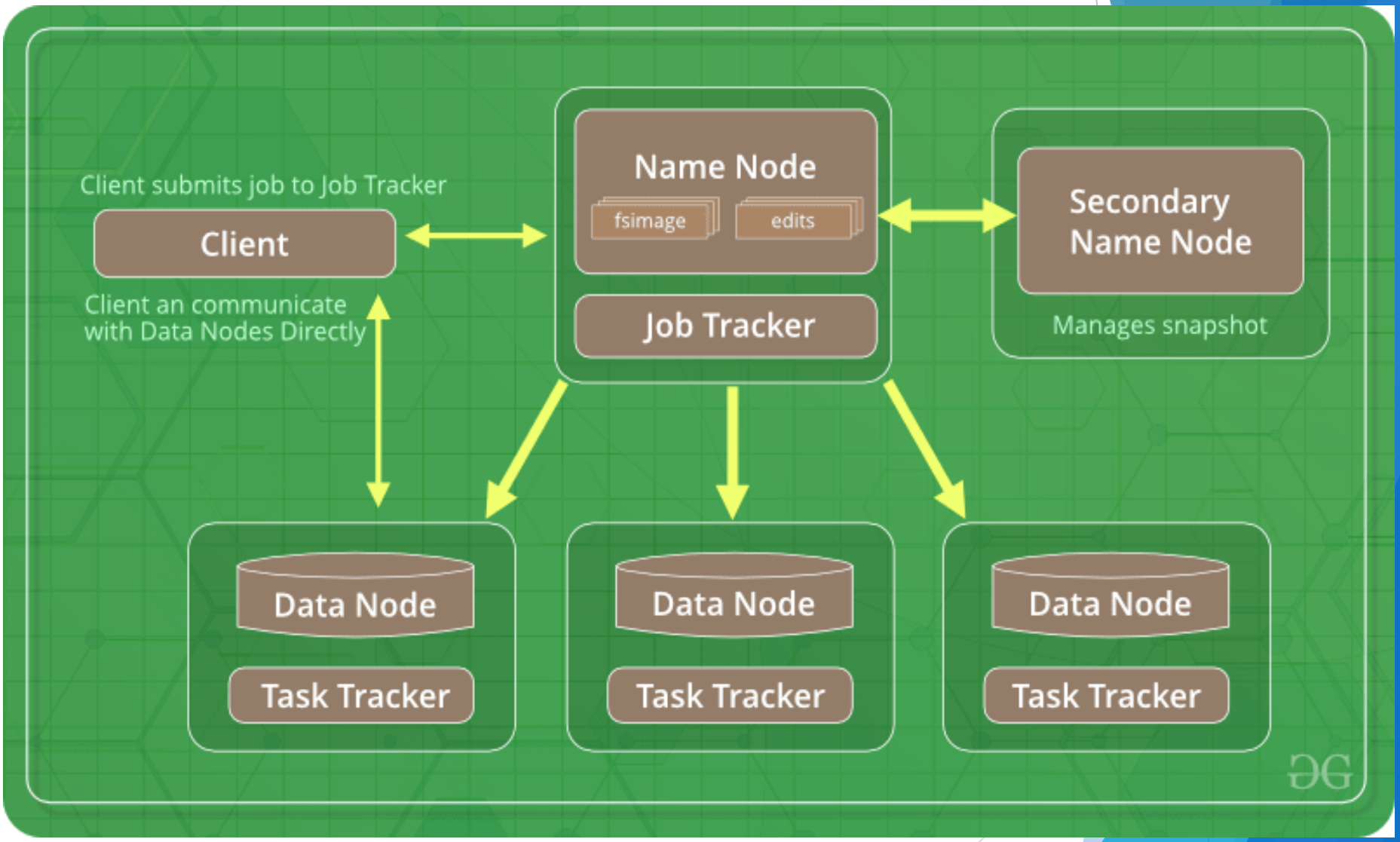


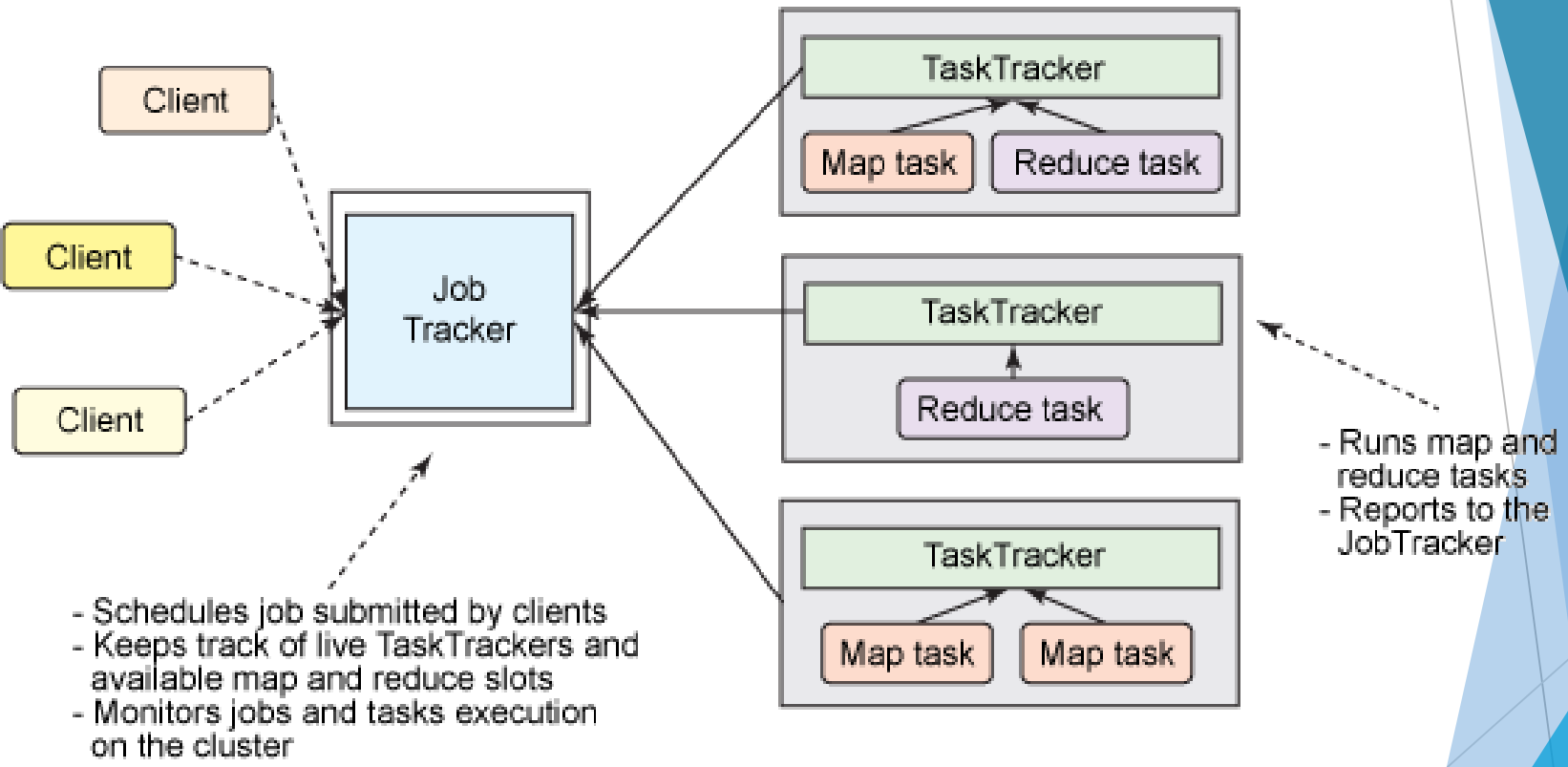
What is MapReduce Programming?

MapReduce Programming is a software framework. MapReduce Programming helps you to process massive amounts of data in parallel.



Daemon refers to highly dedicated program that runs in the background in a system


Client submits job to Job Tracker
Client
Client an communicate with Data Nodes Directly
Name Node
fsimage
edits
Job Tracker
Secondary Name Node
Manages snapshot
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker


Client
Client
Client
Job Tracker
TaskTracker
Map task
Reduce task
TaskTracker
Reduce task
TaskTracker
Map task
Map task
- Schedules job submitted by clients
- Keeps track of live TaskTrackers and available map and reduce slots
- Monitors jobs and tasks execution on the cluster
- Runs map and reduce tasks
- Reports to the JobTracker

Jobtracker

1. The **jobtracker** daemon is responsible for accepting job requests from a client and scheduling/assigning tasktrackers with tasks to be performed. There is single JobTracker per Hadoop Cluster.

2. The jobtracker daemon tries to assign tasks to the tasktracker daemon on the datanode daemon where the data to be processed is stored. This feature is called **data locality**.

3. If for some reason the node hosting the datanode and tasktracker daemons fails, the jobtracker daemon assigns the task to another tasktracker daemon where the replica of the data exists. This is possible because of the replication factor configuration for HDFS where the data blocks are replicated across multiple datanodes. This ensures that the job does not fail even if a node fails within the cluster.

Tasktracker

1. The **tasktracker** daemon is a daemon that accepts tasks (map, reduce, and shuffle/sort) from the jobtracker daemon.

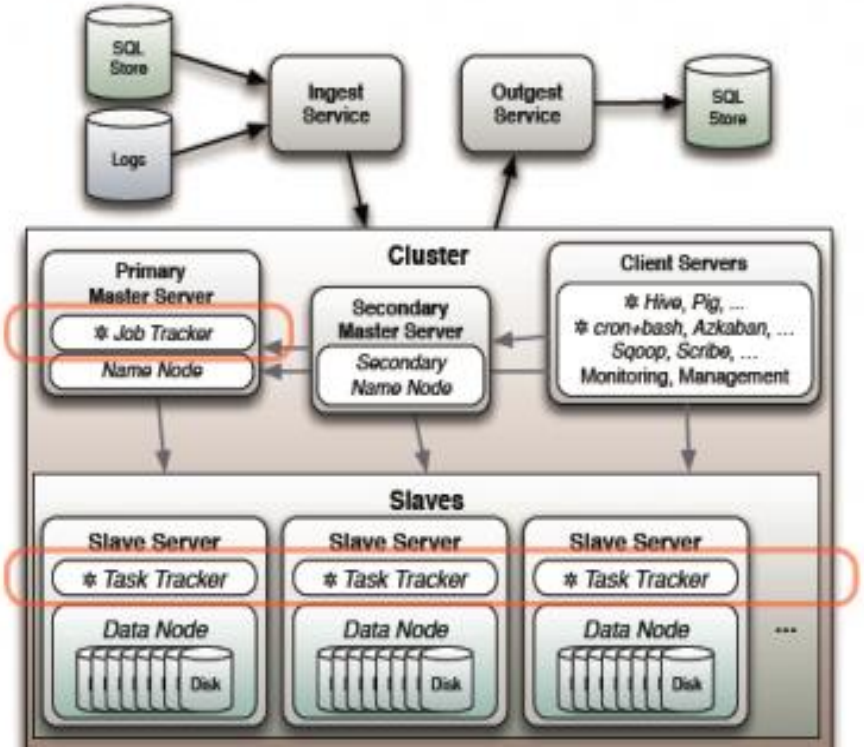
2. The tasktracker daemon is the daemon that performs the actual tasks during a MapReduce operation.

3. The tasktracker daemon sends a heartbeat message to jobtracker, periodically, to notify the jobtracker daemon that it is alive. Along with the heartbeat, it also sends the free slots available within it, to process tasks.

4. The tasktracker daemon starts and monitors the map, and reduces tasks and sends progress/status information back to the jobtracker daemon.

Jobs and Tasks

- Services
  - Job Tracker
  - Task Trackers



An advantage of using HDFS is data awareness between the job tracker and task tracker.

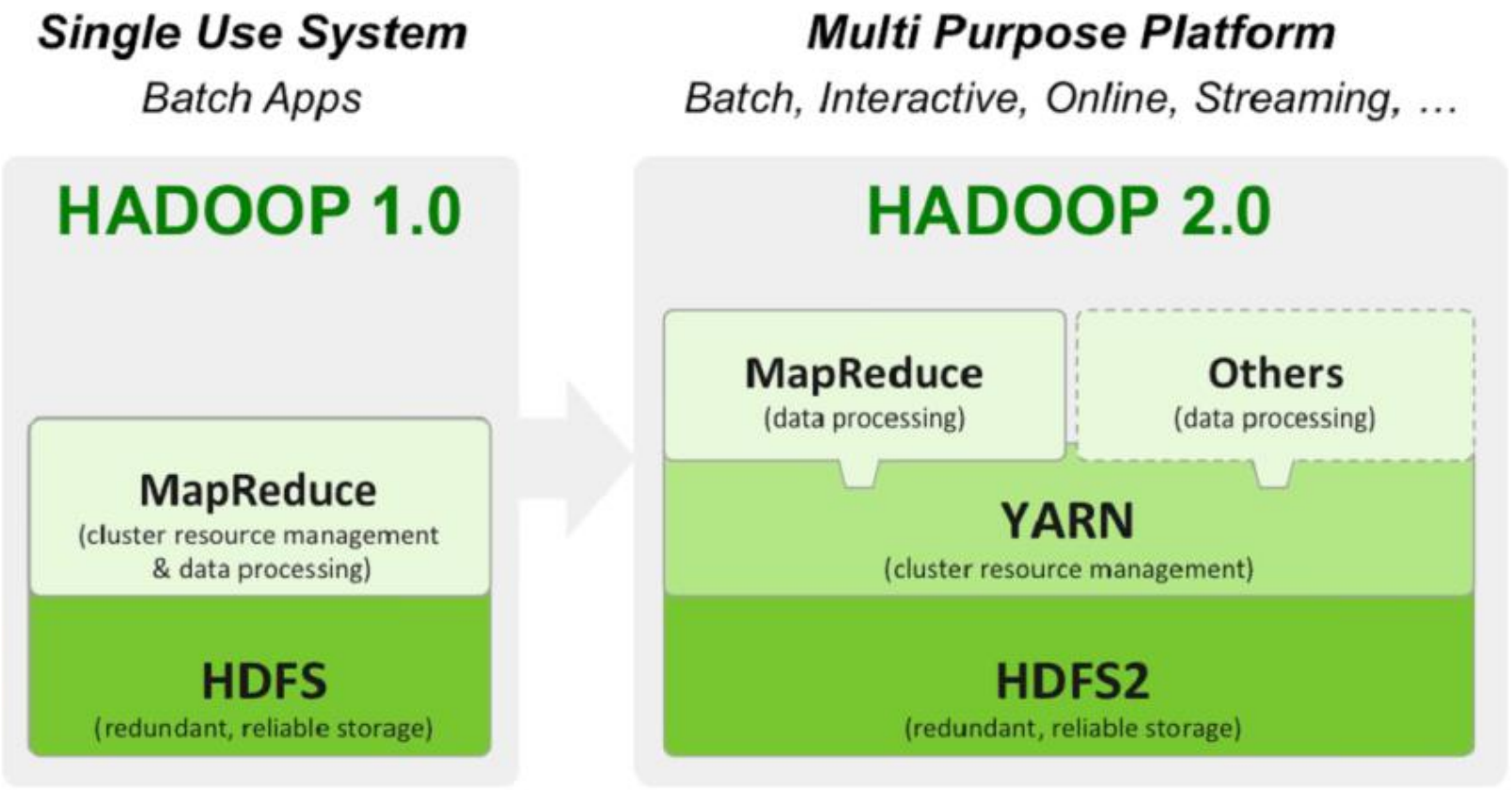
The job tracker schedules map or reduce jobs to task trackers with an awareness of the data location.

For example, if node A contains data (x, y, z) and node B contains data (a, b, c), the job tracker schedules node B to perform map or reduce tasks on (a,b,c) and node A would be scheduled to perform map or reduce tasks on (x,y,z). This reduces the amount of traffic that goes over the network and prevents unnecessary data transfer.

Nodes

- NameNode:
  - Master of the system
  - Maintains and manages the blocks which are present on the DataNodes
- DataNodes:
  - Slaves which are deployed on each machine and provide the actual storage
  - Responsible for serving read and write requests for the clients
- Secondary NameNode:
  - HouseKeeping Daemon
- Jobtracker:
  - takes care of all the job scheduling and assign tasks to Task Trackers.
- TaskTracker:
  - a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a jobtracker

Versions of Hadoop



Limitations of Hadoop 1.0 Architecture

In Hadoop1.0 HDFS and MapReduce are the Core Components and other components are build around the core.

1. Single NameNode is responsible for managing entire namespace for Hadoop Cluster.

2. It has a restricted processing model which is suitable for **batch-oriented MapReduce jobs.**

3. Hadoop MapReduce is not suitable for interactive analysis.

4. Hadoop 1.0 is not suitable for machine learning algorithms, graphs, and other memory intensive algorithms.

5. **MapReduce is responsible for cluster resource management and data processing.**

Lack of proper Resource Utilization-In this architecture, map slots might be full while reduce slots are empty and vise-versa

Hadoop 1.0 vs Hadoop 2.0

- Early adopters of the Hadoop ecosystem were restricted to processing models that were **MapReduce**-based only. **Hadoop 2 has brought with it effective processing models including interactive SQL queries over big data, analysis of Big Data scale graphs, and scalable machine learning abilities.**
- The difference between Hadoop 1.x and Hadoop 2.x is that the architecture went from being a single use system that only handled Batch jobs to **a multi-purpose system that could not only run the batch jobs that Hadoop 1.x did but also more interactive, online, streaming jobs as well.**
- In Hadoop 1.0-Mapreduce was used for Cluster management and Data Processiong. There was a lot of issues that this brought up including scalability limitations, availability, resource utilization, and more. **Hadoop 2.0 separate the cluster resource management work from MapReduce and built YARN to do it more efficiently and effectively.**
- In Hadoop 1.X, there is a single NameNode which is thus the single point of failure whereas**, in Hadoop 2.x, there are Active and Passive NameNodes. In case, the active NameNode fails, the passive NameNode replaces the active NameNode and takes the charge. As a result, high availability is there in Hadoop 2.x.**
- In Hadoop 1.0 they were called “jobs” and everything that was submitted to cluster for processing was a MapReduce Job. **Now with the ability to use not only MapReduce but also the others tools like Storm and Spark a job now is referred to as an “Application”.**

Hadoop 1 vs Hadoop 2

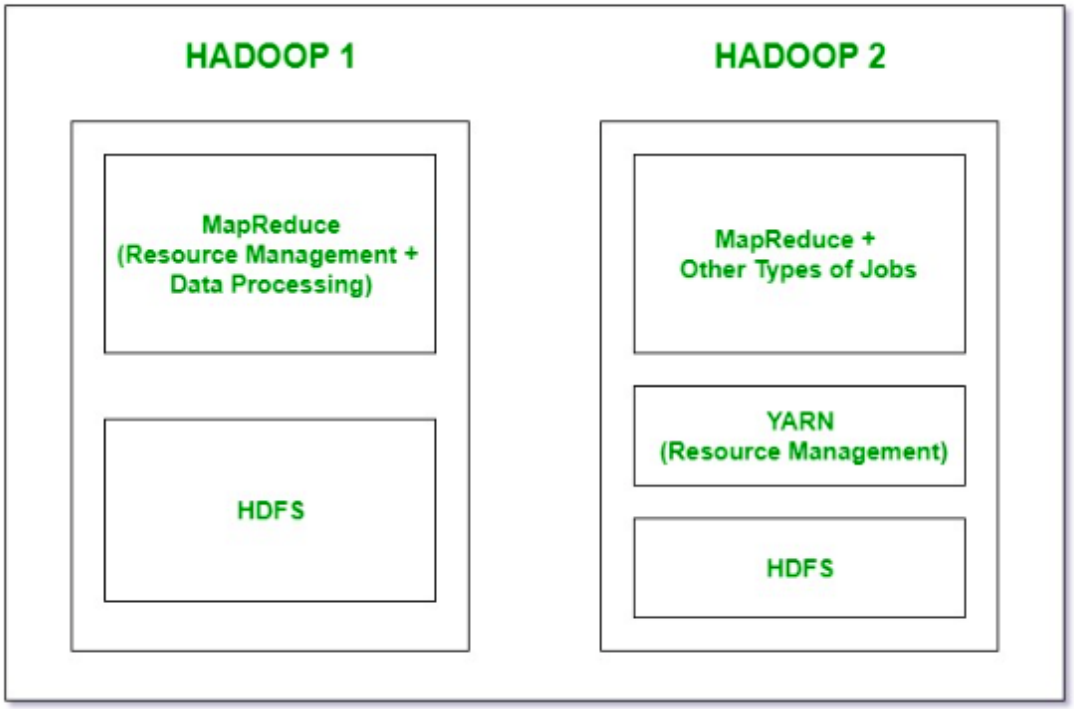
1. Components: In Hadoop 1 we have MapReduce but Hadoop 2 has YARN(Yet Another Resource Negotiator) and MapReduce version 2.

Hadoop 1	Hadoop 2
HDFS	HDFS
Map Reduce	YARN / MRv2

2. Daemons:

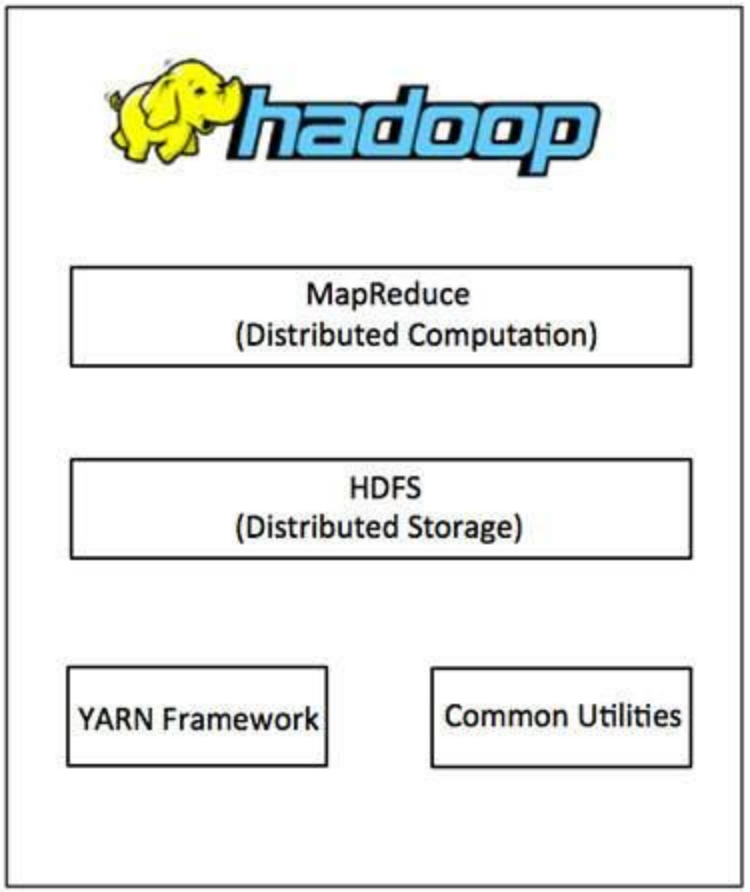
Hadoop 1	Hadoop 2
Namenode	Namenode
Datanode	Datanode
Secondary Namenode	Secondary Namenode
Job Tracker	Resource Manager
Task Tracker	Node Manager

- 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.



The Core Apache Hadoop Project

- Hadoop Common:
  - Java libraries and utilities required by other Hadoop modules.
- Hadoop YARN:
  - a framework for job scheduling and cluster resource management.
- HDFS:
  - A distributed file system
- Hadoop MapReduce:
  - YARN-based system for parallel processing of large data sets.



Hadoop 2 YARN: Taking Hadoop beyond Batch

The fundamental idea behind this architecture is splitting the **JobTracker** responsibility of resource management and Job Scheduling/Monitoring (Processing)into separate daemons. Daemons that are part of YARN Architecture are described below.

A Global Resource Manager -resource management **:** Its main responsibility is to distribute resources among various applications in the system. It has two main components:

NodeManager: This is a per-machine slave daemon. NodeManager responsibility is launching the application containers for application execution. NodeManager monitors the resource usage such as memory, CPU, disk, network, etc. It then reports the usage of resources to the global ResourceManager.

Per-application ApplicationMaster: This is an application-specific entity. Its responsibility is to negotiate required resources for execution from the ResourceManager. It works along with the NodeManager for executing and monitoring component tasks.

Answer a few quick questions…

Match the columns

Column A	Column B
HDFS	DataNode
MapReduce Programming	NameNode
Master node	Processing Data
Slave node	Google File System and MapReduce
Hadoop Implementation	Storage

Match the columns

Column A	Column B
JobTracker	Executes Task
MapReduce	Schedules Task
TaskTracker	Programming Model
Job Configuration	Converts input into Key Value pair
Map	Job Parameters

19CSE357-
Big Data Analytics(3-0-0-3)

Lecture #28,#29

HADOOP-MAPREDUCE

Chapter 5

Sangita Khare

Department of Computer Science & Engineering

Overall Process

Step-1 The client submits the job to Job Tracker

Step-2 Job Tracker asks Name node the location of data

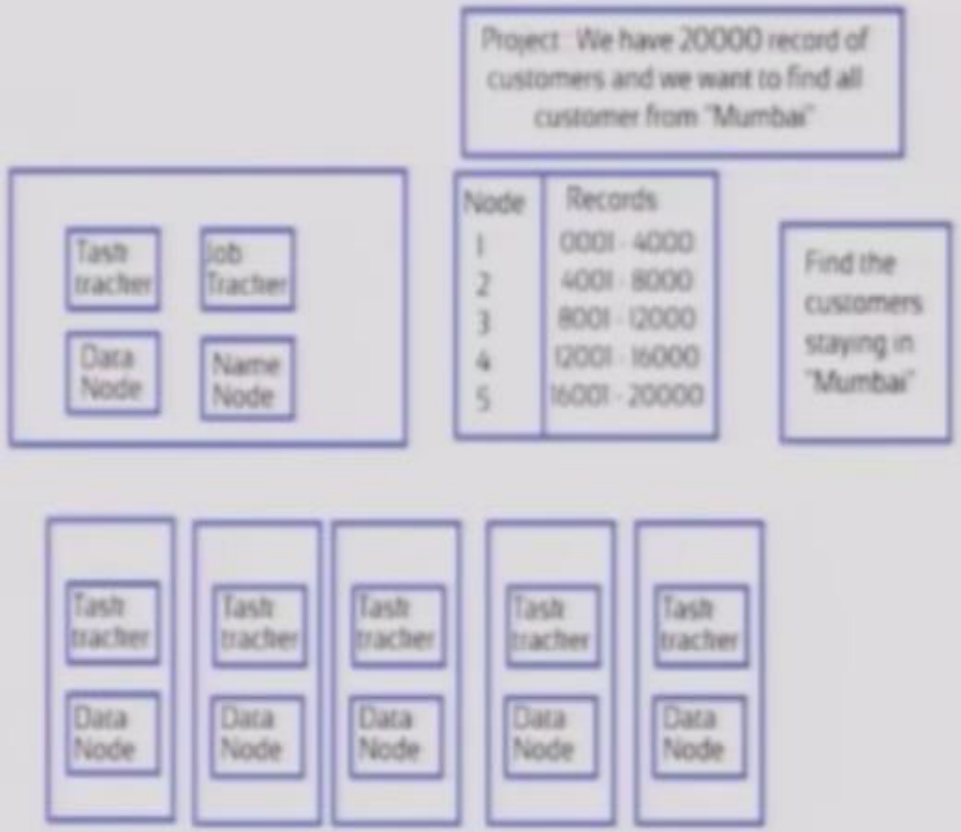
Step-3 As per the reply from name node, the Job Tracker ask respective task trackers to execute the task on their data

Step-4 All the results are stored on some Data Node and the Name Node is informed about the same.

Step-5 The task trackers inform the job completion and progress to Job Tracker

Step-6 The Job Tracker inform the completion to client

Step-7 Client contacts the Name Node and retrieve the results



Map-Reduce

- **MapReduce** is a programming model and an associated implementation for **processing and generating large data sets.**

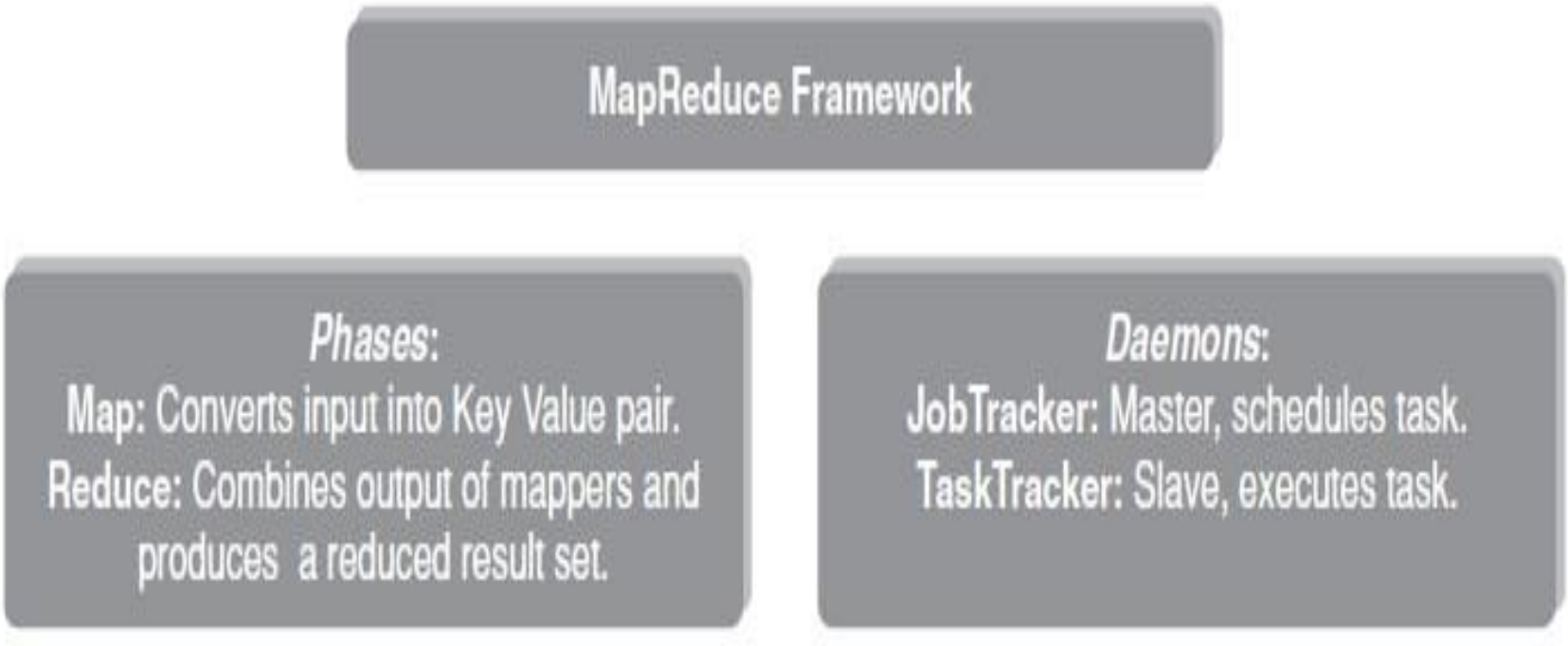
- Users specify a ***map*** function that processes a key/value pair to generate a set of intermediate key/value pairs, and a ***reduce*** function that merges all intermediate values associated with the same intermediate key.

- Many real world tasks are expressible in this model.

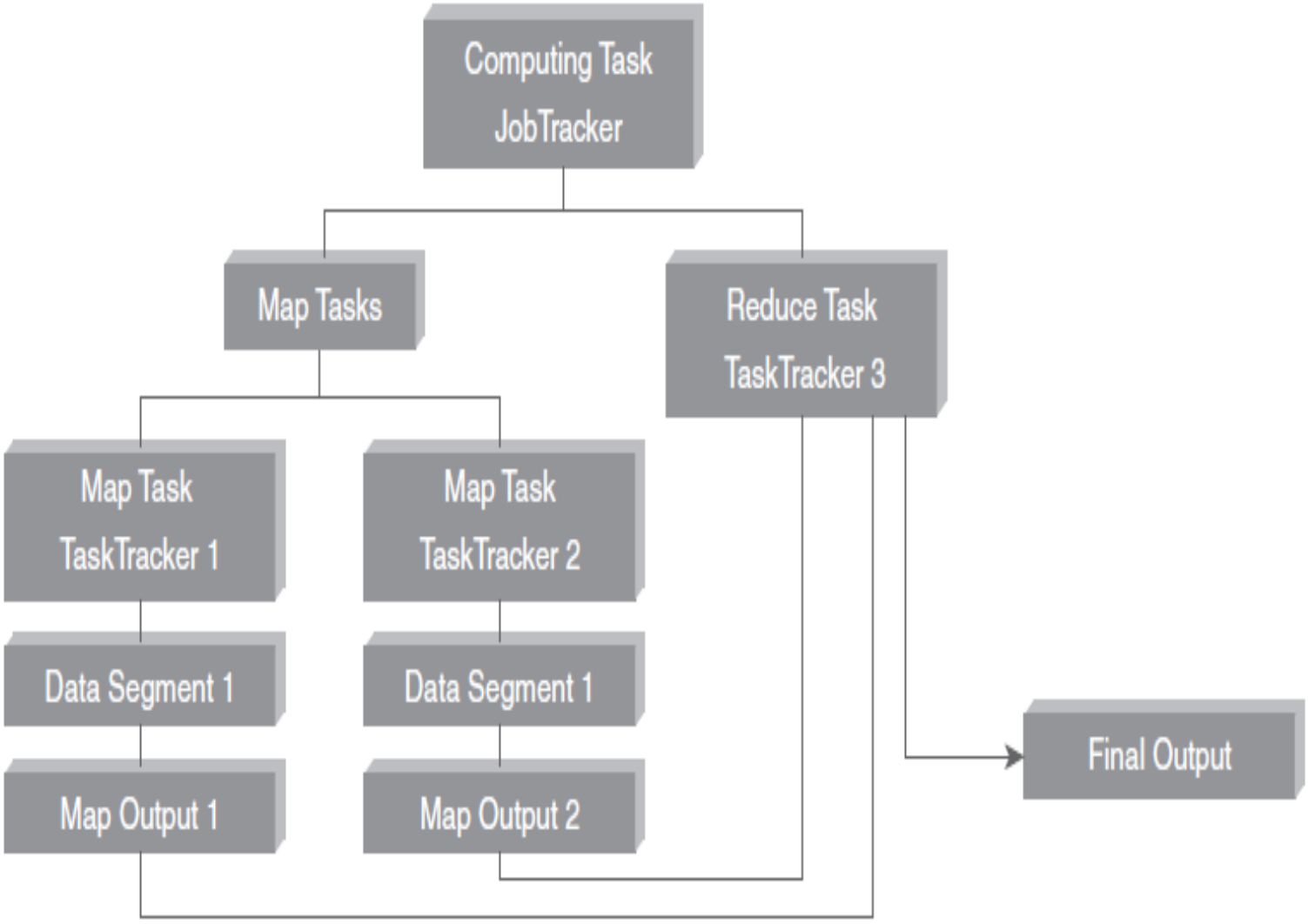
What is MapReduce Programming?

MapReduce Programming is a software framework.

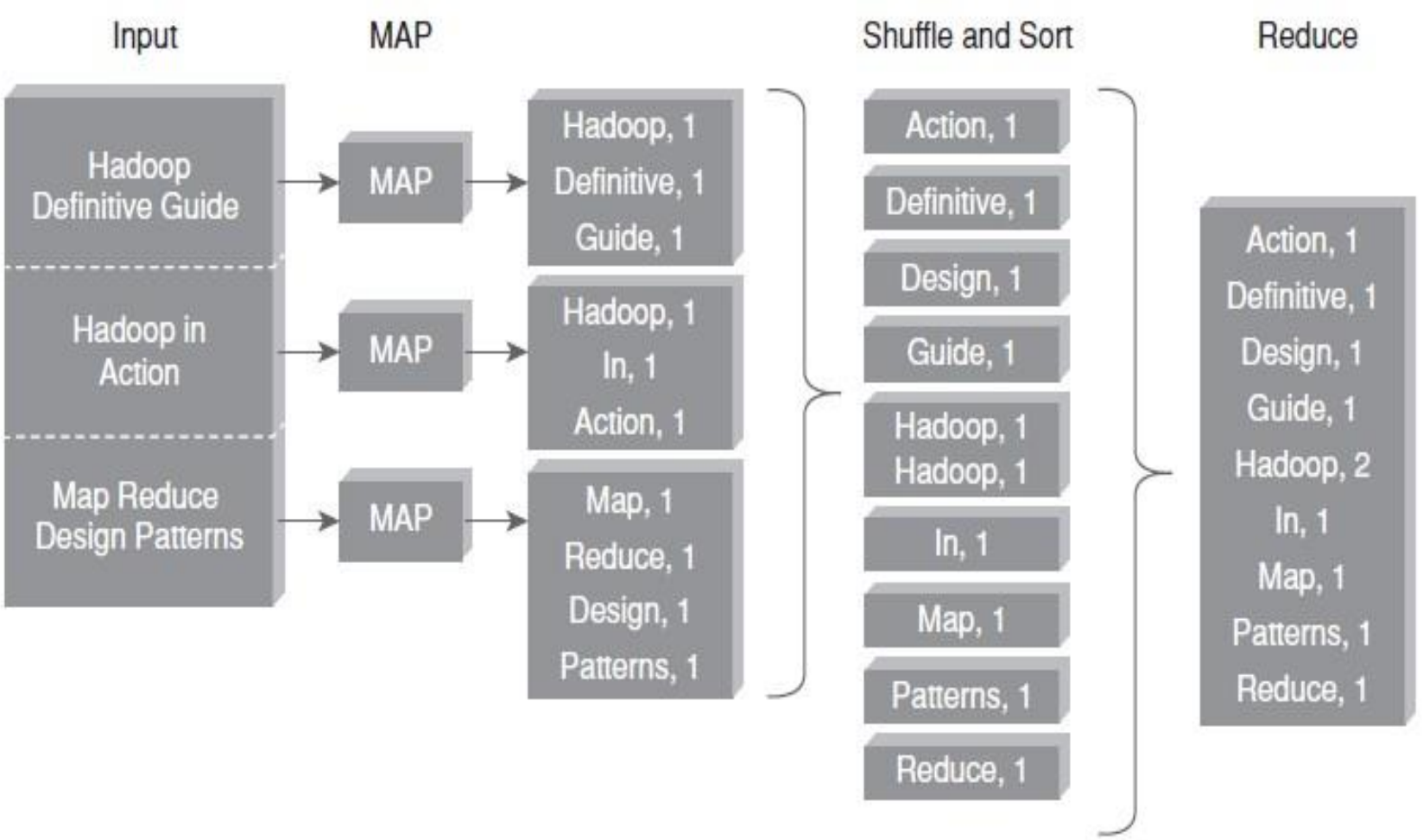
MapReduce Programming helps you to process massive amounts of data in parallel.



How MapReduce Programming Works



MapReduce – Word Count Example

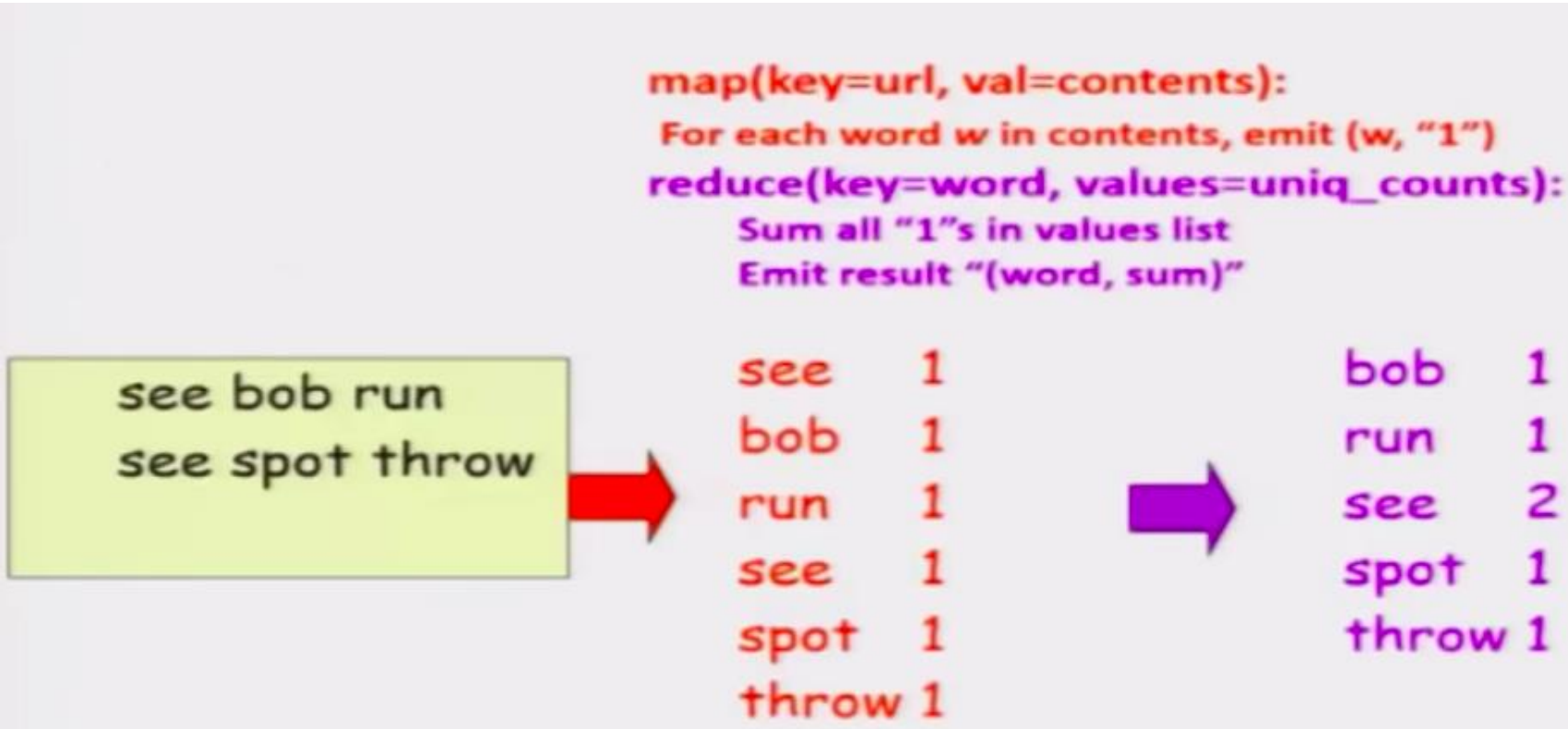


How Map-Reduce works?

In MapReduce Programming, Jobs (Applications) are split into a set of map tasks and reduce tasks. Then these tasks are executed in a distributed fashion on Hadoop cluster.

Each task processes small subset of data that has been assigned to it. This way, Hadoop distributes the load across the cluster.

MapReduce job takes a set of files that is stored in HDFS (Hadoop Distributed File System) as input.


map(key=url, val=contents):
For each word w in contents, emit (w, "1")
reduce(key=word, values=uniq_counts):
Sum all "1"s in values list
Emit result "(word, sum)"
see bob run
see spot throw
see 1
bob 1
run 1
see 1
spot 1
throw 1
bob 1
run 1
see 2
spot 1
throw 1

- Map task takes care of loading, parsing, transforming and filtering.
- Reduce task is grouping and aggregating data that is produced by map tasks to generate final output.
- Each Map-Reduce task is broken into the following phases:
    1. Record reader
    2. Mapper
    3. Combiner
    4. Partitioner/Reducer
    5. Record Writer

Map Task Phases

The following example provides a theoretical idea:
Let us assume we have the following input text file named **input.txt** for MapReduce.

What do you mean by Object
What do you know about Java
What is Java Virtual Machine
How Java enabled High Performance

Record Reader-Input Splits

- This is the first phase of MapReduce where the Record Reader reads every line from the input text file as text and yields output as key-value pairs. (it converts a byte oriented view of the input into a record oriented view)
  - **Input** − Line by line text from the input file.
  - **Output** − Forms the key-value pairs. The following is the set of expected key-value pairs.

```
<1, What do you mean by Object>
<2, What do you know about Java>
<3, What is Java Virtual Machine>
<4, How Java enabled High Performance>
```

Phase 1:Mapper

- The Map phase takes input from the Record Reader, processes it, and produces the output as another set of key-value pairs.
- Output of record reader will be taken as an input to this phase.
- The Map phase reads each key-value pair, divides each word from the value using String Tokenizer, treats each word as key and the count of that word as value.
  - **Output** − The expected output is as follows.

<What,1> <do,1> <you,1> <mean,1> <by,1> <Object,1>
<What,1> <do,1> <you,1> <know,1> <about,1> <Java,1>
<What,1> <is,1> <Java,1> <Virtual,1> <Machine,1>

<How,1> <Java,1> <enabled,1> <High,1> <Performance,1>

Combiner Phase

- A Combiner, also known as a **semi-reducer,** is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class.
- The main function of a Combiner is to summarize the map output records with the same key. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.
- The Combiner phase takes each key-value pair from the Map phase, processes it, and produces the output as **key-value collection** pairs.

Combiner Phase

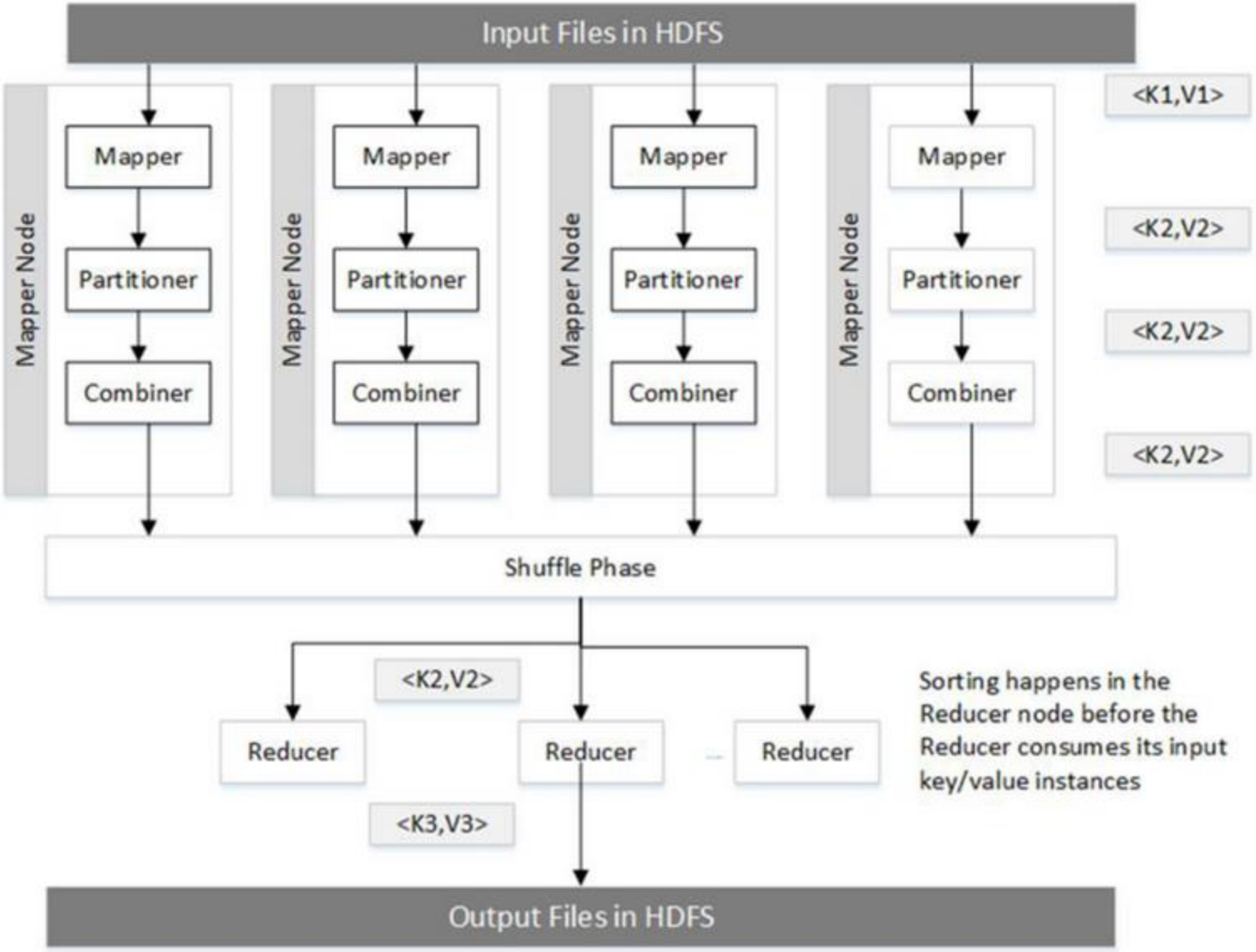
- The Combiner phase reads each key-value pair, combines the common words as key and values as collection. Usually, the code and operation for a Combiner is similar to that of a Reducer.
  - **Output** − The expected output is as follows

<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1> <Object,1>
<know,1> <about,1> <Java,1,1,1>
<is,1> <Virtual,1> <Machine,1>

<How,1> <enabled,1> <High,1> <Performance,1>

Partitioner Phase

- A partitioner works like a condition in processing an input dataset. The partition phase takes place after the Map phase and before the Reduce phase.
- The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers. Therefore, the data passed from a single partitioner is processed by a single Reducer.
- A partitioner partitions the key-value pairs of intermediate Map-outputs. It partitions the data using a user-defined condition, which works like a hash function. The total number of partitions is same as the number of Reducer tasks for the job


Input Files in HDFS
Mapper Node
Mapper
Partitioner
Combiner
Mapper Node
Mapper
Partitioner
Combiner
Mapper Node
Mapper
Partitioner
Combiner
Mapper Node
Mapper
Partitioner
Combiner
<K1,V1>
<K2,V2>
<K2,V2>
<K2,V2>
Shuffle Phase
<K2,V2>
Reducer
Reducer
Reducer
Sorting happens in the Reducer node before the Reducer consumes its input key/value instances
<K3,V3>
Output Files in HDFS

Phase 2: REDUCER

- The primary chore of reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values.
- The reducer has three primary phases:
  i. Shuffle and Sort
  ii. Reduce
  iii. Output format.

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** − The expected output from the Reducer phase is as follows

```
<What,3> <do,2> <you,2> <mean,1> <by,1> <Object,1>
<know,1> <about,1> <Java,3>
<is,1> <Virtual,1> <Machine,1>
<How,1> <enabled,1> <High,1> <Performance,1>
```

Record Writer

- This is the last phase of MapReduce where the Record Writer writes every key-value pair from the Reducer phase and sends the output as text.
- **Input** − Each key-value pair from the Reducer phase along with the Output format.
- **Output** − It gives you the key-value pairs in text format. Following is the expected output

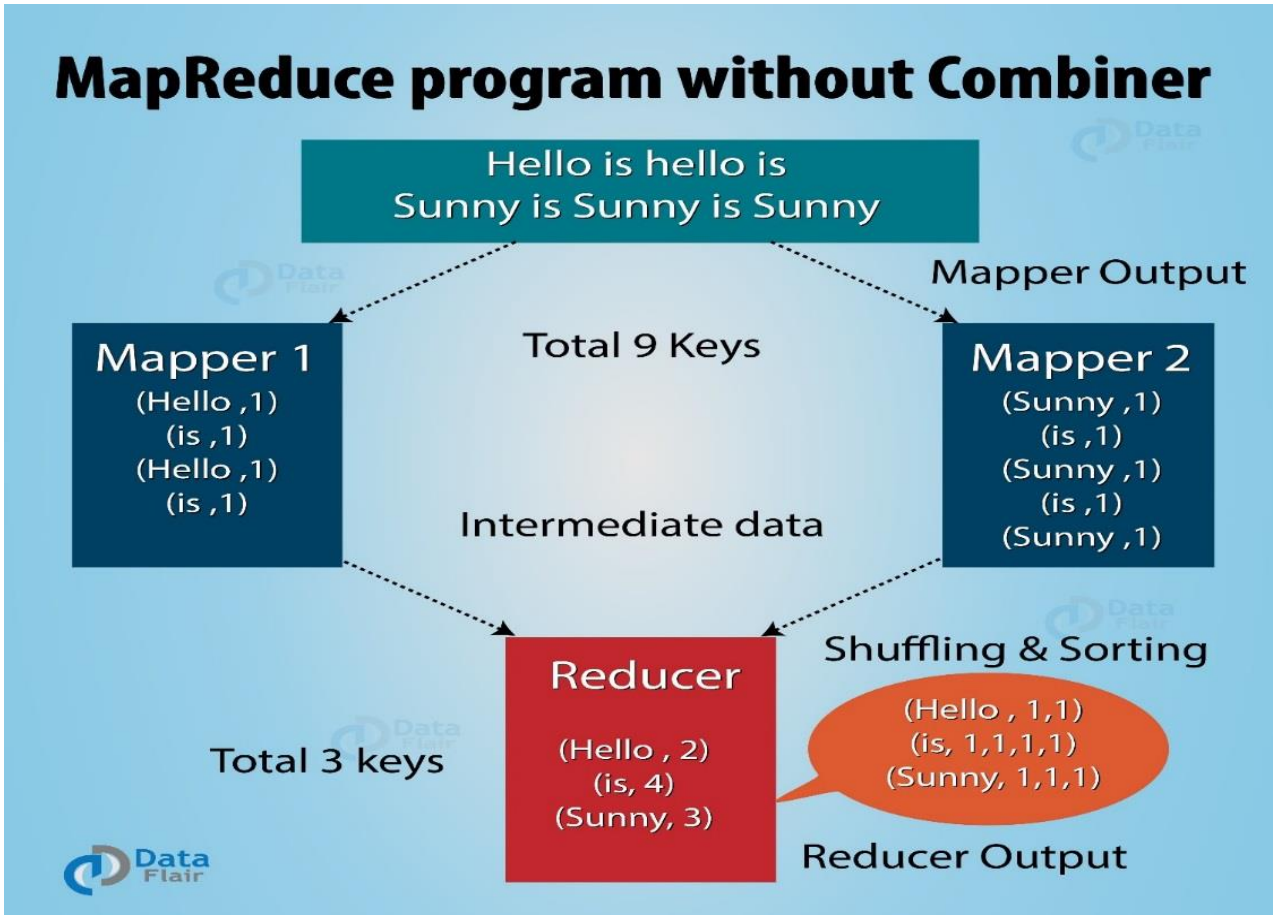
```
What        3
do          2
you         2
mean        1
by          1
Object      1
know        1
about       1

Java        3
```

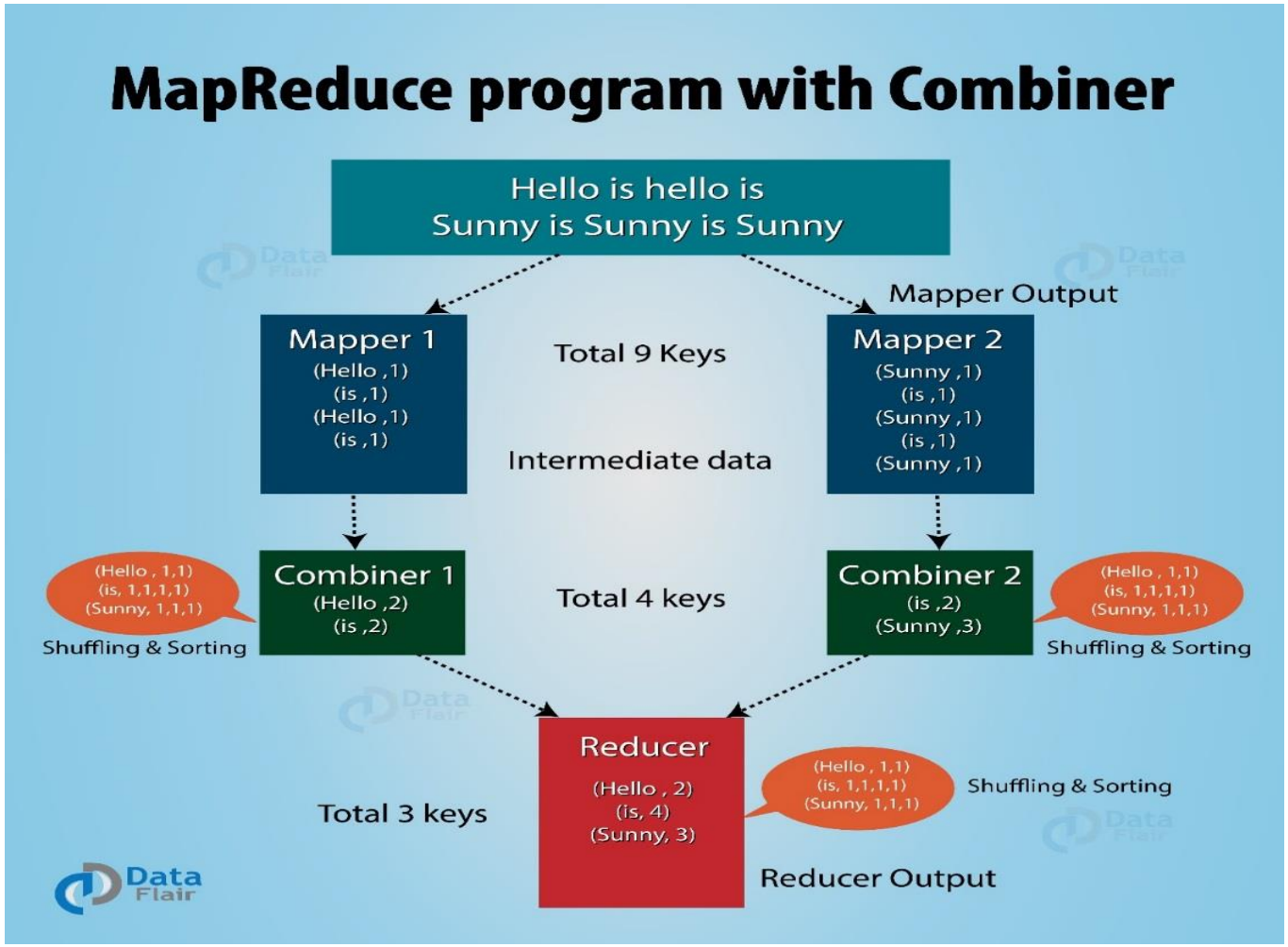
```
map(key, value):
// key: document name; value: text of document
    for each word w in value:
        emit(w, 1)


reduce(key, values):
// key: a word; values: an iterator over counts
        result = 0
        for each count v in values:
            result += v
        emit(key, result)
```

- **Input:** a set of key/value pairs
- User supplies two functions:

  map(k,v) → list(k1,v1)

  reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- **Output** is the set of (k1,v2) pairs


MapReduce program without Combiner
Hello is hello is
Sunny is Sunny is Sunny
Mapper Output
Mapper 1
(Hello ,1)
(is ,1)
(Hello ,1)
(is ,1)
Total 9 Keys
Intermediate data
Mapper 2
(Sunny ,1)
(is ,1)
(Sunny ,1)
(is ,1)
(Sunny ,1)
Shuffling & Sorting
Reducer
(Hello , 2)
(is, 4)
(Sunny, 3)
(Hello , 1,1)
(is, 1,1,1,1)
(Sunny, 1,1,1)
Total 3 keys
Reducer Output
Data Flair


MapReduce program with Combiner
Hello is hello is
Sunny is Sunny is Sunny
Mapper Output
Mapper 1
(Hello ,1)
(is ,1)
(Hello ,1)
(is ,1)
Total 9 Keys
Intermediate data
Mapper 2
(Sunny ,1)
(is ,1)
(Sunny ,1)
(is ,1)
(Sunny ,1)
(Hello , 1,1)
(is, 1,1,1,1)
(Sunny, 1,1,1)
Shuffling & Sorting
Combiner 1
(Hello ,2)
(is ,2)
Total 4 keys
Combiner 2
(is ,2)
(Sunny ,3)
(Hello , 1,1)
(is, 1,1,1,1)
(Sunny, 1,1,1)
Shuffling & Sorting
Reducer
(Hello , 2)
(is, 4)
(Sunny, 3)
Total 3 keys
(Hello , 1,1)
(is, 1,1,1,1)
(Sunny, 1,1,1)
Shuffling & Sorting
Reducer Output
Data Flair

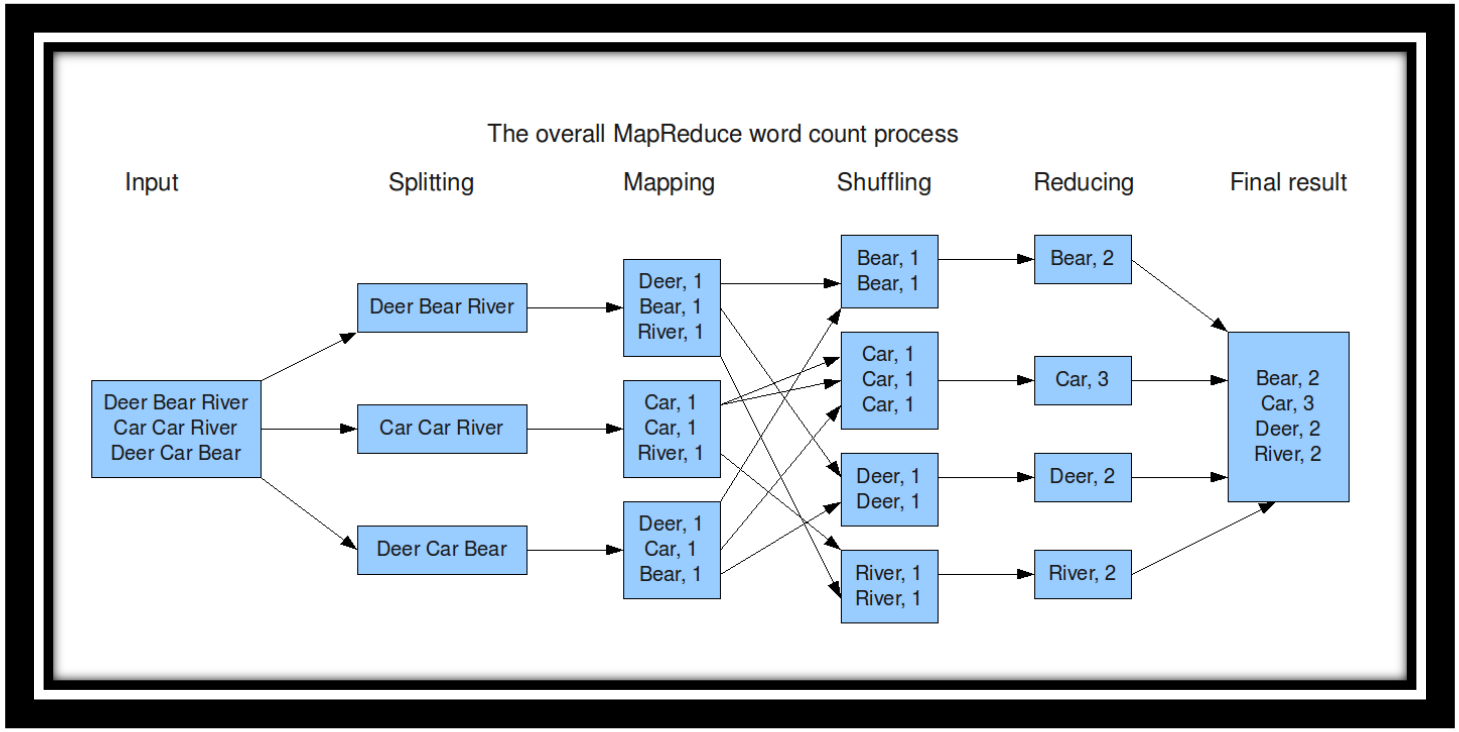
Advantages of MapReduce Combiner

- Hadoop Combiner reduces the time taken for data transfer between mapper and reducer.
- It decreases the amount of data that needed to be processed by the reducer.
- The Combiner improves the overall performance of the reducer.

Disadvantages of Hadoop combiner in MapReduce

- MapReduce jobs cannot depend on the Hadoop combiner execution because there is no guarantee in its execution.
- In the local filesystem, the key-value pairs are stored in the Hadoop and run the combiner later which will cause expensive disk IO.

MapReduce: Word Count



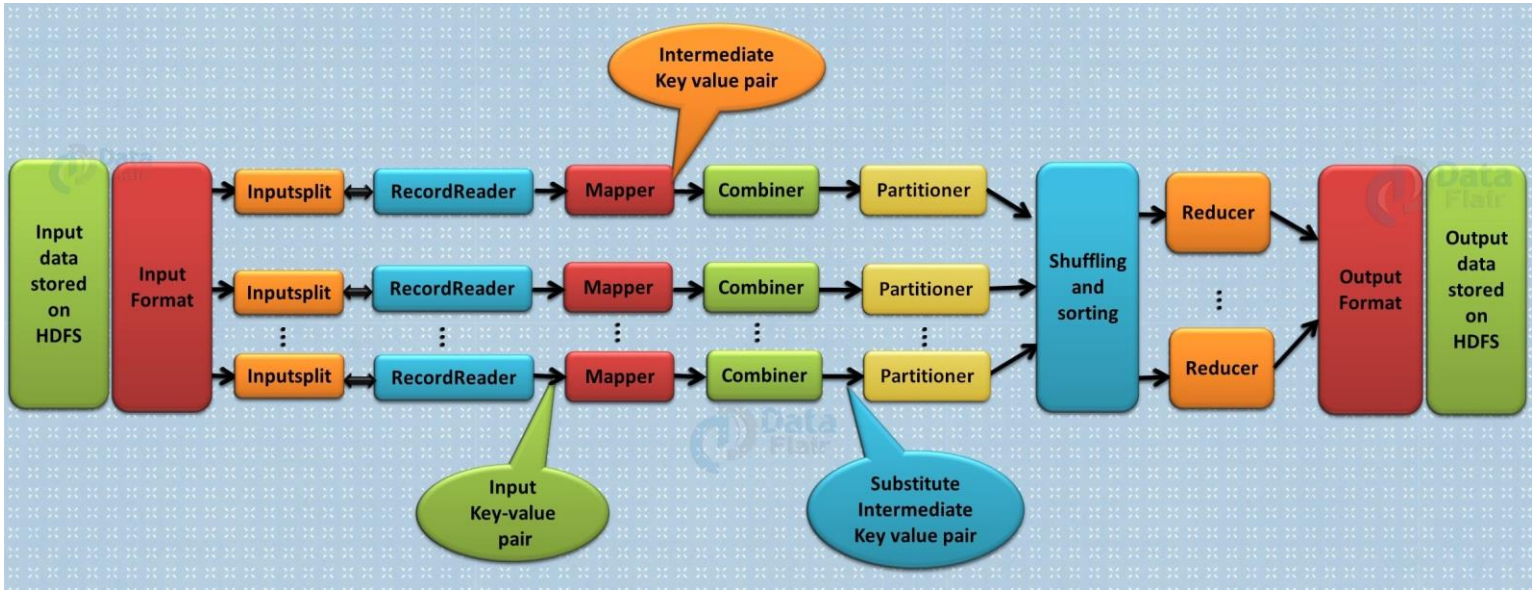
19CSE357-
Big Data Analytics(3-0-0-3)

Lecture #30

HADOOP-YARN

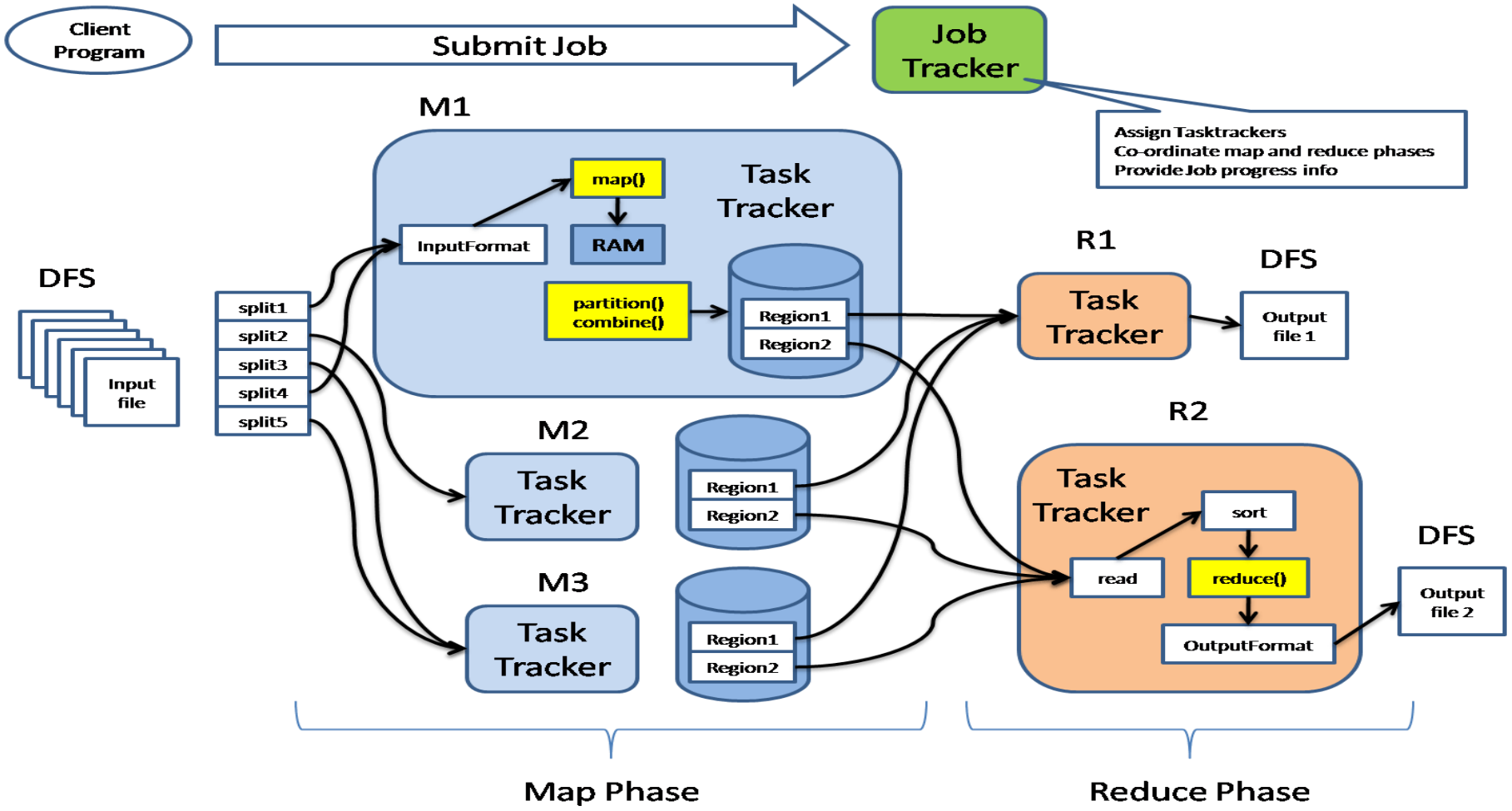
Chapter 5

- **Sangita Khare**
- Department of Computer Science & Engineering



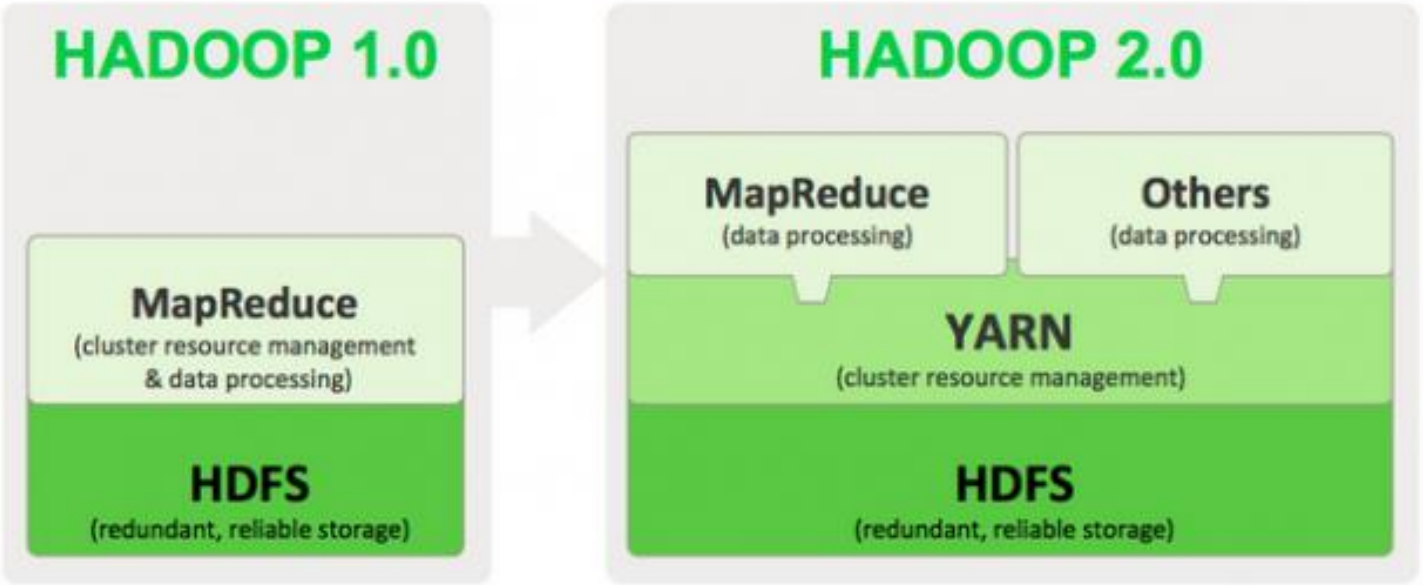
Source https://www.section.io/engineering-education/understanding-map-reduce-in-hadoop/

The following diagram shows a MapReduce architecture.



MANAGING RESOURCES AND APPLICATIONS WITH HADOOP - YARN

(YET ANOTHER RESOURCE NEGOTIATOR)



As part of Hadoop 2.0, YARN takes the resource management capabilities that were in MapReduce and packages them so they can be used by new engines.

This also streamlines MapReduce to do what it does best, process data.

With YARN, we can now run multiple applications in Hadoop, all sharing a common resource management.


Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

The YARN Scheduler

- Used in Hadoop 2.x +
- YARN = Yet Another Resource Negotiator
- Treats each server as a collection of *containers*
  - Container = fixed CPU + fixed memory
- Has 3 main components
  - Global *Resource Manager (RM)*
    - Scheduling
  - Per-server *Node Manager (NM)*
    - Daemon and server-specific functions
  - Per-application (job) *Application Master (AM)*
    - Container negotiation with RM and NMs
    - Detecting task failures of that job

Hadoop 2 YARN: Taking Hadoop beyond Batch

The fundamental idea behind this architecture is splitting the JobTracker responsibility of Resource management and Job Scheduling/Monitoring into separate daemons. Daemons that are part of YARN Architecture are described below.

1. A Global ResourceManager: Its main responsibility is to distribute resources among various applications in the system. It has two main components: Scheduler and AppManager

2. NodeManager: This is a per-machine slave daemon. NodeManager responsibility is launching the application containers for application execution. NodeManager monitors the resource usage such as memory, CPU, disk, network, etc. It then reports the usage of resources to the global ResourceManager.[horizontal scalability]

3. Per-application ApplicationMaster: This is an application-specific entity. Its responsibility is to negotiate required resources for execution from the ResourceManager. It works along with the NodeManager for executing and monitoring component tasks.

Global Resource Manager

Resource Manager is the master daemon of YARN. It is responsible for managing several other applications, along with the global assignments of resources such as CPU and memory. It is used for job scheduling. Resource Manager has two components:

- **Scheduler:** Schedulers' task is to distribute resources to the running applications. It only deals with the scheduling of tasks and hence it performs no tracking and no monitoring of applications.
- **Application Manager:** The application Manager manages applications running in the cluster.
  1. Accepting Job Submissions
  2. Tasks, such as the starting of Application Master or monitoring, are done by the Application Manager.Negotiating containers for executing application specific Application Master.
  3. Restarting Application Master in case of failure.

Node Manager

- Node Manager is the slave daemon of YARN. It has the following responsibilities:
- Node Manager has to monitor the container's resource usage, along with reporting it to the Resource Manager.
- The health of the node on which YARN is running is tracked by the Node Manager.
- It takes care of each node in the cluster while managing the workflow, along with user jobs on a particular node.
- It keeps the data in the Resource Manager updated
- Node Manager can also destroy or kill the container if it gets an order from the Resource Manager to do so.

Application Master

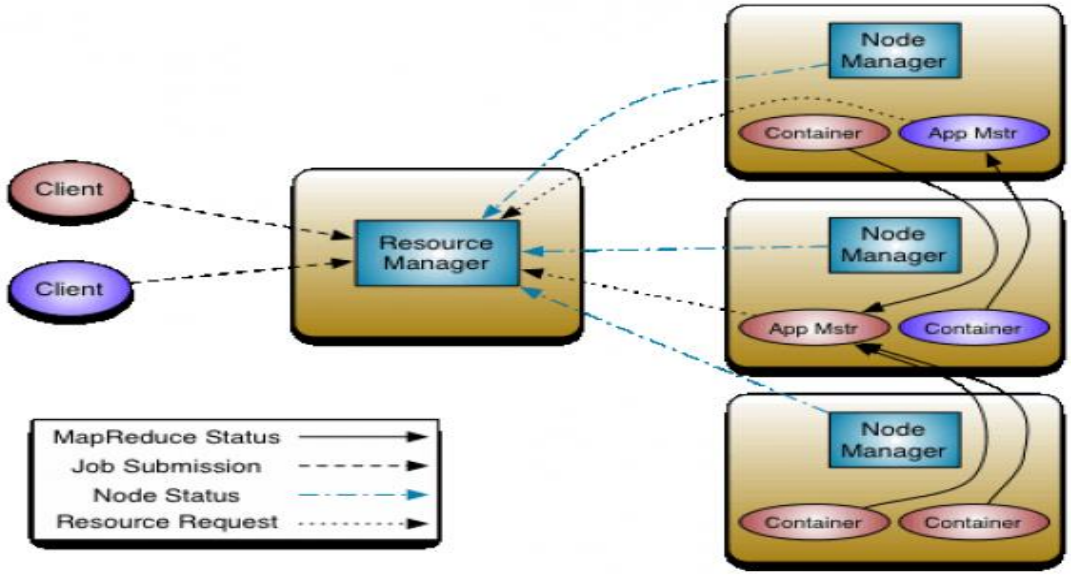
- Every job submitted to the framework is an application, and every application has a specific Application Master associated with it. Application Master performs the following tasks:
- It coordinates the execution of the application in the cluster, along with managing the faults.
- It negotiates resources from the Resource Manager.
- It works with the Node Manager for executing and monitoring other components' tasks.
- At regular intervals, heartbeats are sent to the Resource Manager for checking its health, along with updating records according to its resource demands.

What YARN does

YARN enhances the power of a Hadoop compute cluster in the following ways:

- **Scalability**: The processing power in data centers continues to grow quickly. Because YARN ResourceManager focuses exclusively on scheduling, it can manage those larger clusters much more easily. The scheduler in Resource manager of YARN architecture allows Hadoop to extend and manage thousands of nodes and clusters.
- **Compatibility with MapReduce**: Existing MapReduce applications and users can run on top of YARN without disruption to their existing processes.
- **Improved cluster utilization**: The ResourceManager is a pure scheduler that optimizes cluster utilization according to criteria such as capacity guarantees, fairness, and SLAs. Also, unlike before, there are no named map and reduce slots, which helps to better utilize cluster resources. Since YARN supports Dynamic utilization of cluster in Hadoop, which enables optimized Cluster Utilization.
- **Support for workloads other than MapReduce**: Additional programming models such as graph processing and iterative modeling are now possible for data processing.

How YARN works??



The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker/TaskTracker into separate entities:

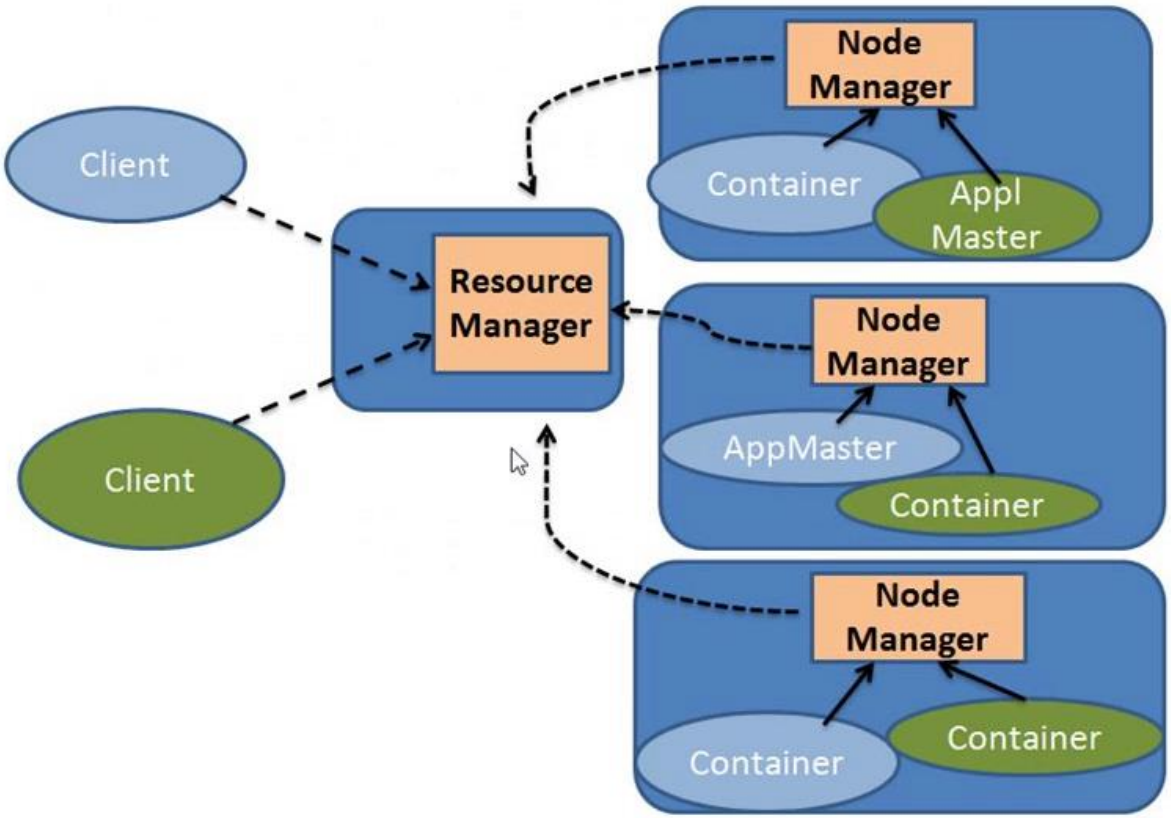
- a global ResourceManager
- a per-application ApplicationMaster
- a per-node slave NodeManager and
- a per-application container running on a NodeManager

The ResourceManager and the NodeManager form the new, and generic, system for managing applications in a distributed manner.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

Application is a job submitted to the framework. (map-reduce job)

Container is basic unit of allocation(Memory+CPU+Disk+N/W….).Replaces fixed Map reduce slots



✓Job tracker 1.0 responsibility is now split

- **Resource Manager** manages the resource allocation in the cluster
- **Application master** manages resource needs of individual applications

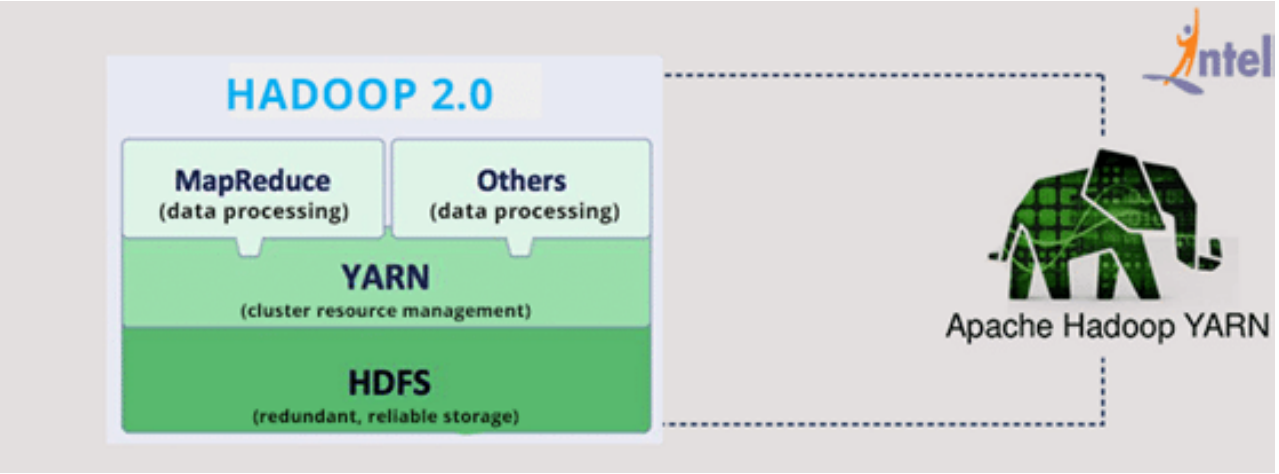
✓Node Manager is a generalized task tracker

✓A container executes an application specific process

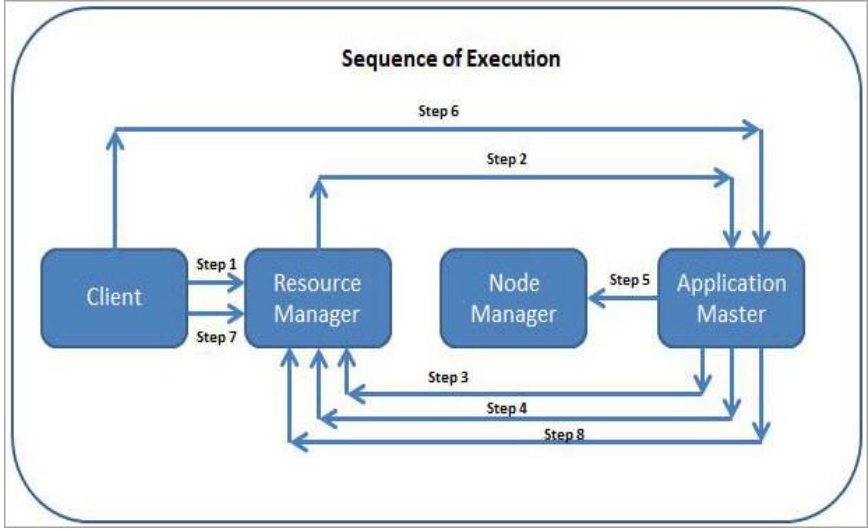
- The **ResourceManager** and the **NodeManager** formed the new generic system for managing applications in a distributed manner. The ResourceManager is the ultimate authority that arbitrates resources among all applications in the system. The **ApplicationMaster** is a framework-specific entity that negotiates resources from the ResourceManager and works with the NodeManager(s) to execute and monitor the component tasks.
- The ResourceManager has a scheduler, which is responsible for allocating resources to the various applications running in the cluster, according to constraints such as queue capacities and user limits. The scheduler schedules based on the resource requirements of each application.
- Each ApplicationMaster has responsibility for negotiating appropriate resource containers from the scheduler, tracking their status, and monitoring their progress. From the system perspective, the ApplicationMaster runs as a normal container.
- The NodeManager is the per-machine slave, which is responsible for launching the applications' containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager.

How does Apache Hadoop YARN work?

YARN separates HDFS and MapReduce, making the Hadoop environment more suitable for applications that can't wait for the batch processing jobs to get finished. So, no more batch processing delays with YARN! This architecture lets you process data with multiple processing engines using real-time streaming, interactive SQL, batch processing, handling of data stored in a single platform, and working with analytics in a completely different manner. It can be considered as the basis of the next generation of the Hadoop ecosystem, ensuring that the forward-thinking organizations are realizing the modern data architecture.



Workflow of an Application in YARN



1. Submission of the application by Client
2. Container allocation for starting Application Master by Global Resource Manager.
3. Registering the Application Master with Global Resource Manager on boot-up. This helps client program to query Resource Manager directly for details.
4. Application Master asks for containers from Global Resource Manager(resource- request protocol)
5. Application Master notifies Node Manager to launch containers

   Application code gets executed in the container(by Node Manager)

6,7 Client contacts Resource Manager/Application Master directly to monitor the status of the application

8. On completion of work Application Master gets disconnected with Resource Manager(deregisters ,freeing containers)

YARN vs MapReduce

In Hadoop 1.x, the batch processing framework MapReduce was closely paired with HDFS. With the addition of YARN to these two components, giving birth to Hadoop 2.x, came a lot of differences in how Hadoop worked.

Criteria	YARN	MapReduce
Type of processing	Real-time, batch, and interactive processing with multiple engines	Silo and batch processing with a single-engine
Cluster resource optimization	Excellent due to central resource management	Average due to fixed Map and Reduce slots
Suitable for	MapReduce and non-MapReduce applications	Only MapReduce applications
Managing cluster resource	Done by YARN	Done by JobTracker
Namespace	Hadoop supports multiple namespaces	Supports only one namespace, i.e., HDFS

Interacting with Hadoop Ecosystem

Other Hadoop components in Ecosystem

HBase	Hadoop database for random read/write access
Hive	SQL-like queries and tables on large datasets
Pig	Data flow language and compiler
Oozie	Workflow scheduler system for interdependent Hadoop jobs
Sqoop	Transfer bulk data between Hadoop and structured data source.Integration of databases and data warehouses with Hadoop
Flume	Configurable streaming data collection
ZooKeeper	Coordination service for distributed applications

Apache Hadoop Ecosystem

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop Data Exchange

Flume Log Collector

Zookeeper Coordination

Oozie Workflow

Pig Scripting

Mahout Machine Learning

R Connectors Statistics

Hive SQL Query

Hbase Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System

Hbase

- HBase is an open source, non-relational, distributed database modeled after Google's BigTable(compares well with RDBMS).
- It runs on top of Hadoop and HDFS, providing BigTable-like capabilities for Hadoop.
- It is column oriented NoSQL database.
- It provides random read/write operations.
- It supports record level updates which is not possible using HDFS.

Hbase in CAP theorem

- Eric Brewer’s CAP theorem, HBase is a CP type system.

When to use Hbase

- When there is real big data: millions or billions of rows, in other way data can not store in a single node.
- When random read/write access to big data
- When require to do thousands of operations on big data
- When there is no need of extra features of RDMS like typed columns, secondary indexes, transactions, advanced query languages, etc.
- When there is enough hardware.

Hive

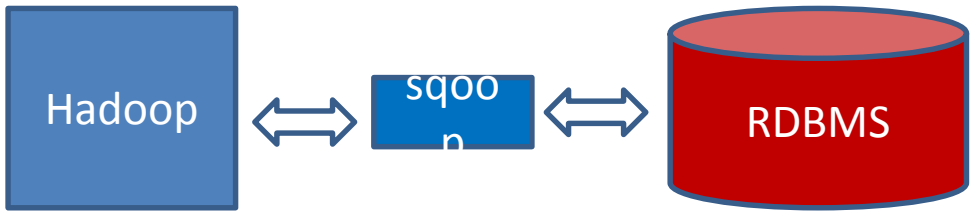
- An sql like interface to Hadoop.
- Data warehouse infrastructure built on top of Hadoop
- Provide data summarization, query and analysis
- Query execution via MapReduce
- Hive interpreter convert the query to Map reduce format.
- Open source project.
- Developed by Facebook
- Also used by Netflix, Cnet, Digg, eHarmony etc.

Pig

- A scripting platform for processing and analyzing large data sets
- Apache Pig allows to write complex MapReduce programs using a simple scripting language.
- High level language: Pig Latin
- Pig Latin is data flow language.
- Pig translate Pig Latin script into MapReduce to execute within Hadoop.
- Open source project
- Developed by Yahoo

Sqoop

- Command-line interface for transforming data between relational database and Hadoop
- Support incremental imports
- Imports use to populate tables in Hadoop
- Exports use to put data from Hadoop into relational database such as SQL server



Zookeeper

- Because coordinating distributed systems is a Zoo.
- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Interacting with Hadoop Ecosytem-summary

Pig : Pig is a data flow system for Hadoop. It uses Pig Latin to specify data flow. Pig is an alternative to MapReduce Programming. It abstracts some details and allows you to focus on data processing.

Hive: Hive is a Data Warehousing Layer on top of Hadoop. Analysis and queries can be done using an SQL-like language. Hive can be used to do ad-hoc queries, summarization, and data analysis. Figure 5.31 depicts Hive in the Hadoop ecosystem.

Sqoop: Sqoop is a tool which helps to transfer data between Hadoop and Relational Databases. With the help of Sqoop, you can import data from RDBMS to HDFS and vice-versa. Figure 5.32 depicts the Sqoop in Hadoop ecosystem.

HBase: HBase is a NoSQL database for Hadoop. HBase is column-oriented NoSQL database. HBase is used to store **billions of rows and millions of columns.** HBase provides random read/write operation. It also supports record level updates which is not possible using HDFS. HBase sits on top of HDFS.

What makes Hadoop unique

- Moving computation to data, instead of moving data to computation.
- Simplified programming model: allows user to quickly write and test
- Automatic distribution of data and work across machines

Match the columns

Column A

- HDFS
- MapReduce Programming
- Master node
- Slave node
- Hadoop Implementation

Column B

- DataNode
- NameNode
- Processing Data
- Google File System and MapReduce
- Storage

Revise

1. Hadoop is a_____based flat structure.
2. Hadoop can be deployed on_____.
3. HDFS cluster consists of single_____and number of _____.
4. Namenode uses _____ to store filesystem namespace.
5. Namenode uses _____ to record every transaction.
6. Secondary namenode is a_____ daemon.
7. Datanode is responsible for_____ file operation.
8. Hadoop has _____ architecture.
9. Hadoop is buid using_____language.
10. The number of copies of a file is called_____ of that file.
11. Namenode periodically receives _____ & ____ from each datanode in the cluster.
12. A typical block size by HDFS_____