Doug Cutting – Hadoop creator – is reported to have explained how the name for his Big Data technology came about: "The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria."

Introduction to Apache Hadoop

- With the continuous business growth and start-ups flourishing up, the need to store a large amount of data has also increased rapidly.
- To meet the growth of business and gain profits the companies would fetch tools to analyse the Big Data.
- To meet the growing demand, Apache Software Foundation launched Hadoop, a tool to store, analyse, and process Big Data.

What is Hadoop?

- Apache Hadoop runs on Java and is an open source framework used to process and store a large amount of data on a huge cluster.

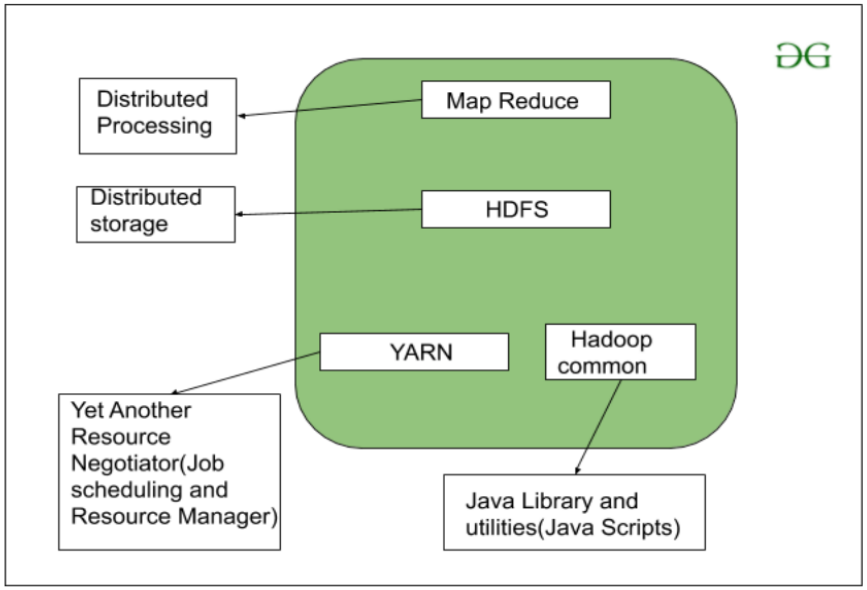
- The Hadoop is an open source project of Apache Software Foundation and was originally created by Yahoo in 2006.

- Since then, this open source project has brought revolution in Big Data analytics and taken over the Big Data market.

- Apache Hadoop is easily scalable and you can scale a number of machines through a single server.

Hadoop framework is composed of four main components

- Hadoop Common
- Hadoop Distributed File System
- YARN
- MapReduce



Hadoop Common:

- It refers to the common Java utilities and libraries that support Hadoop modules.

Hadoop Distributed File Systems:

- It is the primary storage system that Hadoop applications use.
- It is a distributed file system that enables you to have an access to applications data.

Hadoop MapReduce:

- Hadoop MapReduce is a software framework used for the parallel processing of big data.

Hadoop YARN:

- YARN is the resource management technology used by Hadoop.
- It is responsible for resource management and job scheduling

Features of Apache Hadoop

Scalability: Apache Hadoop uses distributed processing of local data, this allows the data to be stored, processed, and analyzed at a large scale.

Reliability: In Apache Hadoop, the data is auto-replicated and hence can generate a redundant copy of data when it comes to system failures. Thus, Apache Hadoop has fault tolerance feature.

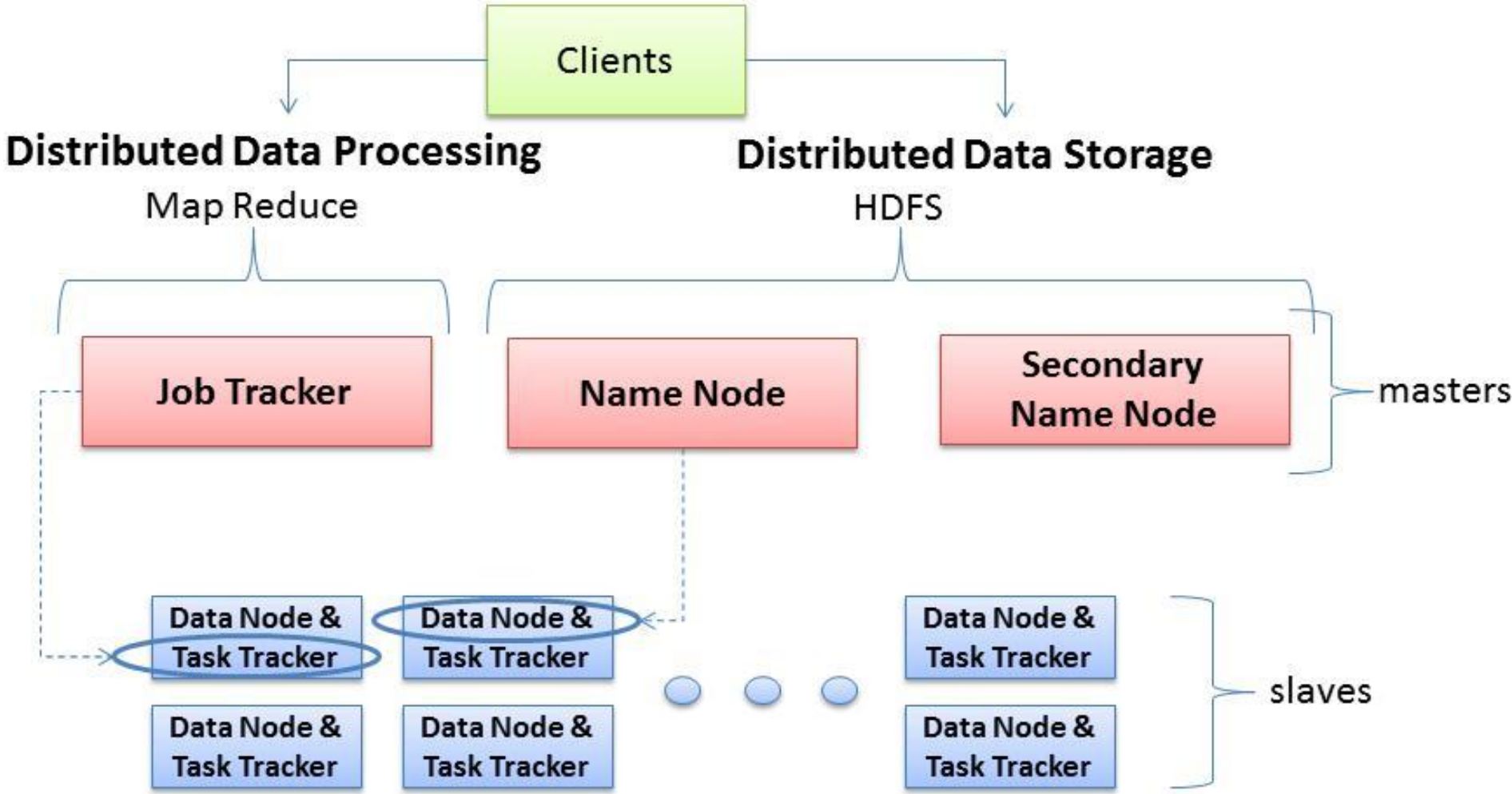
Flexibility: Apache Hadoop does not follow the traditional relational database rules. It can store information and data in any format such as structured, unstructured, and semi-structured.

Cost-effectiveness: Apache Hadoop is open source and is free of cost. This makes it cost effective and available for all.

Compatibility: Apache Hadoop, being a Java-based framework, is compatible with all the platforms.

Hadoop Distributed File System (HDFS)

HDFS Architecture



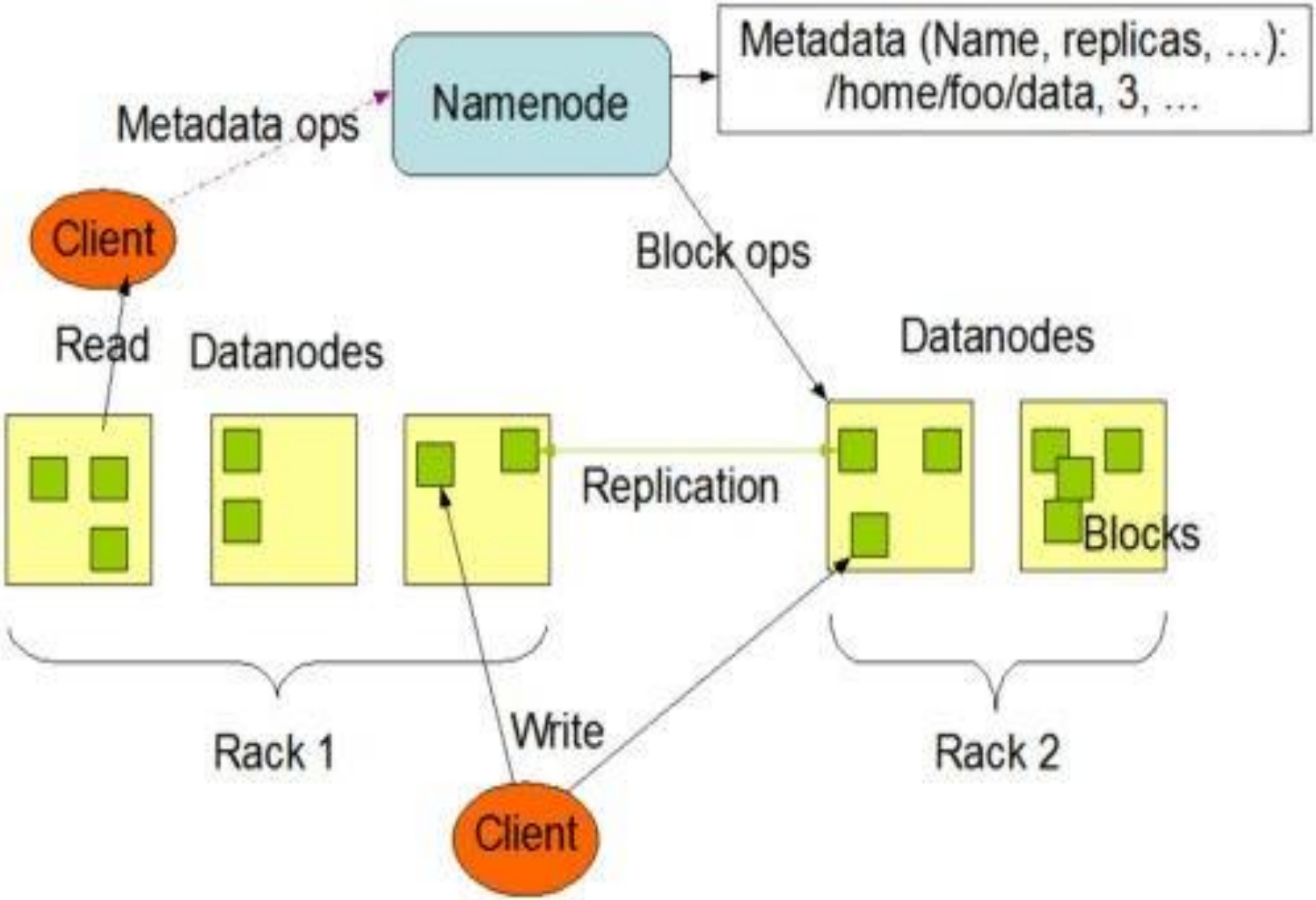
What is HDFS?

- The Hadoop Distributed File System is a java-based file, developed by Apache Software Foundation with the purpose of providing versatile, resilient, and clustered approach to manage files in a Big Data environment using commodity servers.

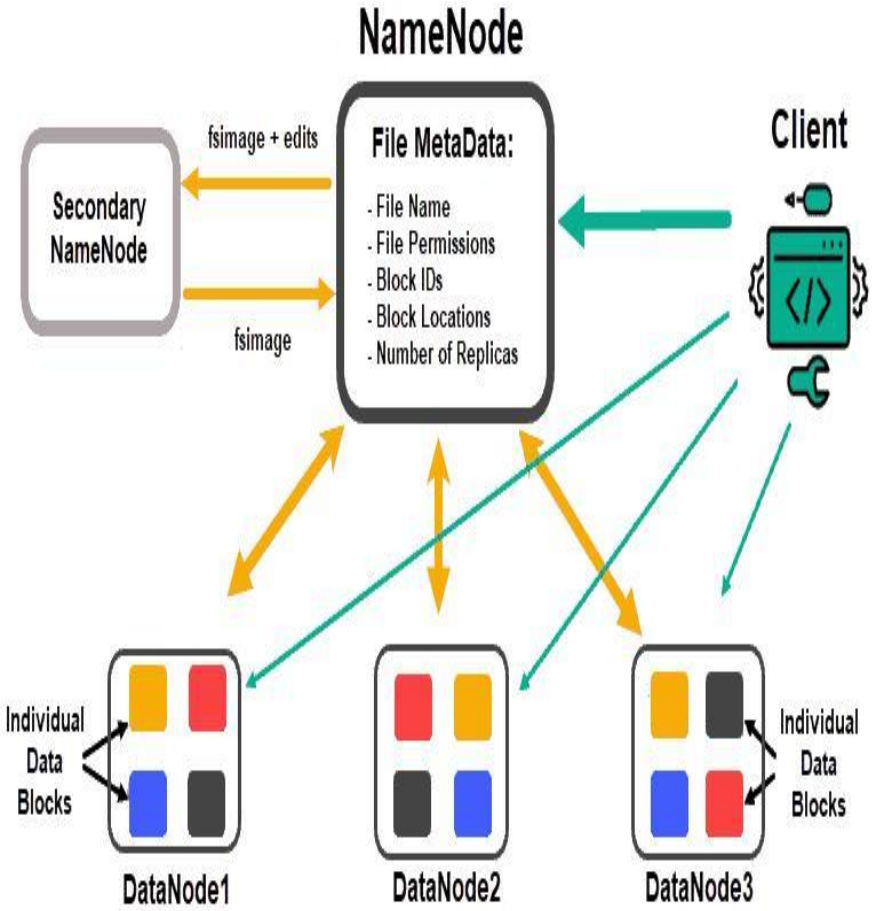
- HDFS used to store a large amount of data by placing them on multiple machines as there are hundreds and thousands of machines connected together.

- The goal is to store a smaller number of larger files rather than the greater number of small files.
- HDFS provides high reliability of data because it used to replicate data into three different copies- two are saved in one group and third one in another.
- HDFS is scalable in nature as it can be extended to 200 PB of storage where a single cluster contains 4500 to 10000 servers, supporting billions of blocks and files.

- HDFS is used to split files into multiple blocks. Each file is replicated when it is stored in Hadoop cluster.
- The default size of that block of data is 64 MB but it can be extended up to 256 MB as per the requirement.
- HDFS stores the application data and the file system metadata on two different servers.
- NameNode is used to store the file system metadata while and application data is stored by the DataNode.
- To ensure the data reliability and availability to the highest point, HDFS replicates the file content many times.
- NameNode and DataNode communicate with each other by using TCP protocols.
- Hadoop architecture performance depends upon Hard-drives throughput and the network speed for the data transfer.


Metadata (Name, replicas, …):
/home/foo/data, 3, …
Namenode
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

- Hadoop works on a master node which is NameNode and multiple slave nodes which are DataNodes on a commodity cluster.
- As all the nodes are present in the same rack in the data center, data is broken into different blocks that are distributed among different nodes for the storage.
- These blocks are replicated across nodes to make data available in case of a failure.



Features of HDFS

- **Fault-Tolerant**
  - HDFS is highly fault-tolerant.
  - HDFS replicates and stores data in three different locations.
  - So, in the case of corruption or unavailability, data can be accessed from the previous location.
- **Scalability**
  - Scalability means adding or subtracting the cluster from HDFS environment.
  - Scaling is done by the two ways – vertical or horizontal.
  - In vertical scaling, you can add up any number of nodes to the cluster but there is some downtime.
  - In horizontal scaling, there is no downtime; you can add any number of nodes in the cluster in real time.
- **Data Availability**
  - Data is replicated and stored on different nodes due to which data is available all the time.
  - In case of network, node or some hardware failure, data is accessible without any trouble from a different source or from a different node.

- **Data Reliability**
  - HDFS provides highly reliable data storage, as data are divided into blocks and each block is replicated in the cluster which made data reliable.
  - If one node contains the data is down, it can be accessed from others because HDFS creates three replicas of each block.
  - When there is no loss of data in case of failure then it is highly reliable.

- **Replication**

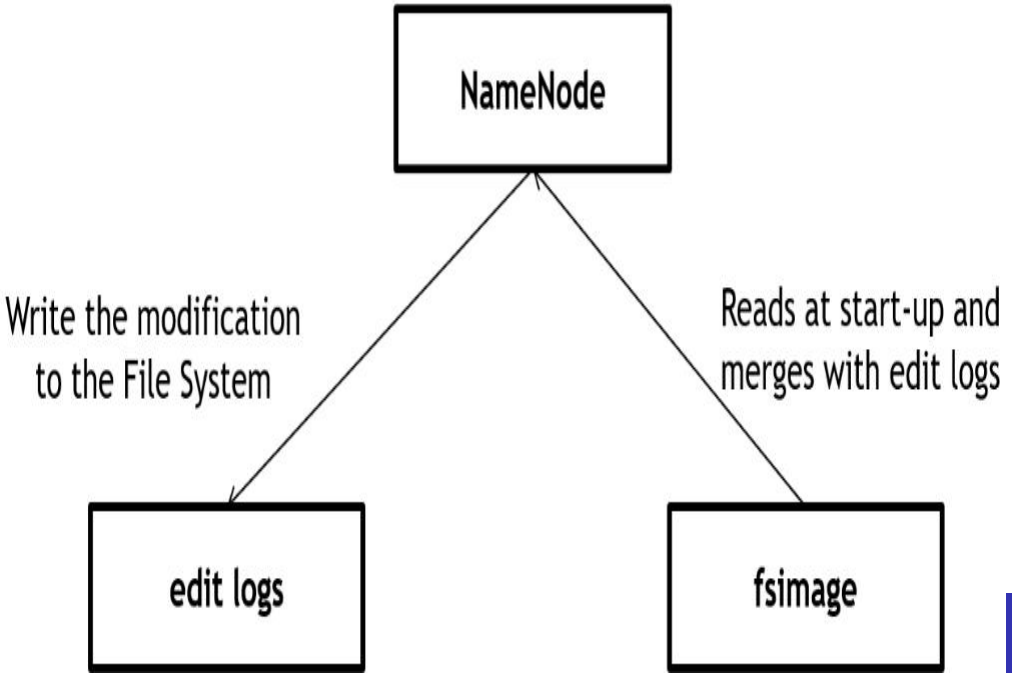
- Data replication is one of the unique and important features of HDFS.
- HDFS used to create replicas of data in the different cluster.
- As if one node goes down it can be accessed from other because every data block has three replicas created.

This is why, there is no chance of data loss.

NameNode

- Single node in cluster
- Brain of HDFS
- Hadoop employs master/slave architecture for both distributed storage and distributed processing.

- NameNode is the master of HDFS that directs slave DataNode to perform low level tasks.



NameNode as book Keeper of HDFS

- NameNode is maintains list of all blocks of file and list of all data nodes that contains the block.
- It keeps track of how files are stored into file blocks, which nodes store these blocks and overall health of HDFS
- The function of NameNode is memory and IO intensive
- The server hosting NameNode doesn’t store any user data or perform any processing.

Name node – Drawback

- Name node is the Single point of failure in Hadoop cluster
- For other data nodes, if their host node fails due to h/w or s/w reasons, the Hadoop cluster will continue smoothly.

Data Nodes

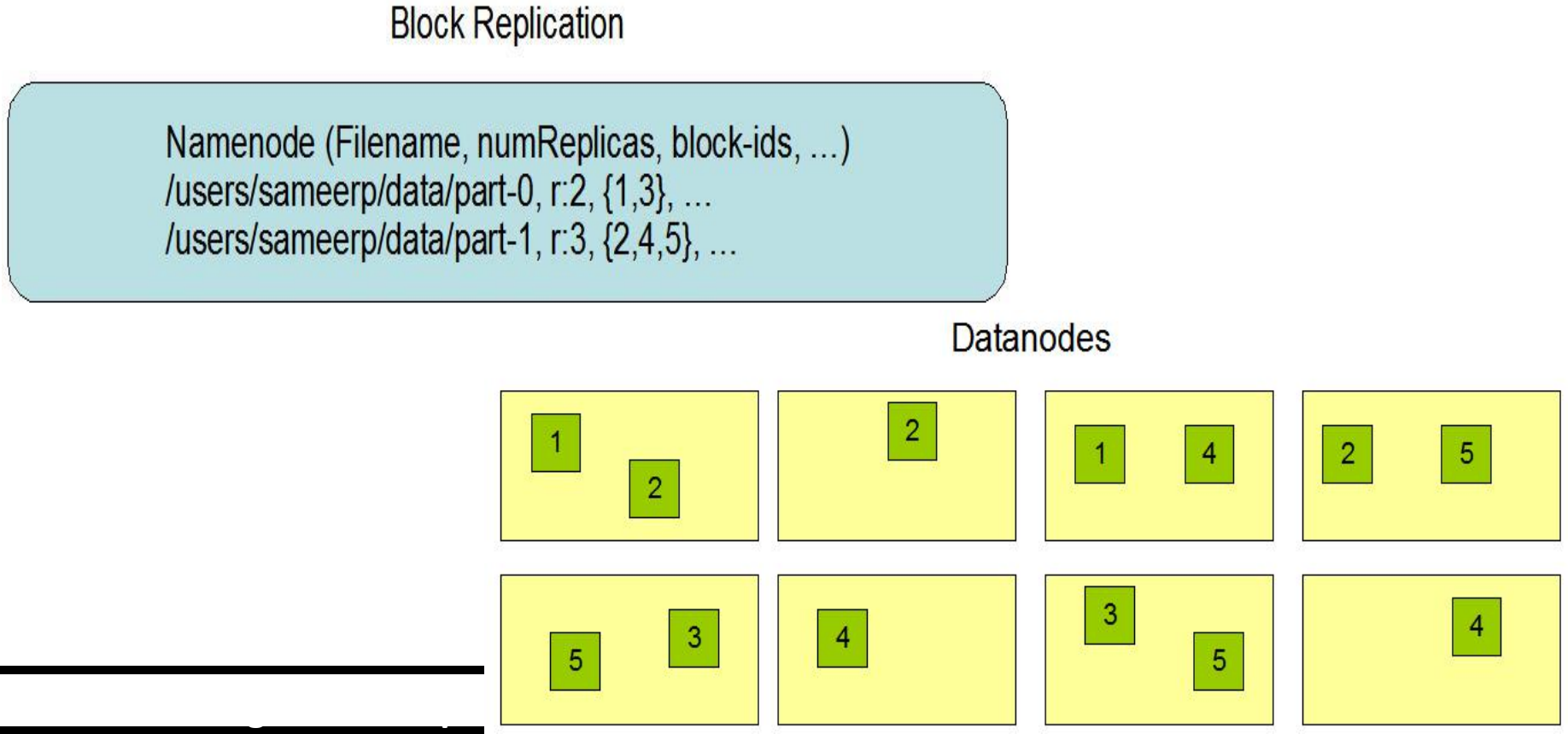
- Each slave machine in the cluster will host a data node to perform grunt work of Distributed File System
- Reading and writing HDFS blocks to actual files on local file system
- During r/w a HDFS file, the file is broken into blocks and NameNode will tell the client which data node each block resides in.
- Client communicates directly with the DataNodes to process local files corresponding to the blocks
- Data node may communicate with other DataNodes to replicate data blocks for redundancy.

Data Replication

- HDFS is designed to reliably store very large files across machines in a large cluster.
- It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.
- The blocks of a file are replicated for fault tolerance.
- The block size and replication factor are configurable per file.
- An application can specify the number of replicas of a file.
- The replication factor can be specified at file creation time and can be changed later.
- Files in HDFS are write-once and have strictly one writer at any time.
- The NameNode makes all decisions regarding replication of blocks.

Heartbeat and Block report of data nodes

- NameNode periodically receives a Heartbeat and a Block report from each of the DataNodes in the cluster
- Receipt of a Heartbeat implies that the DataNode is functioning properly
- A Block report contains a list of all blocks on a DataNode.



Replication factor

When the replication factor is three, HDFS's placement policy is to put:

1.one replica:

- on the local machine if the writer is on a DataNode,
- otherwise on a random DataNode,

1.another replica on a node in a different (remote) rack,

1.last on a different node in the same remote rack.

This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure.

Replication factor >3

•If the replication factor is greater than 3, the placement of the 4th and following replicas are determined randomly while keeping the number of replicas per rack below the upper limit (which is basically (replicas - 1) / racks + 2)

DataNodes and NameNodes

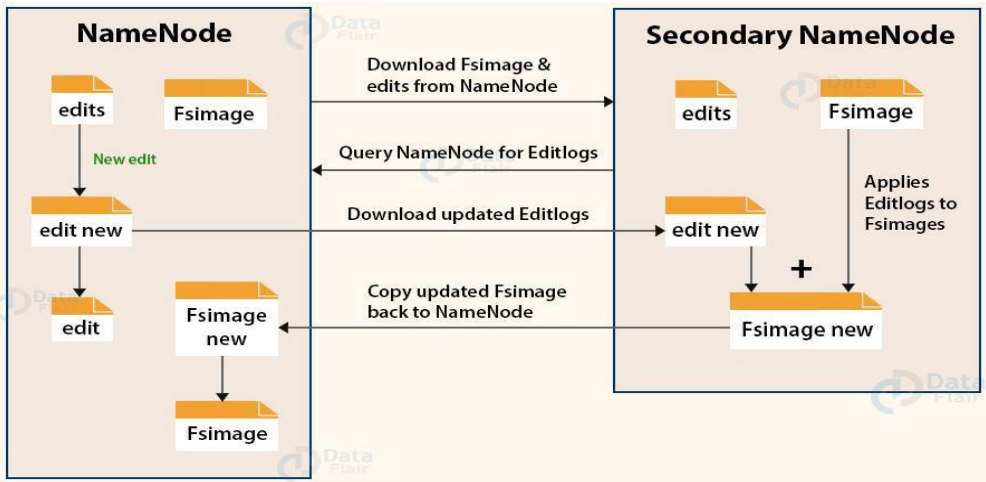
•DataNode constantly report to NameNode

•Upon initialization, each DataNode informs NameNode of the blocks it is currently storing

•After mapping, the DataNode continually poll the NameNode to provide information regarding local changes and receive instructions to create, move or delete blocks from local disk.

Secondary Name Node (SNN)

- SNN is an assistant daemon for maintaining the state of clusters HDFS.
- Like NameNode, each cluster has one SNN.
- SNN does not receive or record any real-time changes to HDFS.
- SNN communicates with NameNode to take snapshots of HDFS meta data at intervals defined by cluster configuration.
- NameNode is a single point of failure for Hadoop clusters and SNN snapshots help minimize the downtime and loss of data.
- NameNode failure requires human intervention to reconfigure the cluster to use SNN as primary NameNode

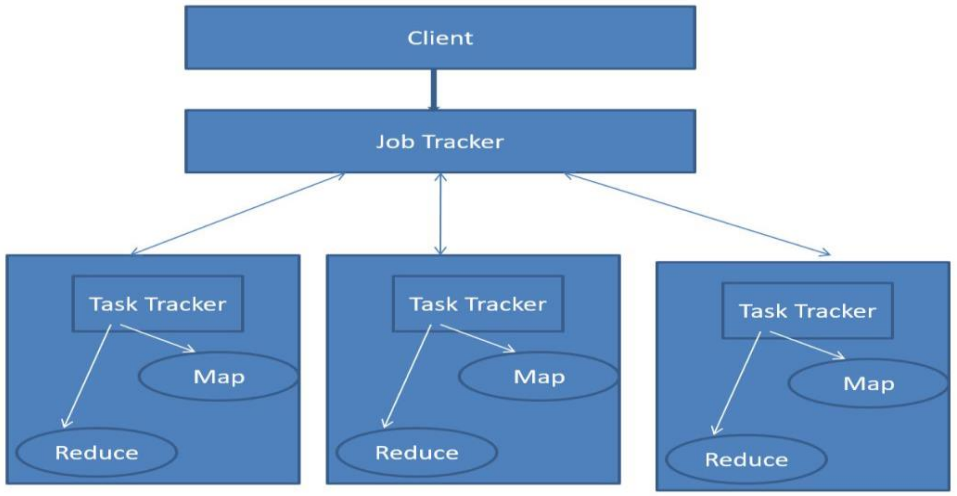


Job Tracker

- Job tracker is a liaison between the client application and Hadoop
- Once the client submits the code to the cluster the JobTracker determines the execution plan by determining which files to process
- Assigns nodes to different tasks and monitors all tasks as they are running
- If a task fails, the job tracker will automatically relaunch the task on a different node

Task Tracker

- Job tracker is the master overseeing the overall execution of MapReduce job.
- Task trackers manages the execution of individual tasks.
- Task tracker can spawn multiple JVMs to handle many map/reduce tasks in parallel.
- Main responsibility is to constantly communicate with Job tracker.
- If Job tracker fails to receive a heartbeat from TaskTracker within the specified amount of time, it will assume task tracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster.



- **NameNode**
- The NameNode is known as a smart node in the cluster.
- NameNode knows which data node contains which blocks and where data node has been placed in the clusters.
- The NameNode also contains the access authority which is given to files to perform read, write, create, delete and replication of various blocks between the data nodes.
- The NameNode is connected with the data nodes in the loosely coupled fashion.
- This provides the feature of scalability in real time means it can add or delete nodes as per the requirement.
- Data nodes used to communicate every time with the NameNode to check whether the certain task is completed or not.
- Communication between these two ensures that NameNode knows the status of each data node all the time.
- If one of the data nodes is not functioning properly then NameNode can assign that task to another node.

- To operate normally, data nodes in the same block communicate with each other.
- The NameNode is replicated to overcome system failure and is the most critical part of the whole system.
- Data nodes are not considered to be smart, but flexible in nature and data blocks are replicated across various nodes and the NameNode is used to manage the access.
- To gain the maximum efficiency, replication mechanism is used and all the nodes of the cluster are placed into a rack. Rack Id is used by the NameNode to track data nodes in the cluster.
- A heartbeat message is put through to ensure that the data nodes and the NameNode are still connected in every 8 seconds.
- When the heartbeat is no longer available, then the NameNode detach that data node from the cluster and works in the normal manner.
- When the heartbeat comes, data node is added back to the cluster.
- Transaction log and the checksum are used to maintain the data integrity.

- Transaction log used to keep track of every operation and help them in auditing and rebuilding the file system, in case of an exception.
- Checksum validates the content in HDFS.
- When a user requests a file, it verifies the checksum of that content.
- If checksum validation matches then they can access it.
- If the checksum reports an error, then the file is hidden to avoid tampering.
- The performance also depends on where the data is stored, so it is stored on local disk in the commodity servers.
- To ensure that one server failure doesn’t corrupt the whole file, data blocks are replicated in various data nodes.
- The degree of replication and the number of data nodes are managed at a time when the cluster is implemented.

A basic data flow of the Hadoop system can be divided into four phases:

1.Capture Big Data : The sources can be extensive lists that are structured, semi-structured, and unstructured, some streaming, real-time data sources, sensors, devices, machine-captured data, and many other sources. For data capturing and storage, we have different data integrators such as, Flume, Sqoop, Storm, and so on in the Hadoop ecosystem, depending on the type of data.

2.Process and Structure: We will be cleansing, filtering, and transforming the data by using a MapReduce-based framework or some other frameworks which can perform distributed programming in the Hadoop ecosystem. The frameworks available currently are MapReduce, Hive, Pig, Spark and so on.

3.Distribute Results: The processed data can be used by the BI and analytics system or the big data analytics system for performing analysis or visualization.

4.Feedback and Retain: The data analyzed can be fed back to Hadoop and used for improvements.

Dataflow in MapReduce

- Mapper is the first phase where datasets are splits into chunks.
- Mapper works on a one chunk at a time.
- The output from the mapper phase is written to the local disk machine where it is running.
- An output created by the mapper phase is known as Intermediate output.
- Intermediate output by mapper phase is written to the local disk always.
- When the mapper phase is done, the intermediate output is transferred to reducer node.
- Hence, the transferring of intermediate data from the mapper to reducer is known as Shuffle.
- If the shuffle phase will not happen then the reducer will not have any input to work on.
- The entire keys which are generated during mapper are sorted by MapReduce.
- It starts even before reducers and all the intermediate key-value pairs are generated by the mapper, are sorted by key and not by the value.
- Sorting helps reducers to start a new reduce task at the right time.
- Reduce task is started by the reducers when the key is sorted and input data is different from the previous.
- Key-value pairs are taken as input in very reduce task and key-value pairs as output are generated.
- If reducers are specified as zero then no shuffling and sorting are performed and which makes the mapper phase faster.

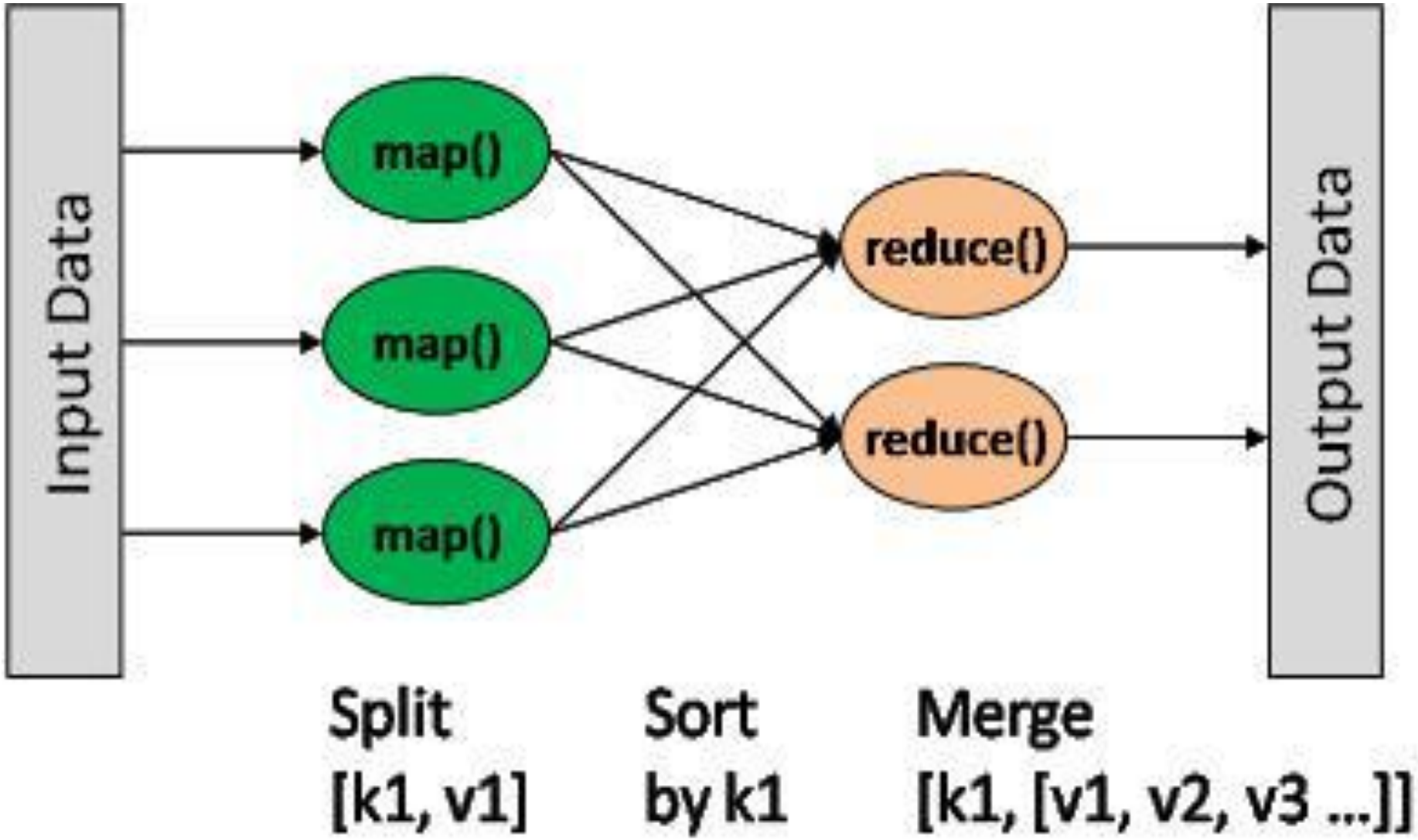
MapReduce

- **The heart of Apache Hadoop is Hadoop MapReduce.**

- It’s a programming model used for processing large datasets in parallel across hundreds or thousands of Hadoop clusters on commodity hardware.

- The framework does all the work, you just need to put the business logic into the MapReduce.

- All the work is divided into the smaller works and assigned to the slave by a master who is submitted by the user.
- Hadoop MapReduce is designed by using a different approach.
- Different kinds of lists as input is presented to MapReduce for processing and it converts those lists into output which
- is also in the form lists.

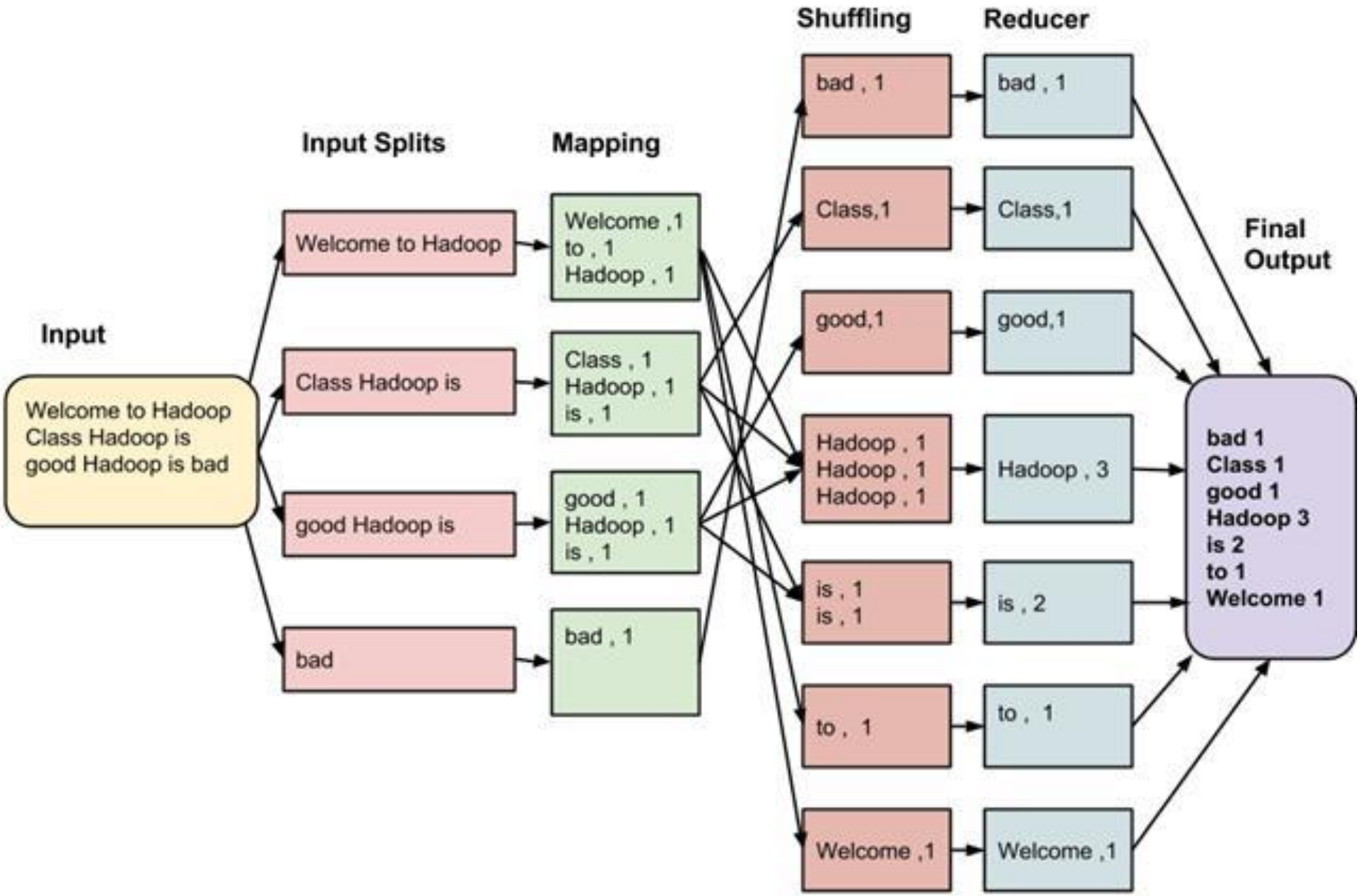

Input Data
map()
map()
map()
reduce()
reduce()
Output Data
Split
[k1, v1]
Sort
by k1
Merge
[k1, [v1, v2, v3 ...]]

- Hadoop MapReduce parallel processing has made Hadoop more efficient and powerful.
- Jobs are divided into small parts by MapReduce and each of them resides in parallel on the cluster.
- All the problems are divided into a larger number of small chunks and each of the chunks are processed individually for the output.For the final output, these are further processed.
- Hadoop MapReduce can be scaled hundreds to thousands of computers across the clusters.
- Different computers in the clusters used to process the jobs which could not be processed by a large one.
- These two transform the lists of input data elements by providing those key-pair values and then back into the lists of output data elements by combining the key pair values.

- A small phase of Shuffle and Sort also come during the Map and Reduce phase in MapReduce.
- Mapper and Reducer is an execution of two processing layer.
- Input data, the MapReduce program, and configuration information are what a MapReduce Job contains.
- So, if the client wants a MapReduce Job to execute, he needs to provide input data, write a MapReduce program, and provide some configuration information.
- Execution of a chunk of data in Mapper or Reducer phase is known as the Task or Task-In-Progress.
- The attempt of any instance to execute a task on a node is known as Task Attempt.
- There's possibility task cannot be executed due to machine failure. Then the task is rescheduled another node. Rescheduling of the task can only be done for 4 times.
- If any job fails more than the 4 times then it is considered to be a failed job.

Working Process of Map and Reduce

- User-written function at mapper is used for processing of input data in mapper.
- All the business logic is put into the mapper level because it is a place where all the heavy processing is done.
- A number of mappers are always greater than the reducers.
- The output which is produced by the mapper is intermediate data and it is input for the reducer.
- User-written function at reducer is used to process the intermediate data and after this final output is generated.
- The output generated in reducer phase is stored in HDFS and replication is done as per usual. (Figure)



Working of MapReduce

Map Input

- Map(inkey, invalue)-> list(intermediate key, intermediate value)
- Purpose of map phase is to organize data in preparation for processing done in reduce phase
- Input to map function is in the form of (key, value) pairs, even though the input to MapReduce program is a file or files.

Map

•Value: data record
•Key: Offset of data record from the beginning of the file
•Output: Collection of Key value pairs which are inputs for reduce function
•E.g. Wordcount Problem
•Mapper can run several identical mappers in parallel

Reduce

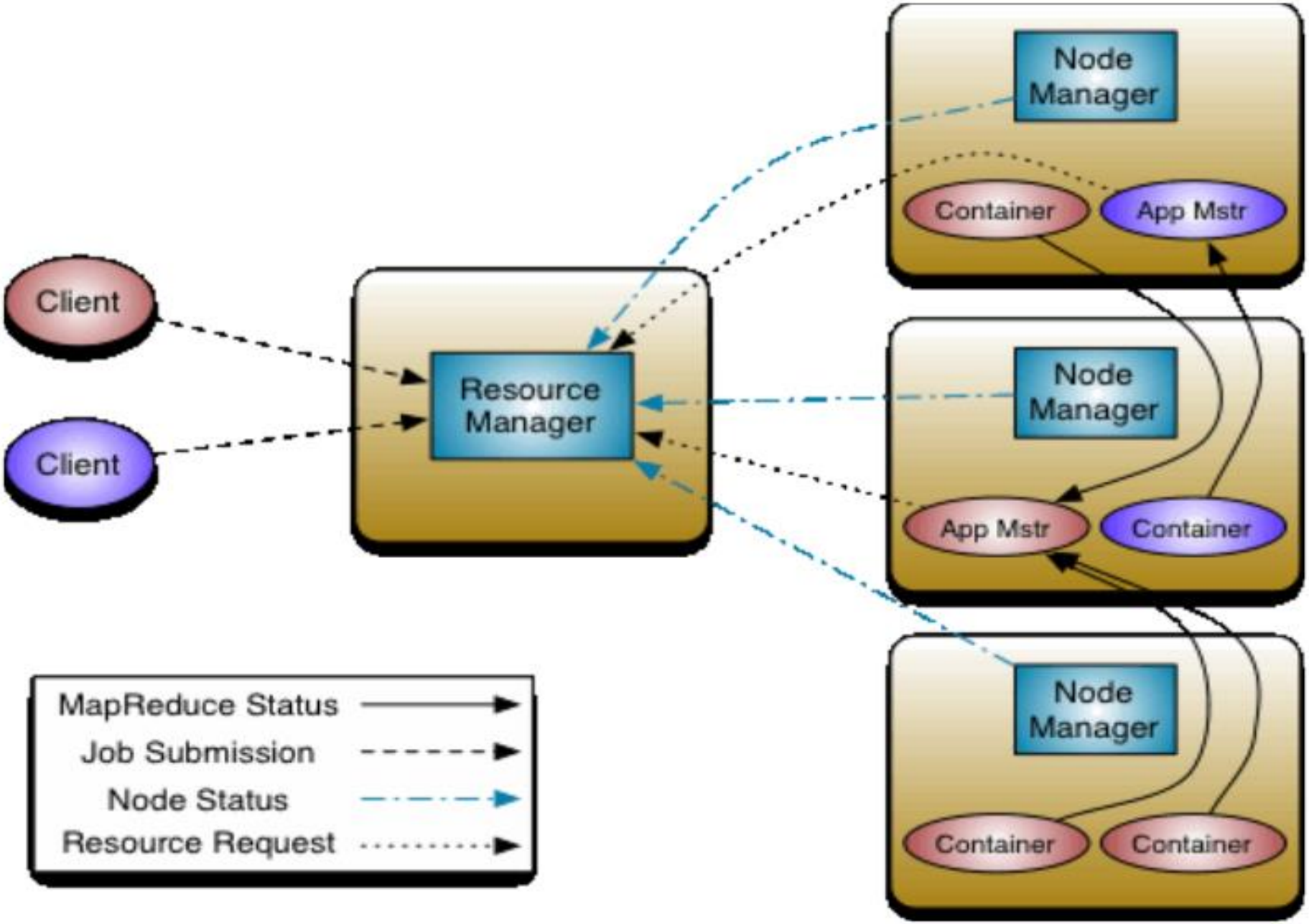
•Reduce function processes intermediate values for particular key generated by map function and generates the output
•One to one mapping between keys and reducers
•Several reducers can run in parallel. Independent of each other
•No. of reducers decided by the user.
•Default no. of reducers is 1

YARN

The actual data processing occurs within the Containers executed by the Application Master. A Container grants rights to an application to use a specific amount of resources (memory, cpu etc.) on a specific host.

YARN is not the only new major feature of Hadoop 2.0. HDFS has undergone a major transformation with a collection of new features that include:

- **NameNode HA:** automated failover with a hot standby and resiliency for the Name Node master service.
- **Snapshots:** point-in-time recovery for backup, disaster recovery and protection against use errors.
- **Federation:** a clear separation of namespace and storage by enabling generic block storage layer

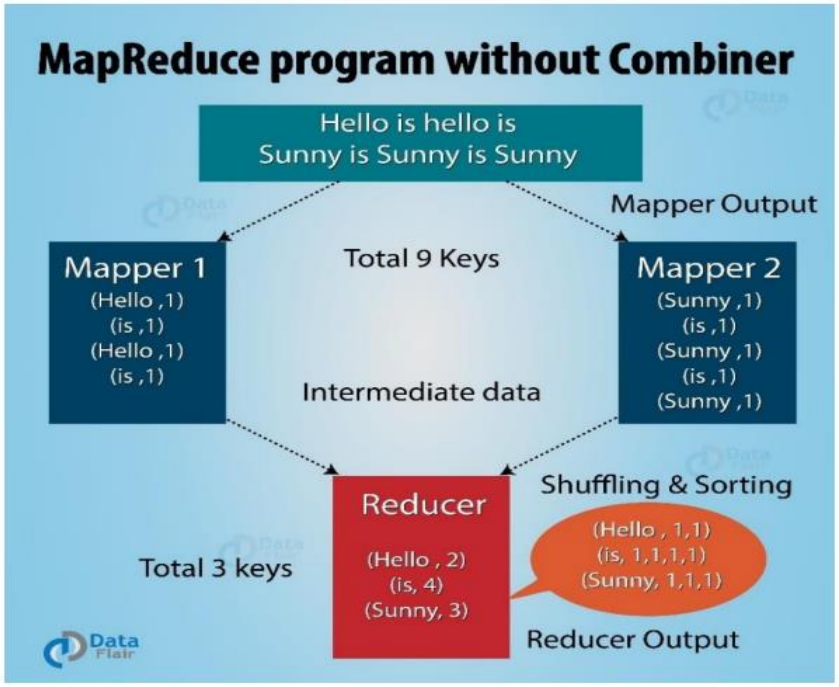


YARN workflow

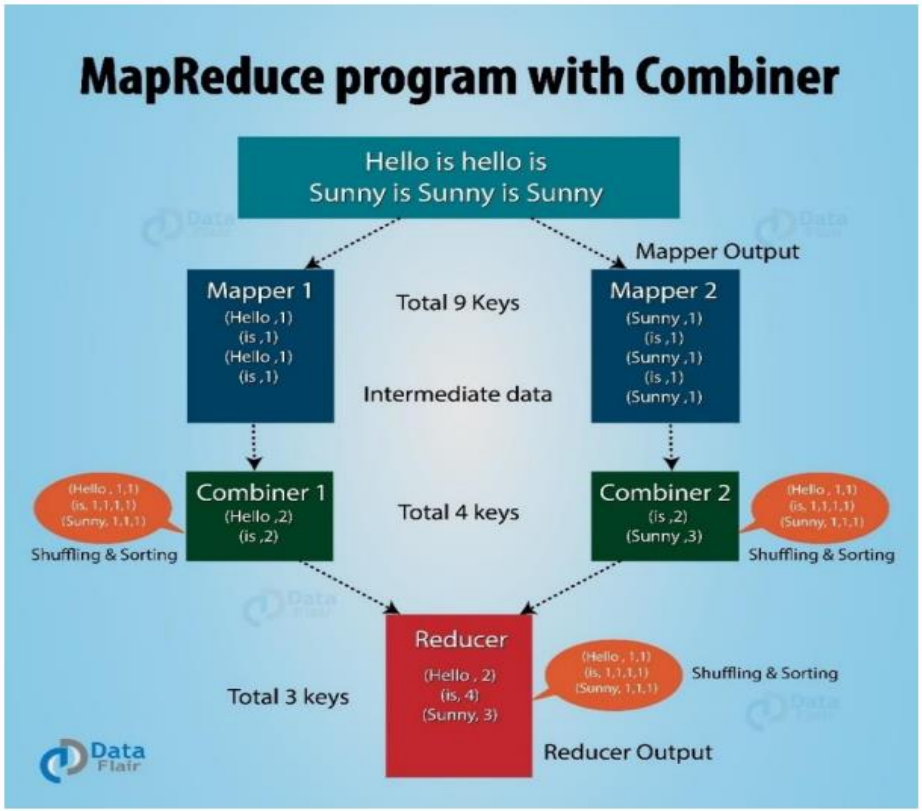
COMBINERS Functions:

- 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.



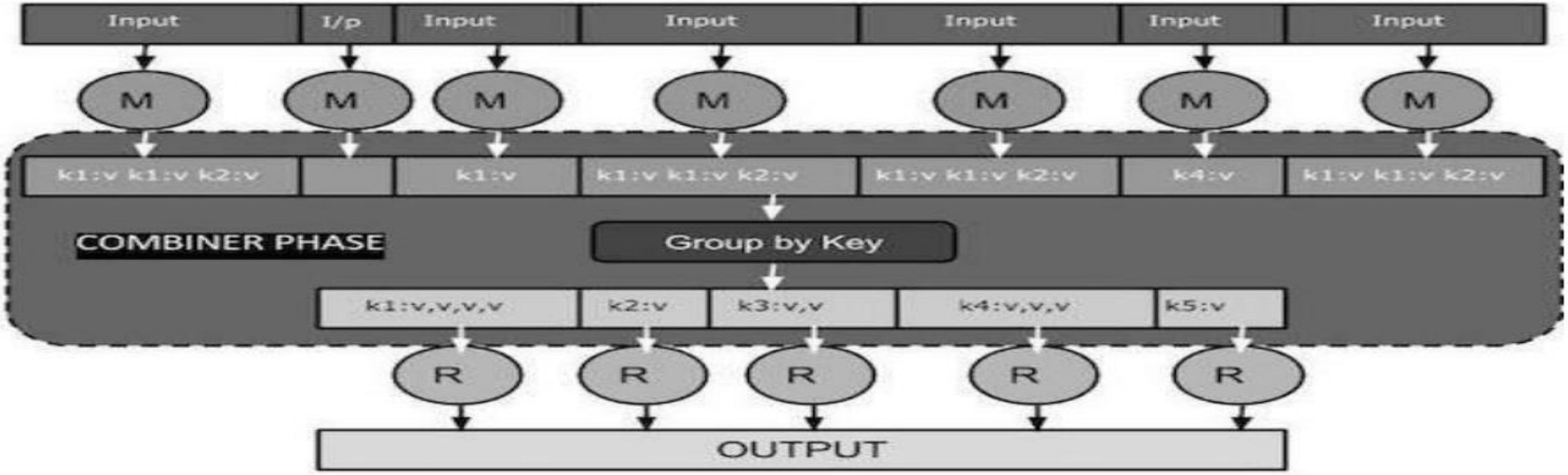
MapReduce without Combiner



MapReduce with Combiner

- Combiner The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

The following MapReduce task diagram shows the COMBINER PHASE.



Combiner Phase

Here is a brief summary on how MapReduce Combiner works

- A combiner does not have a predefined interface and it must implement the Reducer interface's reduce method.
- A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.
- A combiner can produce summary information from a large dataset because it replaces the original Map output.

Although, Combiner is optional yet it helps segregating data into multiple groups for Reduce phase, which makes it easier to process.

Streaming

- Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. For example:

```
mapred streaming \ -
input myInputDirs \ -
output myOutputDir \ -
mapper /bin/cat \ -
reducer /usr/bin/wc
```

- In the above example, both the mapper and the reducer are executables that read the input from stdin (line by line) and emit the output to stdout. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

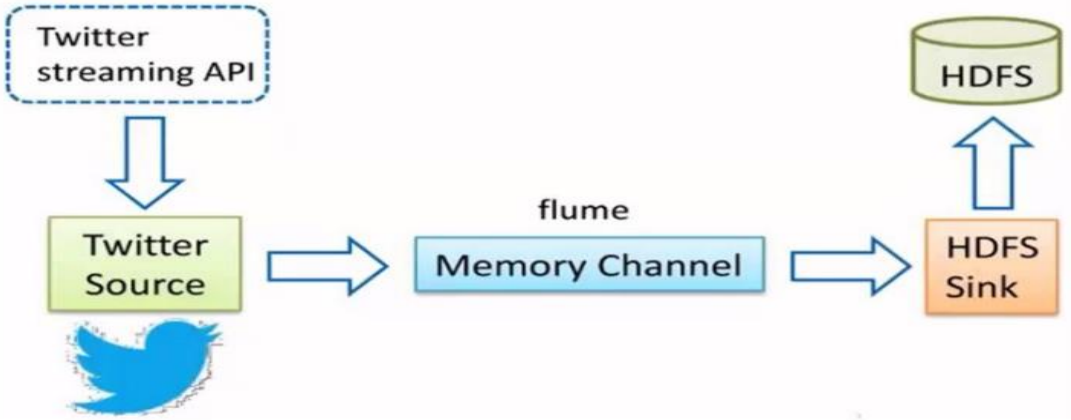
Streaming MapReduce

- Develop using any programming language of your choice, but slightly lower performance and less flexibility than native Java MapReduce.
- An HDFS client is constantly streaming data intoHDFS.
- Both these scenarios have the same interaction with HDFS, except that in the streaming case, the client waits for enough data to fill a data block before writing to HDFS.

You can also use open source Apache streaming technologies like Storm and Spark Streaming in an HDInsight cluster

Data Loading using Flume

Flume is a distributed, reliable and available service for efficiently collecting, aggregating and moving large amounts of streaming event data



Job Scheduling

In Hadoop 2, YARN is separate Daemons for performing Job scheduling, Monitoring, and Resource Management as Application Master, Node Manager, and Resource Manager respectively.

Schedulers and **Applications Manager** are the 2 major components of resource Manager. The Scheduler in YARN is totally dedicated to scheduling the jobs, it can not track the status of the application. On the basis of required resources, the scheduler performs or we can say schedule the Jobs.

A **Job queue** is nothing but the collection of various tasks that we have received from our various clients. The tasks are available in the queue and we need to schedule this task on the basis of our requirements.

There are mainly 3 types of Schedulers in Hadoop:

- FIFO (First In First Out) Scheduler.
- Capacity Scheduler.
- Fair Scheduler.

1. FIFO Scheduler

As the name suggests FIFO i.e. First In First Out, so the tasks or application that comes first will be served first. This is the default Scheduler we use in Hadoop. The tasks are placed in a queue and the tasks are performed in their submission order. In this method, once the job is scheduled, no intervention is allowed. So sometimes the high-priority process has to wait for a long time since the priority of the task does not matter in this method.

Advantage:

- No need for configuration
- First Come First Serve
- simple to execute

Disadvantage:

- Priority of task doesn’t matter, so high priority jobs need to wait
- Not suitable for shared cluster

2. Capacity Scheduler

In Capacity Scheduler we have multiple job queues for scheduling our tasks. The Capacity Scheduler allows multiple occupants to share a large size Hadoop cluster. In Capacity Scheduler corresponding for each job queue, we provide some slots or cluster resources for performing job operation. Each job queue has it's own slots to perform its task. In case we have tasks to perform in only one queue then the tasks of that queue can access the slots of other queues also as they are free to use, and when the new task enters to some other queue then jobs in running in its own slots of the cluster are replaced with its own job.

Capacity Scheduler also provides a level of abstraction to know which occupant is utilizing the more cluster resource or slots, so that the single user or application doesn't take disappropriate or unnecessary slots in the cluster. The capacity Scheduler mainly contains 3 types of the queue that are root, parent, and leaf which are used to represent cluster, organization, or any subgroup, application submission respectively.

Advantage:

- Best for working with Multiple clients or priority jobs in a Hadoop cluster
- Maximizes throughput in the Hadoop cluster

Disadvantage:

- More complex
- Not easy to configure for everyone

3. Fair Scheduler

The Fair Scheduler is very much similar to that of the capacity scheduler. The priority of the job is kept in consideration. With the help of Fair Scheduler, the YARN applications can share the resources in the large Hadoop Cluster and these resources are maintained dynamically so no need for prior capacity. The resources are distributed in such a manner that all applications within a cluster get an equal amount of time. Fair Scheduler takes Scheduling decisions on the basis of memory, we can configure it to work with CPU also.

As we told you it is similar to Capacity Scheduler but the major thing to notice is that in Fair Scheduler whenever any high priority job arises in the same queue, the task is processed in parallel by replacing some portion from the already dedicated slots.

Advantages:

- Resources assigned to each application depend upon its priority.
- it can limit the concurrent running task in a particular pool or queue.

Disadvantages: The configuration is required.

Hadoop I/O

- Unlike any I/O subsystem, Hadoop also comes with a set of primitives. These primitive considerations, although generic in nature, go with the Hadoop IO system as well with some special connotation to it, of course. Hadoop deals with multi-terabytes of datasets; a special consideration on these primitives will give an idea how Hadoop handles data input and output.

Hadoop InputFormat.

How the input files are split up and read in Hadoop is defined by the InputFormat. An Hadoop InputFormat is the first component in Map-Reduce, it is responsible for creating the input splits and dividing them into records.

1. FileInputFormat in Hadoop

It is the base class for all file-based InputFormats. Hadoop FileInputFormat specifies input directory where data files are located. When we start a Hadoop job, FileInputFormat is provided with a path containing files to read. FileInputFormat will read all files and divides these files into one or more InputSplits.

2. TextInputFormat

It is the default InputFormat of MapReduce. TextInputFormat treats each line of each input file as a separate record and performs no parsing. This is useful for unformatted data or line-based records like log files.

- **Key –** It is the byte offset of the beginning of the line within the file (not whole file just one split), so it will be unique if combined with the file name.
- **Value –** It is the contents of the line, excluding line terminators.

3. KeyValueTextInputFormat

It is similar to TextInputFormat as it also treats each line of input as a separate record. While TextInputFormat treats entire line as the value, but the KeyValueTextInputFormat breaks the line itself into key and value by a tab character (‘/t’). Here Key is everything up to the tab character while the value is the remaining part of the line after tab character.

4. SequenceFileInputFormat

Hadoop **SequenceFileInputFormat** is an InputFormat which reads sequence files. Sequence files are binary files that stores sequences of binary **key-value pairs**. Sequence files block-compress and provide direct serialization and deserialization of several arbitrary data types (not just text). Here Key & Value both are user-defined.

5. SequenceFileAsTextInputFormat

Hadoop **SequenceFileAsTextInputFormat** is another form of SequenceFileInputFormat which converts the sequence file key values to Text objects. By calling **'tostring()'** conversion is performed on the keys and values. This InputFormat makes sequence files suitable input for streaming.

6. SequenceFileAsBinaryInputFormat

Hadoop **SequenceFileAsBinaryInputFormat** is a SequenceFileInputFormat using which we can extract the sequence file's keys and values as an opaque binary object.

7. NLineInputFormat

Hadoop **NLineInputFormat** is another form of TextInputFormat where the keys are byte offset of the line and values are contents of the line. Each mapper receives a variable number of lines of input with TextInputFormat and KeyValueTextInputFormat and the number depends on the size of the split and the length of the lines. And if we want our mapper to receive a fixed number of lines of input, then we use NLineInputFormat.N is the number of lines of input that each mapper receives. By default (N=1), each mapper receives exactly one line of input. If N=2, then each split contains two lines. One mapper will receive the first two Key-Value pairs and another mapper will receive the second two key-value pairs.

8. DBInputFormat

Hadoop **DBInputFormat** is an InputFormat that reads data from a relational database, using JDBC. As it doesn't have portioning capabilities, so we need to careful not to swamp the database from which we are reading too many mappers. So it is best for loading relatively small datasets, perhaps for joining with large datasets from HDFS using MultipleInputs. Here Key is LongWritables while Value is DBWritables

Data Integrity

Data integrity means that data should remain accurate and consistent all across its storing, processing, and retrieval operations. To ensure that no data is lost or corrupted during persistence and processing, Hadoop maintains stringent data integrity constraints. Every read/write operation occurs in disks, more so through the network is prone to errors. And, the volume of data that Hadoop handles only aggravates the situation. The usual way to detect corrupt data is through checksums. A ***checksum*** is computed when data first enters into the system and is sent across the channel during the retrieval process. The retrieving end computes the checksum again and matches with the received ones. If it matches exactly then the data deemed to be error free else it contains error. But the problem is – what if the checksum sent itself is corrupt? This is highly unlikely because it is a small data, but not an undeniable possibility. Using the right kind of hardware such as ECC memory can be used to alleviate the situation.

This is mere detection. Therefore, to correct the error, another technique, called CRC (Cyclic Redundancy Check), is used.

Hadoop takes it further and creates a distinct checksum for every 512 (default) bytes of data. Because CRC-32 is 4 bytes only, the storage overhead is not an issue. All data that enters into the system is verified by the datanodes before being forwarded for storage or further processing. Data sent to the datanode pipeline is verified through checksums and any corruption found is immediately notified to the client with *ChecksumException*. The client read from the datanode also goes through the same drill. The datanodes maintain a log of checksum verification to keep track of the verified block. The log is updated by the datanode upon receiving a block verification success signal from the client. This type of statistics helps in keeping the bad disks at bay.

Apart from this, a periodic verification on the block store is made with the help of *DataBlockScanner* running along with the datanode thread in the background. This protects data from corruption in the physical storage media.

Hadoop maintains a copy or replicas of data. This is specifically used to recover data from massive corruption. Once the client detects an error while reading a block, it immediately reports to the datanode about the bad block from the namenode before throwing *ChecksumException*. The namenode then marks it as a bad block and schedules any further reference to the block to its replicas. In this way, the replica is maintained with other replicas and the marked bad block is removed from the system.

For every file created in the Hadoop *LocalFileSystem*, a hidden file with the same name in the same directory with the extension .<*filename*>.*crc* is created. This file maintains the checksum of each chunk of data (512 bytes) in the file. The maintenance of metadata helps in detecting read error before throwing *ChecksumException* by the *LocalFileSystem*.

Compression

Keeping in mind the volume of data Hadoop deals with, compression is not a luxury but a requirement. There are many obvious benefits of file compression rightly used by Hadoop. It economizes storage requirements and is a must-have capability to speed up data transmission over the network and disks. There are many tools, techniques, and algorithms commonly used by Hadoop. Many of them are quite popular and have been used in file compression over the ages. For example, gzip, bzip2, LZO, zip, and so forth are often used.

Serialization: The process that turns structured objects to stream of bytes is called *serialization*. This is specifically required for data transmission over the network or persisting raw data in disks. *Deserialization* is just the reverse process, where a stream of bytes is transformed into a structured object. This is particularly required for object implementation of the raw bytes. Therefore, it is not surprising that distributed computing uses this in a couple of distinct areas: inter-process communication and data persistence.

Hadoop uses RPC (Remote Procedure Call) to enact inter-process communication between nodes. Therefore, the RPC protocol uses the process of serialization and deserialization to render a message to the stream of bytes and vice versa and sends it across the network. However, the process must be compact enough to best use the network bandwidth, as well as fast, interoperable, and flexible to accommodate protocol updates over time.

Hadoop has its own compact and fast serialization format, *Writables*, that MapReduce programs use to generate keys and value types.

Data Structure of Files

There are a couple of high-level containers that elaborate the specialized data structure in Hadoop to hold special types of data. For example, to maintain a binary log, the *SequenceFile* container provides the data structure to persist binary key-value pairs. We then can use the key, such as a timestamp represented by *LongWritable* and value by *Writable*, which refers to logged quantity.

There is another container, a sorted derivation of *SequenceFile*, called *MapFile*. It provides an index for convenient lookups by key.

These two containers are interoperable and can be converted to and from each other.

1. What is Hadoop ? Explain Components and Features of Hadoop?
2. What is HDFS and explain HDFS Architecture with neat sketch ?
3. What is MapReduce and explain with suitable examples?
4. Explain YARN and Combiner Functions, Streaming?
5. Explain Job Scheduling, I/O, Data Integrity?
6. Explain Serialization, File based Data Structures, Replica?