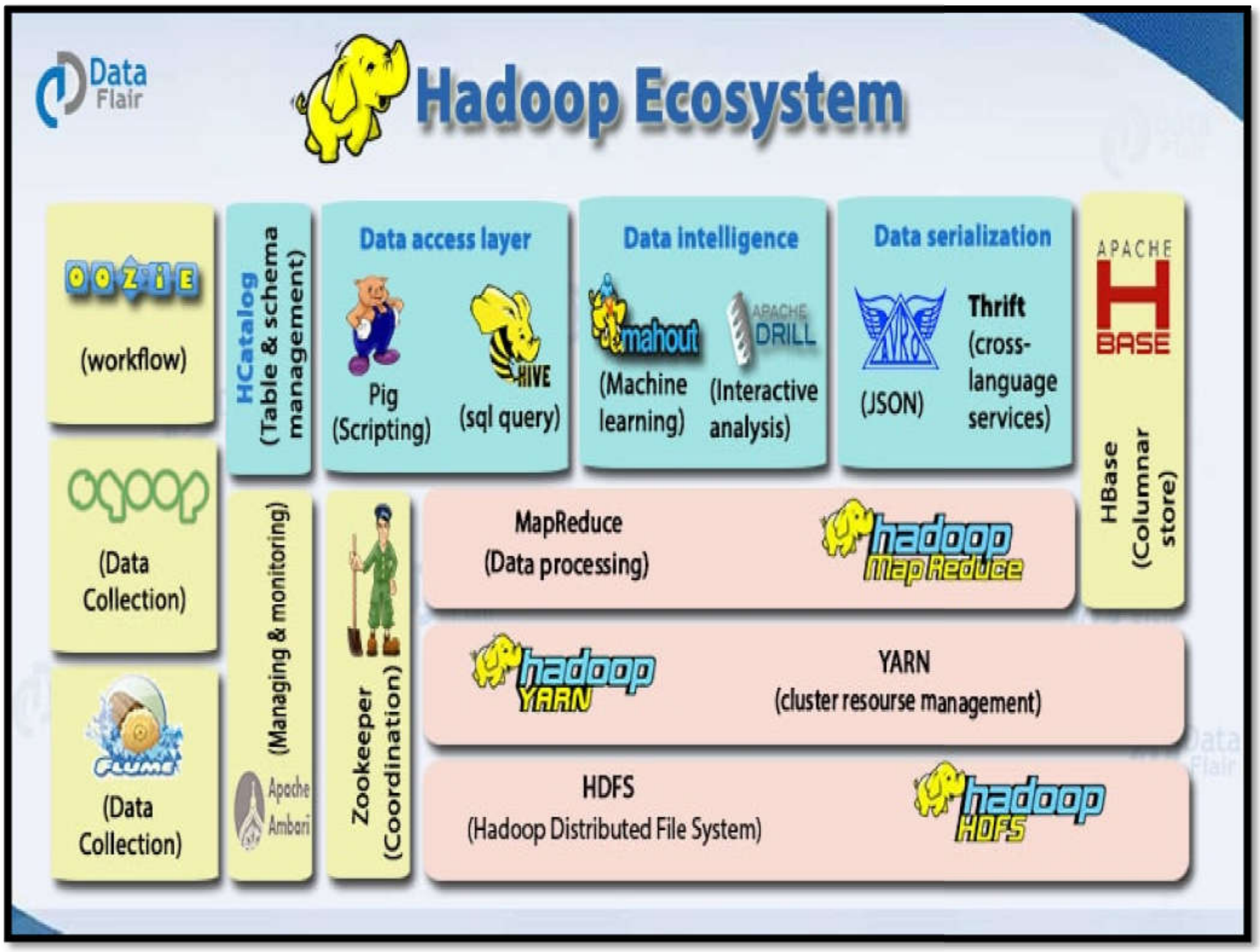
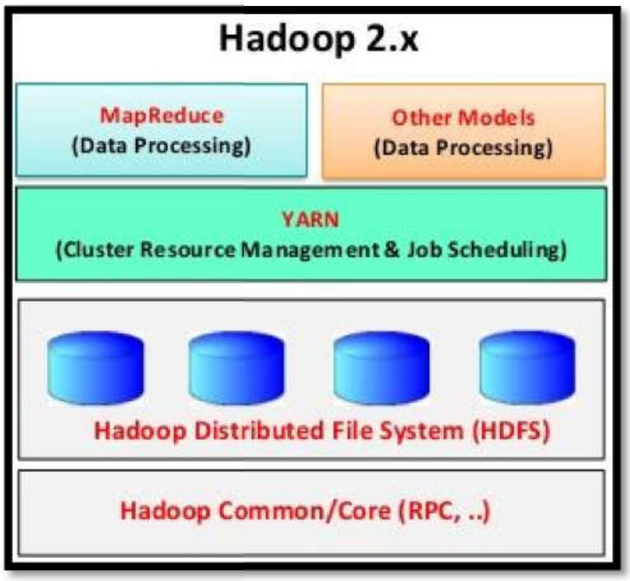
Apache Hadoop 2.x Components

The different components of **"Hadoop ecosystem"** that make Hadoop so powerful and due to which several Hadoop job roles are available are as follows:



Major Components of Hadoop Ecosystem are:

- Hadoop Common
- Hadoop Distributed File System (HDFS)
- Hadoop YARN (Yet Another Resource Negotiator)
- Hadoop MapReduce



Hadoop 2.x
MapReduce (Data Processing)
Other Models (Data Processing)
YARN (Cluster Resource Management & Job Scheduling)
Hadoop Distributed File System (HDFS)
Hadoop Common/Core (RPC, ..)

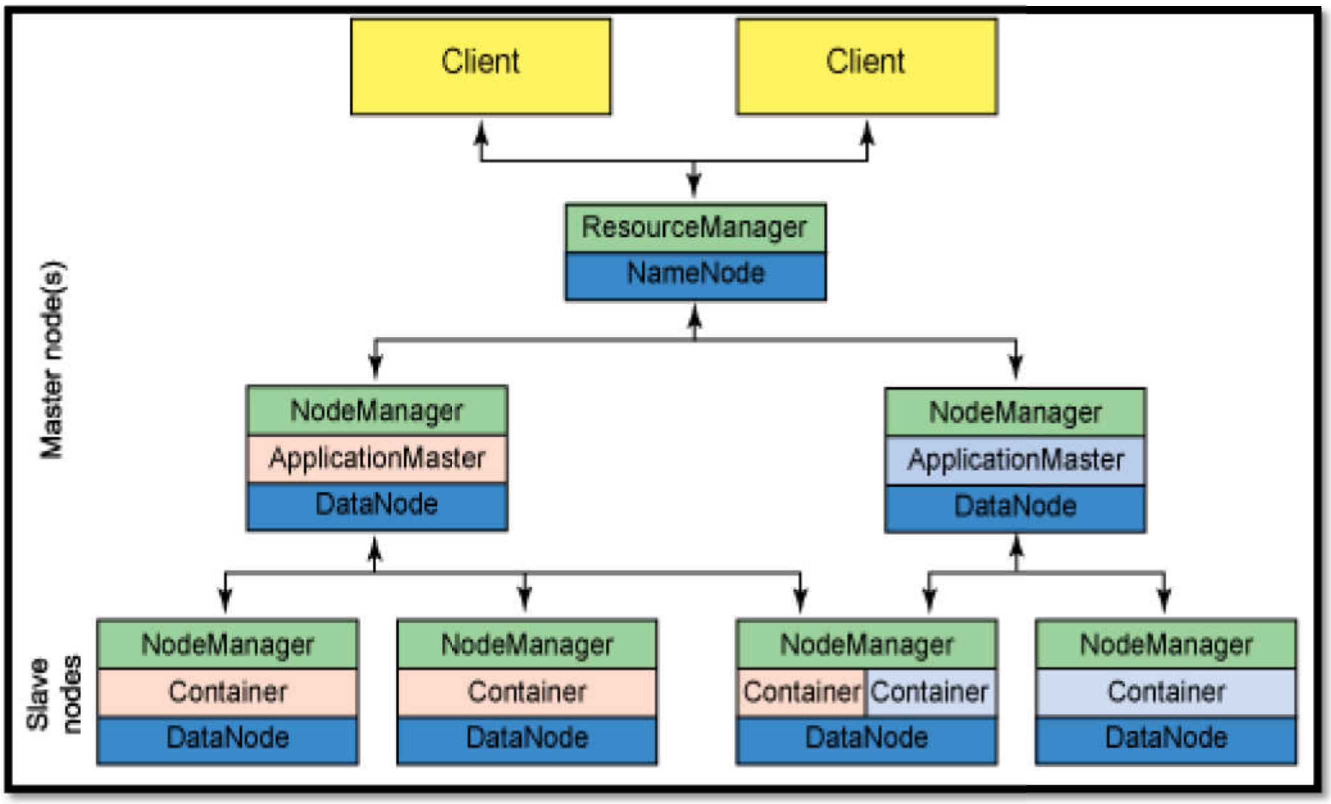
1. Hadoop Common

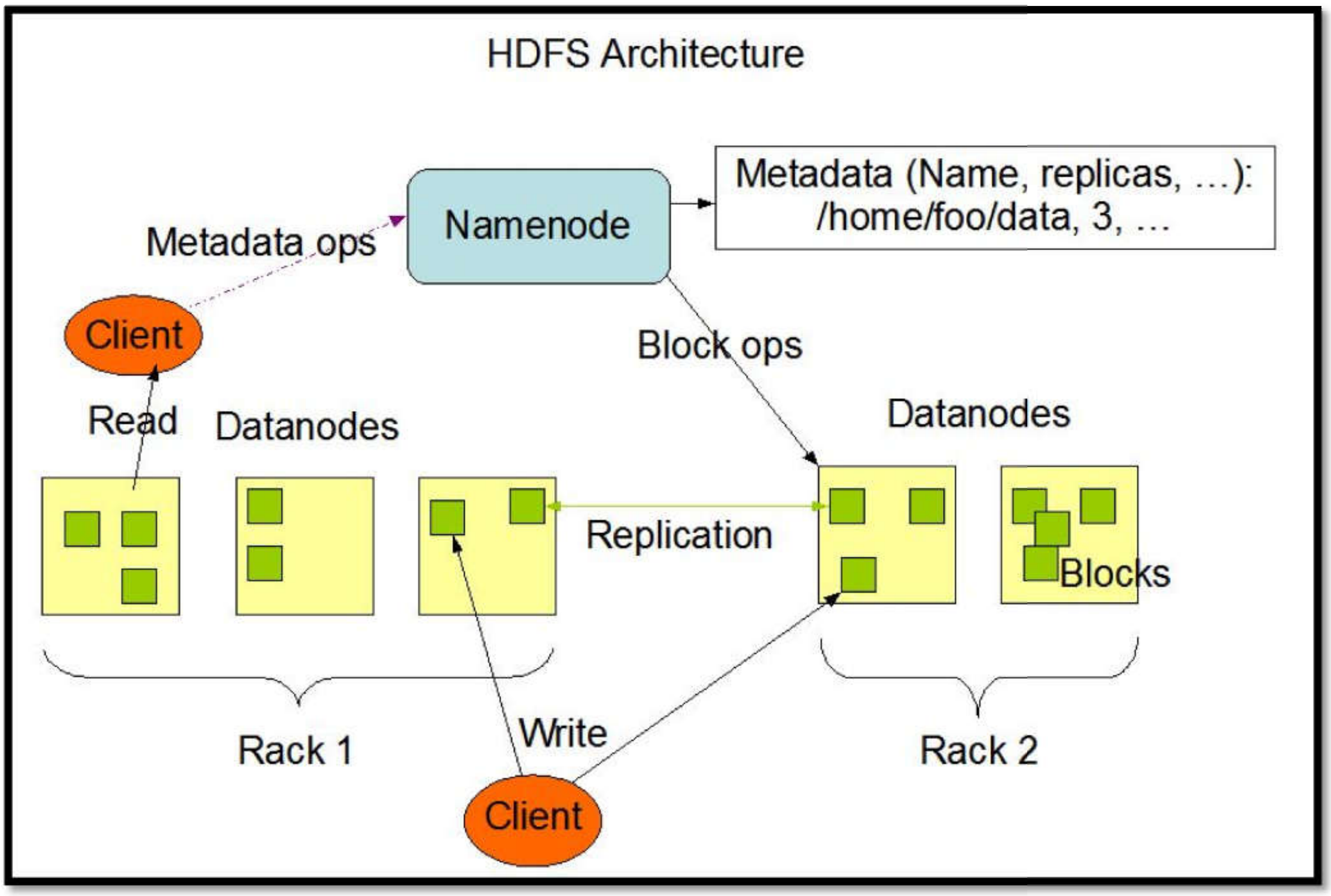
- Apache Foundation has pre-defined set of utilities and libraries that can be used by other modules within the Hadoop ecosystem.
- **Hadoop common** provides all java libraries, utilities, OS level abstraction, necessary java files and script to run Hadoop**.**
- Hadoop Common Module is a Hadoop Base API (a Jar file) for all Hadoop Components. All other components work on top of this module.
  For example, if HBase and Hive want to access HDFS they need to make of Java archives (JAR files) that are stored in Hadoop Common.

2. Hadoop Distributed File System (HDFS)

- HDFS is the **primary storage system** (default big data storage layer) of Hadoop.
- Hadoop distributed file system (HDFS) is java based file system that provides **scalable**, **fault tolerance**, **reliable** and **cost efficient data storage** for big data. HDFS is the "Secret Sauce" of Apache Hadoop components as users can dump huge datasets into HDFS and the data will sit there nicely until the user wants to leverage it for analysis.
- HDFS is a distributed filesystem that runs on **commodity hardware**.
- HDFS component creates several **replicas** of the data block to be distributed across different clusters for reliable and quick data access.
- HDFS is already configured with default configuration for many installations. Most of the time for large clusters configuration is needed.
- Hadoop **interacts** directly with HDFS by **shell-like commands**.
- HDFS comprises of **3 important components**:
  - **NameNode**
  - **DataNode**
  - **Secondary NameNode**
- HDFS operates on a **Master-Slave architecture** model where the NameNode acts as the master node for keeping a track of the storage cluster and the DataNode acts as a slave node summing up to the various systems within a Hadoop cluster.

HDFS Architecture [HDFS has a Master/Slave architecture]





Components of HDFS:

1. NameNode

- An HDFS cluster consists of a single NameNode, **a master server that**
  - Manages/maintains the file system namespace **(Primary Responsibility of Namenode)**: Any change to the file system namespace or its properties is recorded by the NameNode.
  - Regulates access to files by clients.
  - Executes file system namespace operations like opening, closing and renaming files and directories.
  - The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself.
- NameNode does not store actual data or dataset. NameNode stores **Metadata** i.e.
  - number of blocks
  - location of blocks
  - on which Rack
  - determines the mapping of blocks to DataNodes.

  This meta-data is available in memory in the master for faster retrieval of data.

- NameNode maintains and manages the slave nodes (Data Nodes), and assigns tasks to them.
- An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file and this information is stored by the NameNode. Or in other words, NameNode is responsible for taking care of the *Replication Factor* of all the blocks.

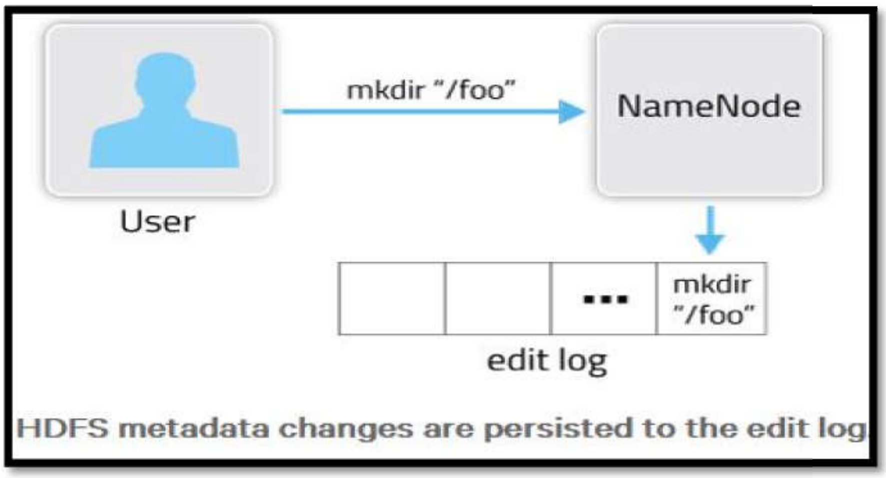
NameNode contains two important files on its hard disk (persistent data storage):

(i) **Fsimage (file system image) files**: An "Image file" that represents a point-in-time snapshot of the filesystem's metadata.
It contains:

- The entire filesystem namespace (all directory structure of HDFS) and stored as a file in the namenode's local file system.
- A serialized form of all the directories and files inodes in the filesystem. Each *inode* is an internal representation of file or directory's metadata.
- Replication level of file
- Modification and access times of files
- Access permission of files and directories
- Block size of files
- The blocks constituting a file

(ii) Edit logs files

- However, while the fsimage file format is very efficient to read, it's unsuitable for making small incremental updates like renaming a single file. Thus, rather than writing a new fsimage every time the namespace is modified, the NameNode instead records the modifying operation in the edit log for durability.



- This way, if the NameNode crashes, it can restore its state by first loading the fsimage then replaying all the operations (also called edits or transactions) in the edit log to catch up to the most recent state of the namespace.
- The edit log comprises a series of files, called edit log segments, which together represent all the namespace modifications made since the creation of the fsimage.
- When a namenode is started, latest fsimage file is loaded into **"in-memory"** and at the same time, edit log file is also loaded into memory, if fsimage file doesn't contain up-to date information, then the information which is available in edit log(s) will be replayed to update the in-memory of fsimage data.
- **Why does Namenode store metadata in "in-memory"?**
  Namenode stores metadata in "in-memory" in order to serve the multiple client request(s) as fast as possible. If it doesn't store metadata information in "in-memory", then for every operation, namenode has to load the metadata information from the disk to in-memory and start performing various checks on this metadata information. This process will consume more disk seek time for every operation (like reading from and writing to disk which is a time consuming processe). As part of in-memory, it will have both file metadata and bitmap metadata information.

2. **DataNode -** also known as Slave Node
   - DataNodes are usually one per node in the cluster.
   - Datanode is responsible for storing actual HDFS data in its local file system. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.
     The DataNode does not create all files in the same directory. Instead, it uses a heuristic to determine the optimal number of files per directory and creates subdirectories appropriately. It is not optimal to create all local files in the same directory because the local file system might not be able to efficiently support a huge number of files in a single directory.
   - Datanode performs read and write operation as per the request of the clients.
   - DataNode performs operations like block replica creation, deletion and replication according to the instruction of NameNode. Replica block of Datanode consists of 2 files on the file system.

     ✓ First file is for data
     ✓ Second file is for recording the block's metadata. HDFS Metadata includes checksums for data.

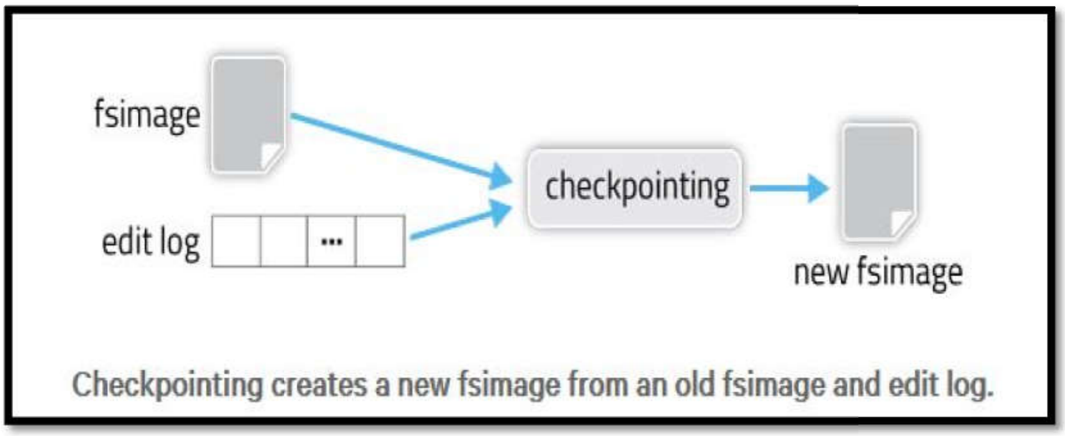
- At startup, each Datanode connects to its corresponding Namenode and does handshaking. Verification of namespace ID and software version of DataNode take place by handshaking. At the time of mismatch found, DataNode goes down automatically.
- **HeartBeat:**

  ✓ Heartbeat is the signal that is sent by the datanode to the namenode after the regular interval to time to indicate its presence, i.e. to indicate that it is alive. If after a certain time of heartbeat, Name Node does not receive any response from Data Node, then that particular Data Node used to be declared as dead.

  ✓ **The default heartbeat interval is 3 seconds**. If the DataNode in **HDFS** does not send heartbeat to NameNode in ten minutes, then NameNode considers the DataNode to be **out of service** and the Blocks replicas hosted by that DataNode to be **unavailable**. The NameNode then schedules the creation of new replicas of those blocks on other DataNodes.

3. Secondary Namenode

- It performs house-keeping activities for namenode, like periodic merging of fsimage and edits.
- **What is Checkpointing?**
  It is a process that takes an fsimage and edit log and compacts them into a new fsimage. This way, instead of replaying a potentially unbounded (larger size) edit log, the NameNode can load the final in-memory state directly from the fsimage. **This is a far more efficient operation and reduces NameNode startup time.**


fsimage
edit log
...
checkpointing
new fsimage
Checkpointing creates a new fsimage from an old fsimage and edit log.

  ✓ **Why Secondary Namenode is required?**

  Creating a new fsimage is an I/O- and CPU-intensive operation, sometimes taking minutes to perform. During a checkpoint, the namesystem also needs to restrict concurrent access from other users. So, rather than pausing the active NameNode to perform a checkpoint, HDFS defers it to either the Secondary NameNode or Standby NameNode, depending on whether NameNode high-availability is configured.

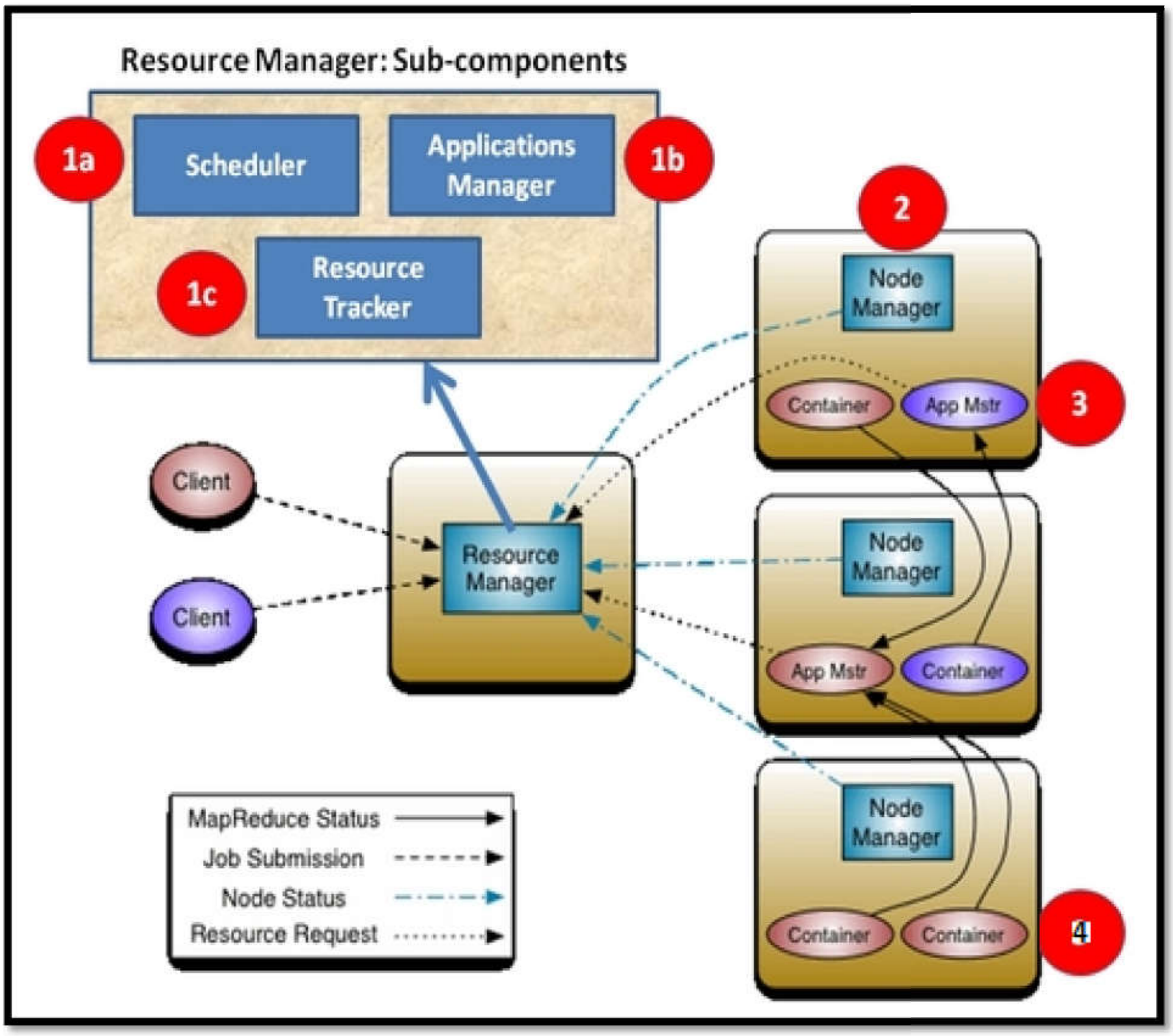
- Secondary NameNode performs a regular checkpoint in HDFS. When is Check-pointing triggerd?
  Checkpointing is triggered by one of two conditions:
  ✓ If enough time has elapsed since the last checkpoint (`dfs.namenode.checkpoint.period`),
  ✓ Or if enough new edit log transactions have accumulated (`dfs.namenode.checkpoint.txns`).

The checkpointing node (may be secondary node or standby node) periodically checks if either of these conditions are met (`dfs.namenode.checkpoint.check.period`), and if so, kicks off the checkpointing process.

- Secondary namenode, keeps edits log size within a limit. It stores the modified FsImage into persistent storage so that we can use it in the case of NameNode failure.
- It is not a backup for Namenode. **Why?**
  - ✓ Secondary Namenode is not designed to act as Namenode. It keeps the recent fsimage of Namenode. But it can work as Namenode after fsimage is merged with latest edits and blocks mapping is received after communication with data nodes. In case Namenode fails suddenly and we are unable to access its edits, then latest updates over HDFS file structure will be lost as Secondary Namenode stores only the recent fsimage but not the edits.

3. Hadoop YARN (Yet Another Resource Negotiator)

YARN forms an integral part of Hadoop 2.0 also known as **MRv2**.YARN is great enabler for dynamic resource utilization on Hadoop framework as users can run various Hadoop applications without having to bother about increasing workloads.



- **The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons.**
- Components of YARN are as follows:

(i) Global ResourceManager (*RM*)

- ✓ ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system for optimal resource utilization.
- ✓ The ResourceManager and the NodeManager form the data-computation framework.
- ✓ One cluster has one instance of Resource Manager.
- ✓ The ResourceManager has two main components: **Scheduler** and **ApplicationsManager**.

(a) Scheduler

- ▪ It is plugged with Resource Manager to help in resource allocation. The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the **CapacityScheduler** and the **FairScheduler** would be some examples of plug-ins.
- ▪ The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. Different schedulers allocate resources using different algorithms.
- ▪ The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based on the resource requirements of the applications; it does so based on the abstract notion of a resource *Container* which incorporates elements such as memory, cpu, disk, network etc.

(b) ApplicationsManager

- ▪ Is responsible for maintaining a collection of submitted applications. After application submission, it first validates the application's specifications and rejects any application that requests unsatisfiable resources for its ApplicationMaster (i.e., there is no node in the cluster that has enough resources to run the ApplicationMaster itself). It then ensures that no other application was already submitted with the same application ID—a scenario that can be caused by an erroneous or a malicious client. Finally, it forwards the admitted application to the scheduler.
- ▪ This component is also responsible for recording and managing finished applications for a while before they are completely evacuated from the ResourceManager's memory. When an application finishes, it places an ApplicationSummary in the daemon's log file. Finally, the ApplicationsManager keeps a cache of completed applications long after applications finish to support users' requests for application data (via web UI or command line).
- ▪ The configuration property "**yarn.resourcemanager.max-completed-applications**" controls the maximum number of such finished applications that the ResourceManager remembers at any point of time. The cache is a first-in, first-out list, with the oldest applications being moved out to accommodate freshly finished applications.

(ii) Node Manager

- ✓ Runs on each node (per-machine framework agent) and is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reports the same to the ResourceManager/Scheduler.

- ✓ It takes instructions from ResourceManager about resource allocation to jobs and maintains life cycle of containers and manages resources available on a single node.

(iii) Application-specific Application Master

- ✓ Per-application ApplicationMaster (*AM*). An application is either a single job or a DAG of jobs. It could be Mapreduce or any other processing framework (e.g. Spark).
- ✓ It is in effect, a framework specific library and is tasked with negotiating resources with the ResourceManager and working with the NodeManager(s) to start the containers so as to get those resources for executing and monitoring the tasks.
- ✓ It is the actual instance which does processing.

(iv) Container

- ✓ It is a set of allocated system resources (CPU Core and Memory).
- ✓ Containers are allocated and managed by NodeManager and are used by tasks.

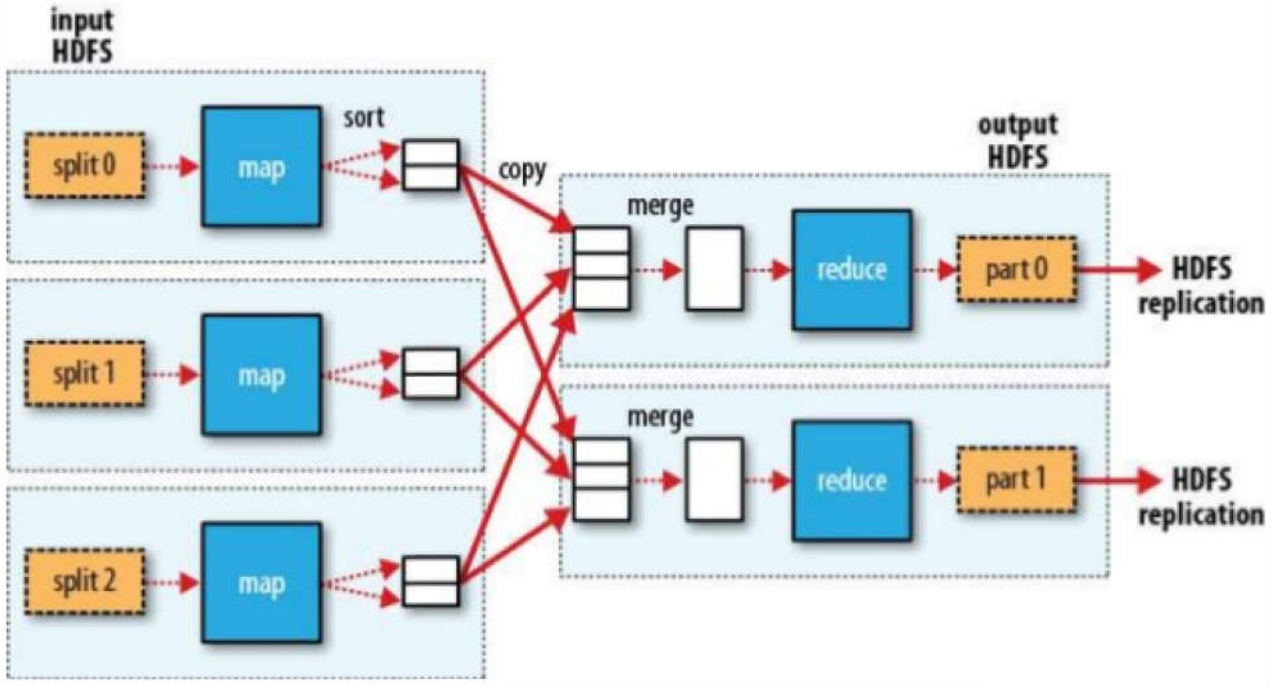
Note:-

- ➢ **Resource is handled by Resource Manager and Node Manager.**
- ➢ **Processing is handled by Application Master (MapReduce is one of the many possible types of Application Master). So, processing other than MapReduce is also possible.**

4. Hadoop MapReduce (MRv1)

- MapReduce is a **Java-based system created by Google** where the actual data from the HDFS store gets processed efficiently.
- MapReduce **breaks down a big data processing job into smaller tasks.**
- MapReduce is responsible for the analysing large datasets in parallel before reducing it to find the results.
- In the Hadoop ecosystem, Hadoop MapReduce is a framework based on YARN architecture. YARN based Hadoop architecture, supports parallel processing of huge data sets and MapReduce provides the framework for easily writing applications on thousands of nodes, considering fault and failure management.
- **The basic principle of operation behind MapReduce** is that the “Map” job sends a query for processing to various nodes in a Hadoop cluster and the “Reduce” job collects all the results to output into a single value. Map Task in the Hadoop ecosystem takes input data and splits into independent chunks and output of this task will be the input for Reduce Task. In the same Hadoop ecosystem Reduce task combines Mapped data tuples into smaller set of tuples. Meanwhile, both input and output of tasks are stored in a file system.
- MapReduce takes care of scheduling jobs, monitoring jobs and re-executes the failed task.
- MapReduce framework forms the compute node while the HDFS file system forms the data node. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

MapReduce data flow with multiple reduce tasks



Data Access Components of Hadoop Ecosystem - Pig and Hive

Pig

Apache Pig is a convenient tool **developed by Yahoo** for analysing huge data sets efficiently and easily. It provides a high level data **flow language Pig Latin** that is optimized, extensible and easy to use. The most outstanding feature of Pig programs is that their structure is open to considerable parallelization making it easy for handling large data sets.

Hive

Hive **developed by Facebook** is a data warehouse built on top of Hadoop and provides a simple language known as **HiveQL** similar to SQL for querying, data summarization and analysis. Hive makes querying faster through indexing.

Data Integration Components of Hadoop Ecosystem - Sqoop and Flume

Sqoop

Sqoop component is **used for importing data from external sources into related Hadoop components** like HDFS, HBase or Hive. It can also be used for exporting data from Hadoop or other external structured data stores. Sqoop parallelizes data transfer, mitigates excessive loads, allows data imports, efficient data analysis and copies data quickly.

Flume

Flume component is used to gather and aggregate large amounts of data. Apache Flume is **used for collecting data from its origin and sending it back to the resting location (HDFS)**. Flume accomplishes this by outlining data flows that consist of 3 primary structures channels, sources and sinks. **The processes that run the dataflow with flume are known as agents and the bits of data that flow via flume are known as events.**

Data Storage Component of Hadoop Ecosystem – HBase

HBase

HBase is a **column-oriented database** that uses HDFS for underlying storage of data. HBase **supports random reads and also batch computations** using MapReduce. With HBase NoSQL database enterprise can create large tables with millions of rows and columns on hardware machine. **The best practice to use HBase is when there is a requirement for random 'read or write' access to big datasets.**

Facebook is one the largest users of HBase with its messaging platform built on top of HBase in 2010. HBase is also used by Facebook for streaming data analysis, internal monitoring system, Nearby Friends Feature, Search Indexing and scraping data for their internal data warehouses.

Monitoring, Management and Orchestration Components of Hadoop Ecosystem - Oozie and Zookeeper

Oozie

Oozie is a **workflow scheduler** where the workflows are expressed as Directed Acyclic Graphs. Oozie runs in a Java servlet container Tomcat and makes use of a database to store all the running workflow instances, their states ad variables along with the workflow definitions to manage Hadoop jobs (MapReduce, Sqoop, Pig and Hive).The workflows in Oozie are executed based on data and time dependencies.

Zookeeper

Zookeeper is the king of **coordination** and provides simple, fast, reliable and ordered operational services for a Hadoop cluster. **Zookeeper is responsible for synchronization service, distributed configuration service and for providing a naming registry for distributed systems.**

Other Components

AMBARI

A Hadoop component, Ambari is a RESTful API which provides easy to use web user interface for Hadoop management. Ambari provides step-by-step wizard for installing Hadoop ecosystem services. It is equipped with central management to start, stop and re-configure Hadoop services and it facilitates the metrics collection, alert framework, which can monitor the health status of the Hadoop cluster. Recent release of Ambari has added the service check for Apache spark Services and supports Spark 1.6.

MAHOUT

Mahout is an important Hadoop component for machine learning, this provides implementation of various machine learning algorithms. This Hadoop component helps with considering user behavior in providing suggestions, categorizing the items to its respective group, classifying items based on the categorization and supporting in implementation group mining or itemset mining, to determine items which appear in group.

Apache Kafka

Kafka is used for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant, wicked fast, and runs in production in thousands of companies.