Apache Hadoop

By-Shraddha Nikam

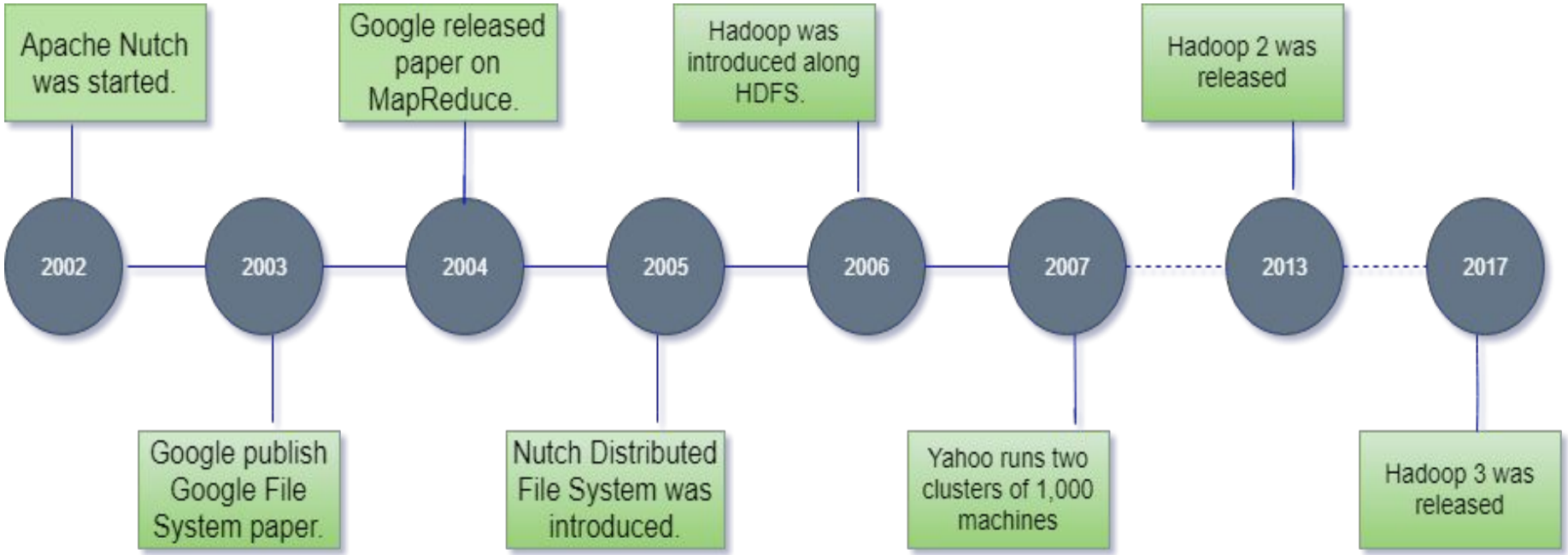
History of Hadoop

The Hadoop was started by Doug Cutting and Mike Cafarella in 2002. Its origin was the Google File System paper, published by Google.

Let's focus on the history of Hadoop in the following steps: -

- In 2002, Doug Cutting and Mike Cafarella started to work on a project, **Apache Nutch.** It is an open source web crawler software project.
- While working on Apache Nutch, they were dealing with big data. To store that data they have to spend a lot of costs which becomes the consequence of that project. This problem becomes one of the important reason for the emergence of Hadoop.
- In 2003, Google introduced a file system known as GFS (Google file system). It is a proprietary distributed file system developed to provide efficient access to data.

- In 2004, Google released a white paper on Map Reduce. This technique simplifies the data processing on large clusters.
- In 2005, Doug Cutting and Mike Cafarella introduced a new file system known as NDFS (Nutch Distributed File System). This file system also includes Map reduce.
- In 2006, Doug Cutting quit Google and joined Yahoo. On the basis of the Nutch project, Dough Cutting introduces a new project Hadoop with a file system known as HDFS (Hadoop Distributed File System). Hadoop first version 0.1.0 released in this year.
- Doug Cutting gave named his project Hadoop after his son's toy elephant.
- In 2007, Yahoo runs two clusters of 1000 machines.
- In 2008, Hadoop became the fastest system to sort 1 terabyte of data on a 900 node cluster within 209 seconds.
- In 2013, Hadoop 2.2 was released.
- In 2017, Hadoop 3.0 was released.


Apache Nutch was started.
Google released paper on MapReduce.
Hadoop was introduced along HDFS.
Hadoop 2 was released
2002
2003
2004
2005
2006
2007
2013
2017
Google publish Google File System paper.
Nutch Distributed File System was introduced.
Yahoo runs two clusters of 1,000 machines
Hadoop 3 was released

Introduction to Hadoop

Hadoop is an open-source software framework that is used for storing and processing large amounts of data in a distributed computing environment.

It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets.

Modules of Hadoop

HDFS: Hadoop Distributed File System. Google published its paper GFS and on the basis of that HDFS was developed. It states that the files will be broken into blocks and stored in nodes over the distributed architecture.

Yarn: Yet another Resource Negotiator is used for job scheduling and manage the cluster.

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

Hadoop Common: These Java libraries are used to start Hadoop and are used by other Hadoop modules.

What is Spark?

Apache Spark is an open-source cluster computing framework. Its primary purpose is to handle the real-time generated data.

Spark was built on the top of the Hadoop MapReduce. It was optimized to run in memory whereas alternative approaches like Hadoop's MapReduce writes data to and from computer hard drives. So, Spark process the data much quicker than other alternatives.

History of Apache Spark

The Spark was initiated by Matei Zaharia at UC Berkeley's AMPLab in 2009. It was open sourced in 2010 under a BSD license.

In 2013, the project was acquired by Apache Software Foundation. In 2014, the Spark emerged as a Top-Level Apache Project.

Features of Apache Spark

- **Fast** - It provides high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.
- **Easy to Use** - It facilitates to write the application in Java, Scala, Python, R, and SQL. It also provides more than 80 high-level operators.
- **Generality** - It provides a collection of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming.
- **Lightweight** - It is a light unified analytics engine which is used for large scale data processing.
- **Runs Everywhere** - It can easily run on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud.

Usage of Spark

- **Data integration:** The data generated by systems are not consistent enough to combine for analysis. To fetch consistent data from systems we can use processes like Extract, transform, and load (ETL). Spark is used to reduce the cost and time required for this ETL process.
- **Stream processing:** It is always difficult to handle the real-time generated data such as log files. Spark is capable enough to operate streams of data and refuses potentially fraudulent operations.
- **Machine learning:** Machine learning approaches become more feasible and increasingly accurate due to enhancement in the volume of data. As spark is capable of storing data in memory and can run repeated queries quickly, it makes it easy to work on machine learning algorithms.
- **Interactive analytics:** Spark is able to generate the respond rapidly. So, instead of running pre-defined queries, we can handle the data interactively.

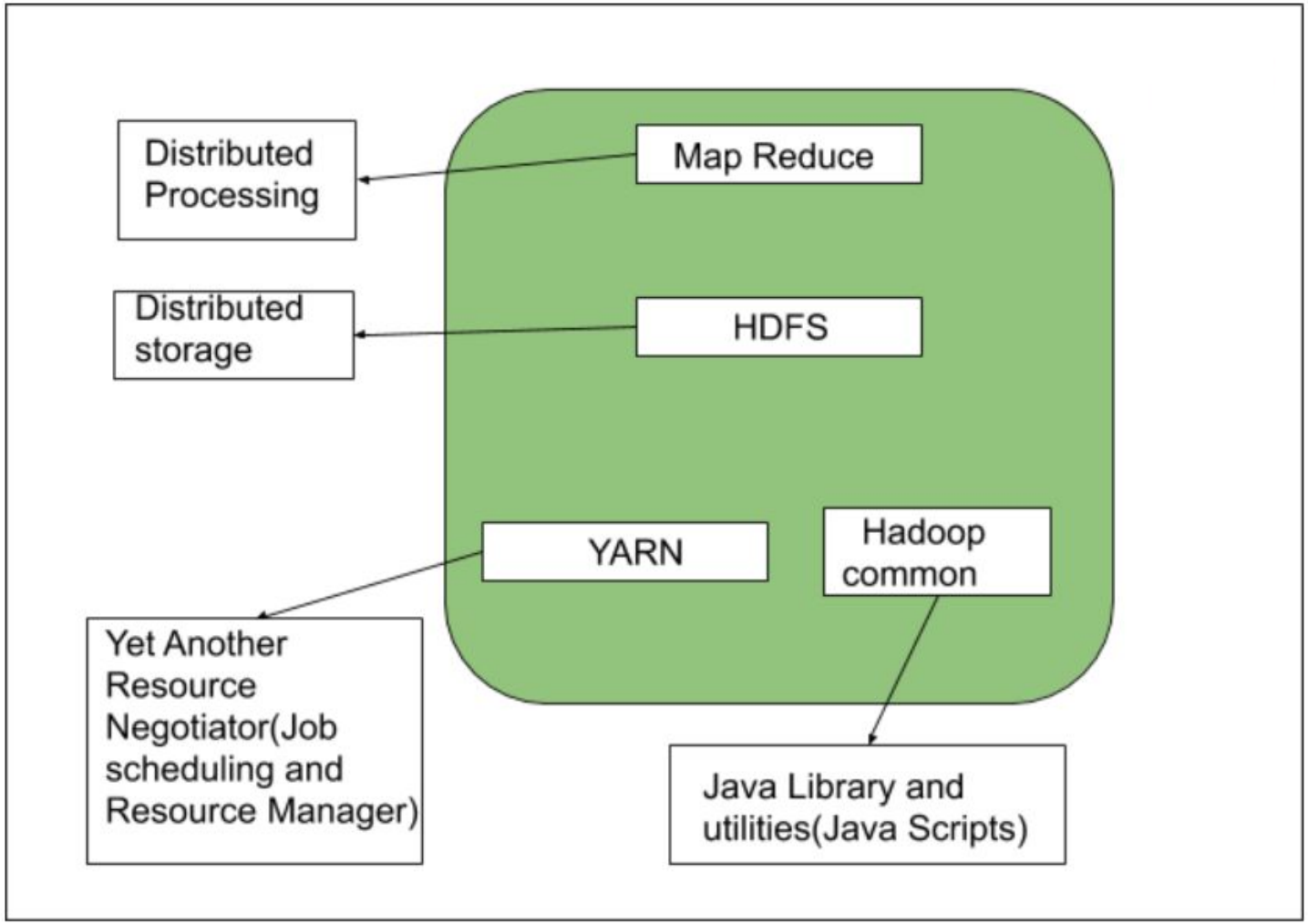
Hadoop Architecture

As we all know Hadoop is a framework written in Java that utilizes a large cluster of commodity hardware to maintain and store big size data.

Hadoop works on MapReduce Programming Algorithm that was introduced by Google. Today lots of Big Brand Companies are using Hadoop in their Organization to deal with big data, eg. Facebook, Yahoo, Netflix, eBay, etc.

The Hadoop Architecture Mainly consists of 4 components.

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

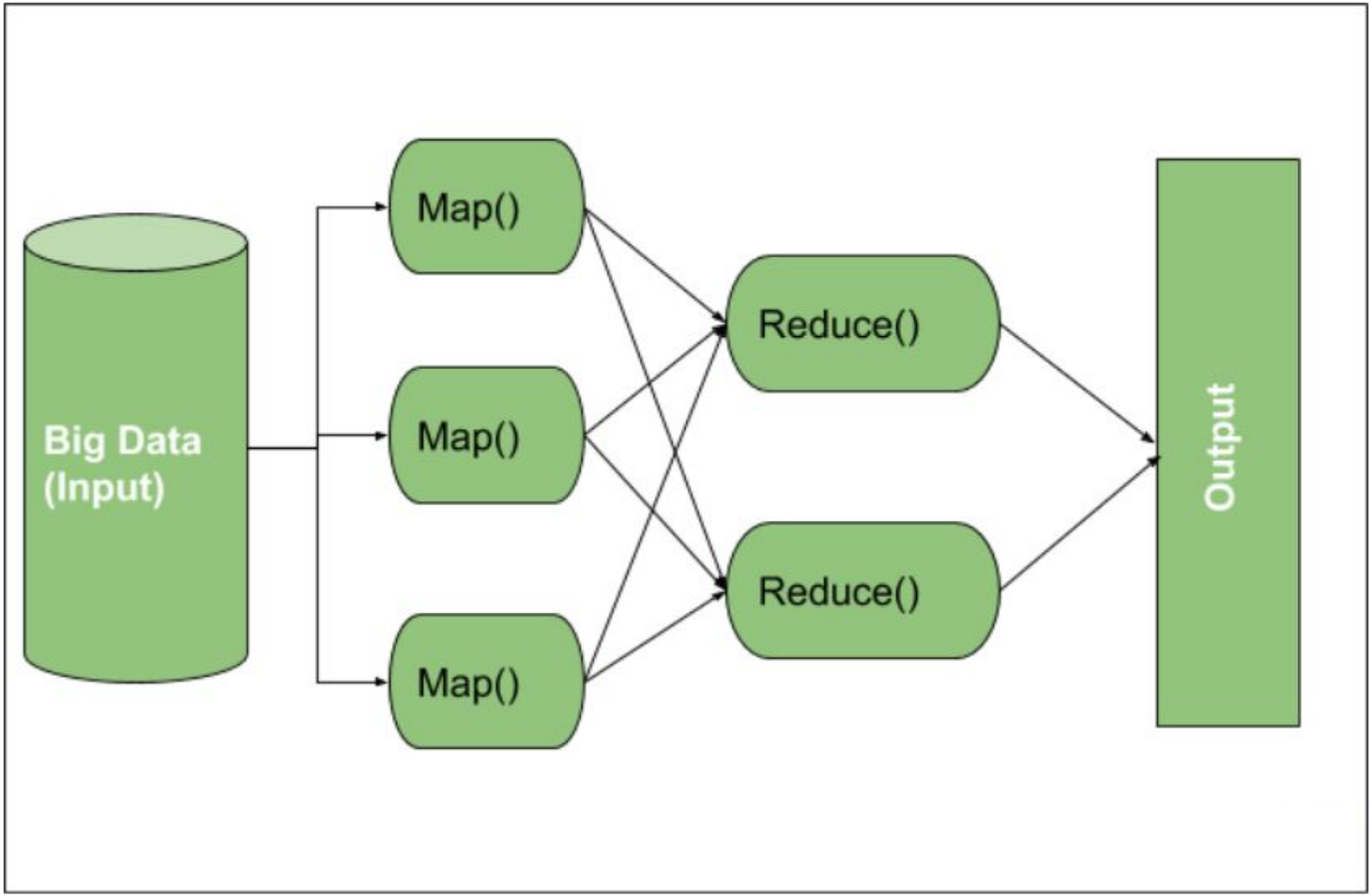

Distributed Processing
Map Reduce
Distributed storage
HDFS
YARN
Hadoop common
Yet Another Resource Negotiator(Job scheduling and Resource Manager)
Java Library and utilities(Java Scripts)

Let’s understand the role of each one of this component in detail.

1. MapReduce

MapReduce nothing but just like an Algorithm or a data structure that is based on the YARN framework. The major feature of MapReduce is to perform the distributed processing in parallel in a Hadoop cluster which Makes Hadoop working so fast. When you are dealing with Big Data, serial processing is no more of any use. MapReduce has mainly 2 tasks which are divided phase-wise:

In first phase, **Map** is utilized and in next phase **Reduce** is utilized.


Big Data
(Input)
Map()
Map()
Map()
Reduce()
Reduce()
Output

Here, we can see that the *Input* is provided to the Map() function then it's *output* is used as an input to the Reduce function and after that, we receive our final output. Let's understand What this Map() and Reduce() does.

As we can see that an Input is provided to the Map(), now as we are using Big Data. The Input is a set of Data.

The **Map()** function here breaks this Data Blocks into **Tuples** that are nothing but a key-value pair. These key-value pairs are now sent as input to the Reduce().

The **Reduce()** function then combines this broken Tuples or key-value pair based on its Key value and form set of Tuples, and perform some operation like sorting, summation type job, etc. which is then sent to the final Output Node. Finally, the Output is Obtained.

The data processing is always done in Reducer depending upon the business requirement of that industry. This is How First Map() and then Reduce is utilized one by one.

Let’s understand the *Map Task* and *Reduce Task* in detail.

Map Task:

- **RecordReader**: The purpose of *recordreader* is to break the records. It is responsible for providing key-value pairs in a Map() function. The key is actually is its locational information and value is the data associated with it.

- **Map:** A map is nothing but a user-defined function whose work is to process the Tuples obtained from record reader. The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.

- **Combiner:** Combiner is used for grouping the data in the Map workflow. It is similar to a Local reducer. The intermediate key-value that are generated in the Map is combined with the help of this combiner. Using a combiner is not necessary as it is optional.

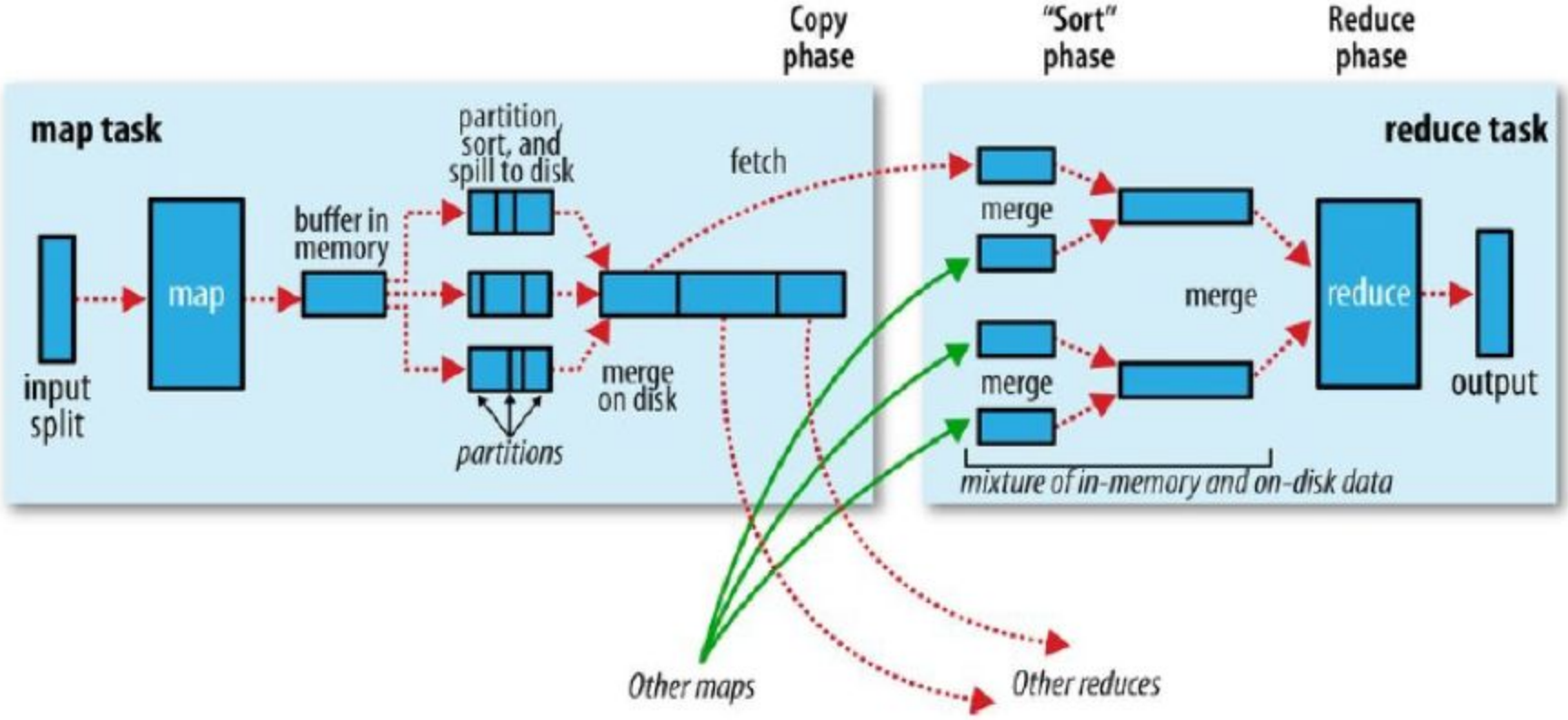
- **Partitionar:** Partitional is responsible for fetching key-value pairs generated in the Mapper Phases. The partitioner generates the shards corresponding to each reducer. Hashcode of each key is also fetched by this partition. Then partitioner performs it's(Hashcode) modulus with the number of reducers(*key.hashcode()%(number of reducers))*.

Reduce Task

- **Shuffle and Sort:** Shuffling is the process by which it transfers the mapper's intermediate output to the reducer. Reducer gets one or more keys and associated values based on reducers. The intermediated key – value generated by the mapper is sorted automatically by key. In Sort phase merging and sorting of the map, the output takes place.Shuffling and Sorting in Hadoop occur simultaneously

Shuffling in MapReduce

The process of moving data from the mappers to reducers is shuffling. Shuffling is also the process by which the system performs the sort. Then it moves the map output to the reducer as input. This is the reason the shuffle phase is required for the reducers. Else, they would not have any input (or input from every mapper). Meanwhile, shuffling can begin even before the map phase has finished. Therefore this saves some time and completes the tasks in lesser time.


Copy phase
"Sort" phase
Reduce phase
map task
reduce task
input split
map
buffer in memory
partition, sort, and spill to disk
partitions
merge on disk
fetch
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces

Sorting in MapReduce

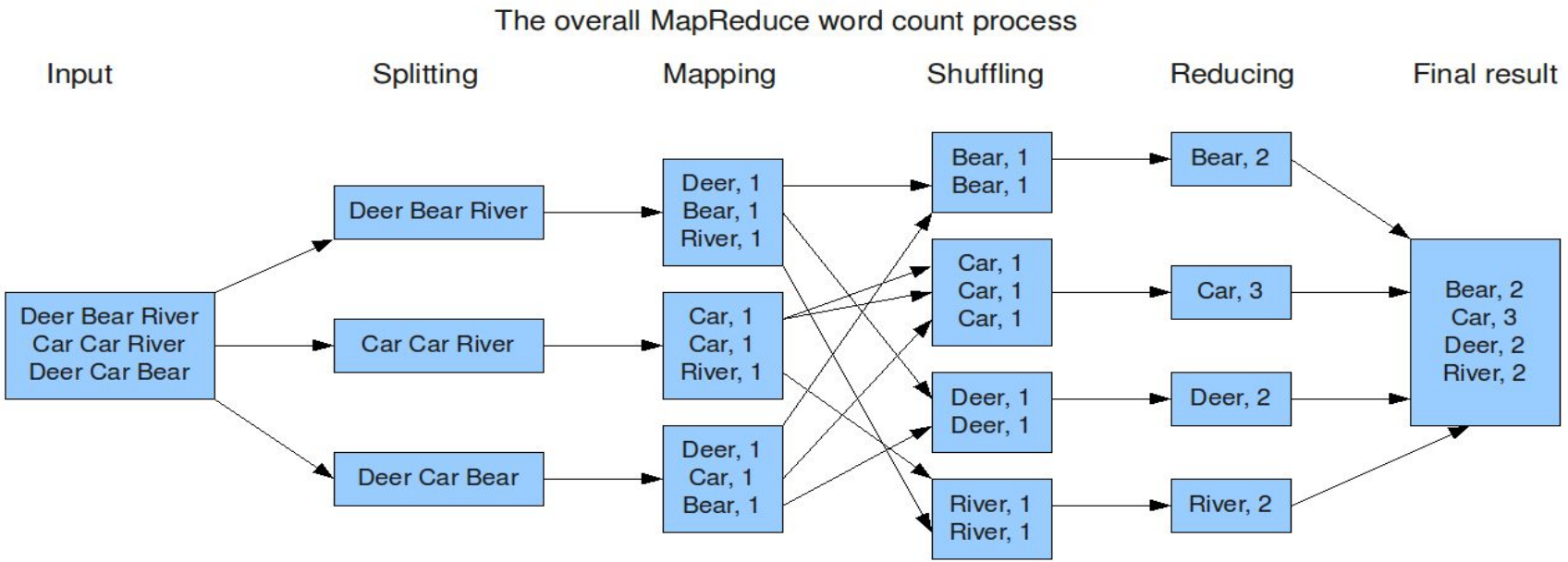
MapReduce Framework automatically sorts the keys generated by the mapper. Therefore, before starting of reducer, all intermediate key-value pairs get sorted by key and not by value. It does not sort values transferred to each reducer. They can be in any order.

Sorting in a MapReduce job helps reducer to easily differentiate when a new reduce task should start. This saves time for the reducer. Reducer in MapReduce begins a new reduce task when the next key in the sorted input data is different from the earlier. Each reduce task takes key-value pairs as input and creates a key-value pair as output.

The crucial thing to note is that shuffling and sorting in Hadoop MapReduce is will not take place at all if you specify zero reducers (setNumReduceTasks(0)). If the reducer is zero, then the MapReduce job stops at the map phase. And the map phase does not comprise any kind of sorting (even the map phase is faster).

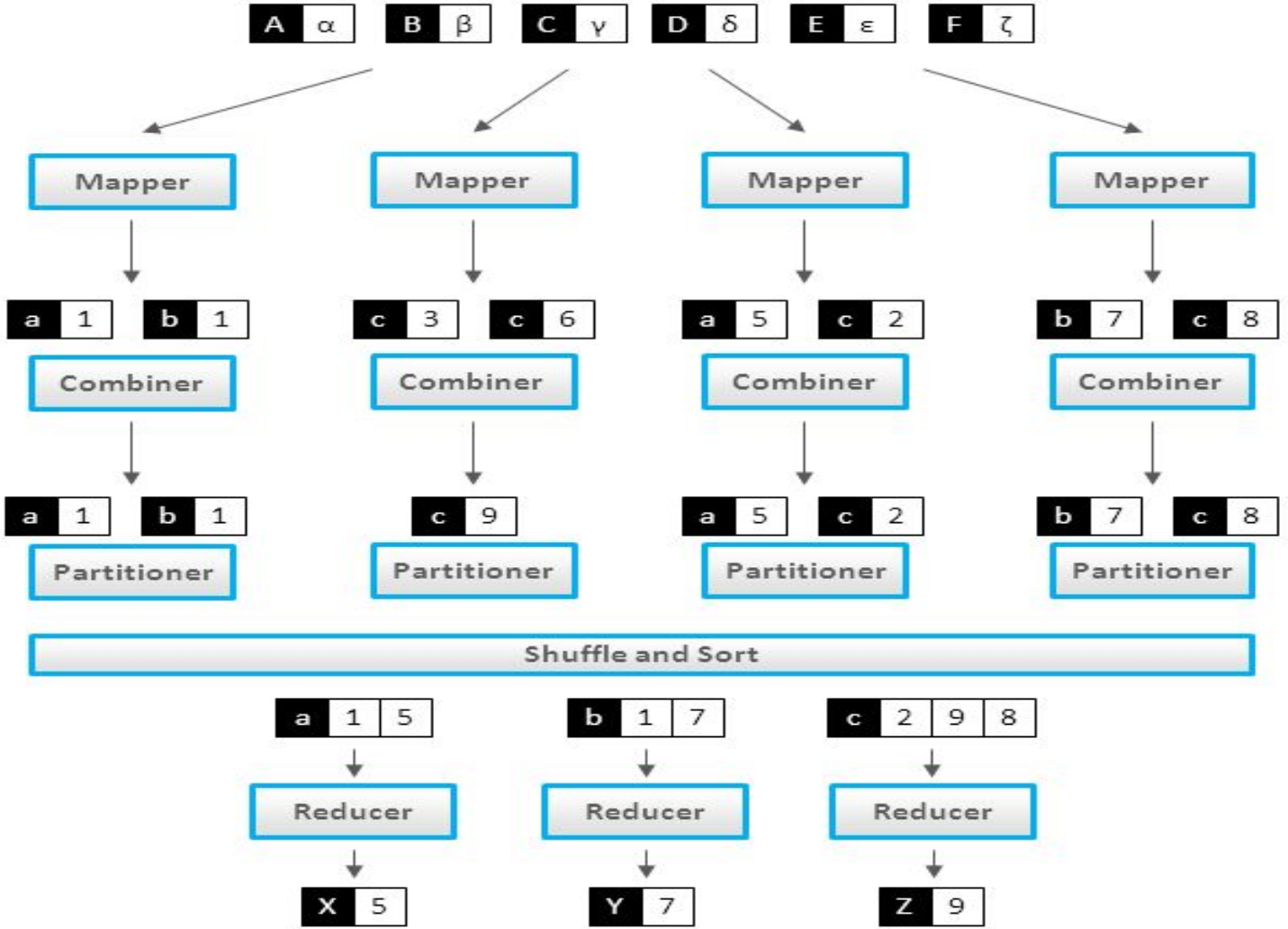
Secondary Sorting in MapReduce

If we need to sort reducer values, then we use a secondary sorting technique. This technique allows us to sort the values (in ascending or descending order) transferred to each reducer.



- **Reduce:** The main function or task of the Reduce is to gather the Tuple generated from Map and then perform some sorting and aggregation sort of process on those key-value depending on its key element.

- **OutputFormat:** Once all the operations are performed, the key-value pairs are written into the file with the help of record writer, each record in a new line, and the key and value in a space-separated manner.


A α
B β
C γ
D δ
E ε
F ζ
Mapper
Mapper
Mapper
Mapper
a 1
b 1
c 3
c 6
a 5
c 2
b 7
c 8
Combiner
Combiner
Combiner
Combiner
a 1
b 1
c 9
a 5
c 2
b 7
c 8
Partitioner
Partitioner
Partitioner
Partitioner
Shuffle and Sort
a 1 5
b 1 7
c 2 9 8
Reducer
Reducer
Reducer
X 5
Y 7
Z 9

2. HDFS(Hadoop Distributed File System)

HDFS(Hadoop Distributed File System) is utilized for storage permission. It is mainly designed for working on commodity Hardware devices(inexpensive devices), working on a distributed file system design. HDFS is designed in such a way that it believes more in storing the data in a large chunk of blocks rather than storing small data blocks.

HDFS in Hadoop provides Fault-tolerance and High availability to the storage layer and the other devices present in that Hadoop cluster.

Data storage Nodes in HDFS.

- NameNode(Master)
- DataNode(Slave)

NameNode:

NameNode works as a Master in a Hadoop cluster that guides the Datanode(Slaves). Namenode is mainly used for storing the Metadata i.e. the data about the data.

Metadata can be the transaction logs that keep track of the user's activity in a Hadoop cluster.

Metadata can also be the name of the file, size, and the information about the location(Block number, Block ids) of Datanode that Namenode stores to find the closest DataNode for Faster Communication.

Namenode instructs the DataNodes with the operation like delete, create, Replicate, etc.

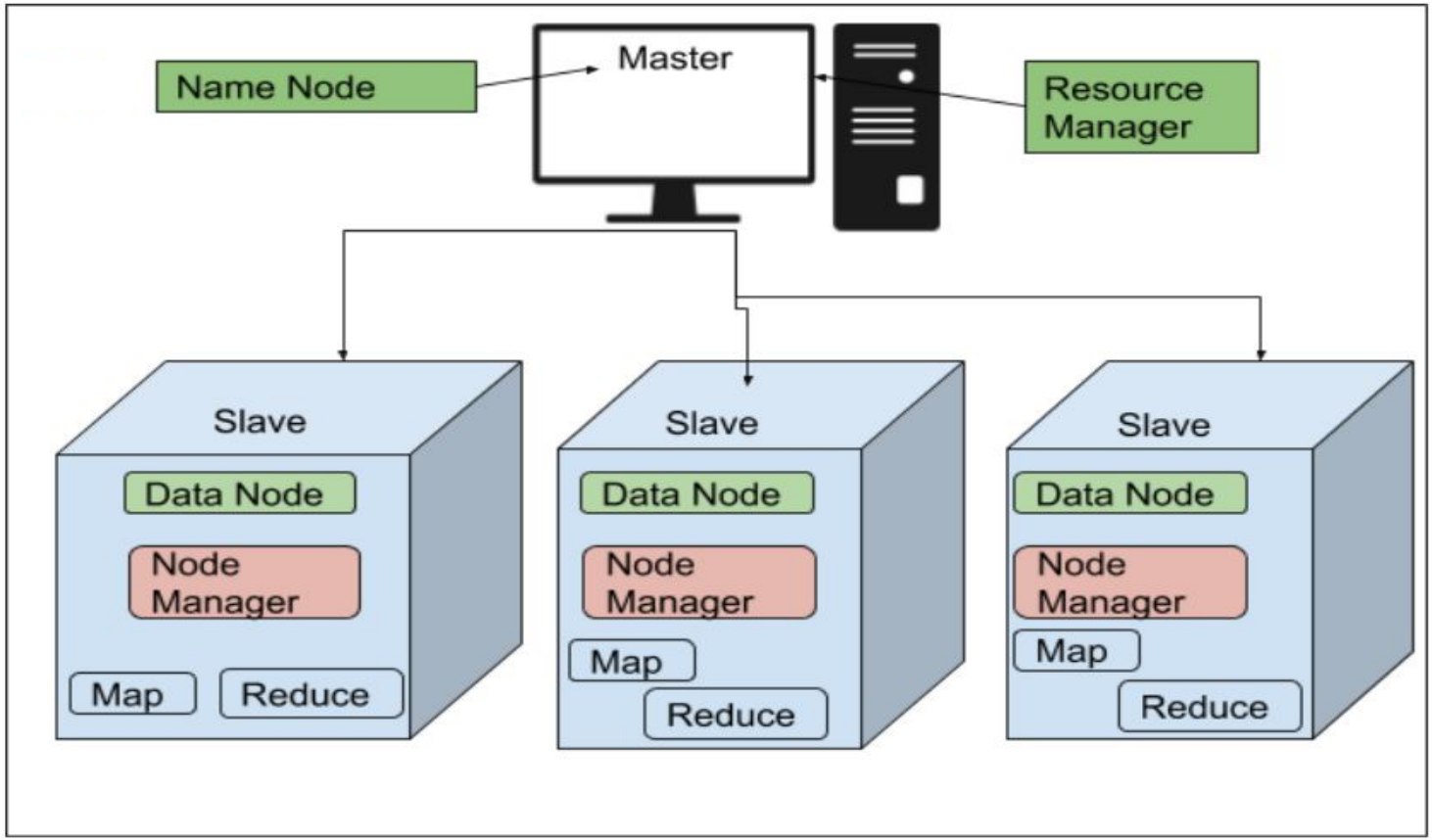
DataNode:

DataNodes works as a Slave DataNodes are mainly utilized for storing the data in a Hadoop cluster, the number of DataNodes can be from 1 to 500 or even more than that.

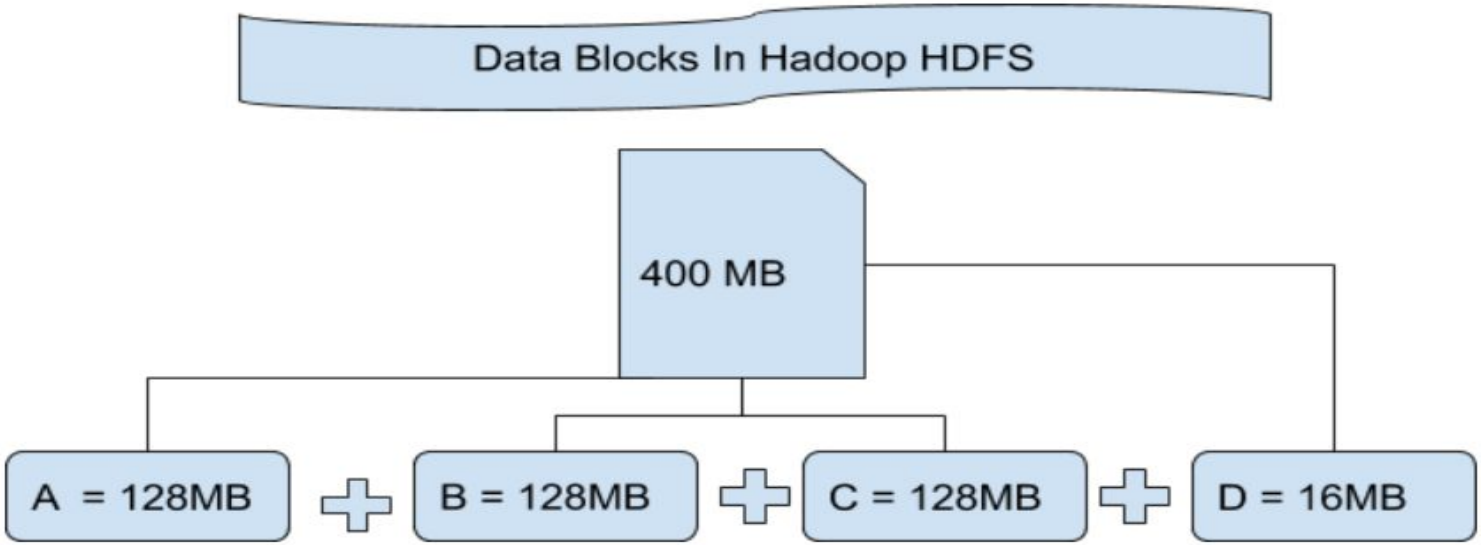
The more number of DataNode, the Hadoop cluster will be able to store more data.

So it is advised that the DataNode should have High storing capacity to store a large number of file blocks.

High Level Architecture Of Hadoop



File Block In HDFS: Data in HDFS is always stored in terms of blocks. So the single block of data is divided into multiple blocks of size 128MB which is default and you can also change it manually.



Let’s understand this concept of breaking down of file in blocks with an example.

Suppose you have uploaded a file of 400MB to your HDFS then what happens is this file got divided into blocks of 128MB+128MB+128MB+16MB = 400MB size. Means 4 blocks are created each of 128MB except the last one.

Hadoop doesn’t know or it doesn’t care about what data is stored in these blocks so it considers the final file blocks as a partial record as it does not have any idea regarding it. In the Linux file system, the size of a file block is about 4KB which is very much less than the default size of file blocks in the Hadoop file system.

As we all know Hadoop is mainly configured for storing the large size data which is in petabyte, this is what makes Hadoop file system different from other file systems as it can be scaled, nowadays file blocks of 128MB to 256MB are considered in Hadoop.

Replication In HDFS :

Replication ensures the availability of the data. Replication is making a copy of something and the number of times you make a copy of that particular thing can be expressed as it's Replication Factor.

As we have seen in File blocks that the HDFS stores the data in the form of various blocks at the same time Hadoop is also configured to make a copy of those file blocks.

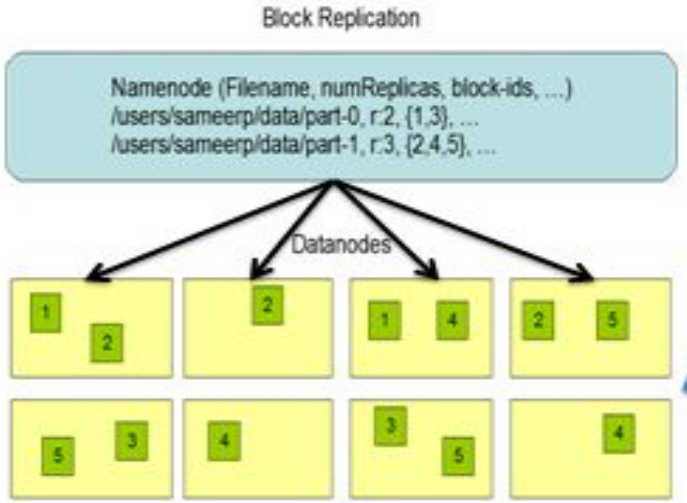
By default, the Replication Factor for Hadoop is set to 3 which can be configured means you can change it manually as per your requirement like in above example we have made 4 file blocks which means that 3 Replica or copy of each file block is made means total of $4 \times 3 = 12$ blocks are made for the backup purpose.

This is because for running Hadoop we are using commodity hardware (inexpensive system hardware) which can be crashed at any time. We are not using the supercomputer for our Hadoop setup.

That is why we need such a feature in HDFS which can make copies of that file blocks for backup purposes, this is known as **fault tolerance.**

Now one thing we also need to notice that after making so many replica's of our file blocks we are wasting so much of our storage but for the big brand organization the data is very much important than the storage so nobody cares for this extra storage.

You can configure the Replication factor in your *hdfs-site.xml* file.



Centralized namenode

- Maintains metadata info about files

File *F*

Blocks (64 MB)

Many datanode (1000s)

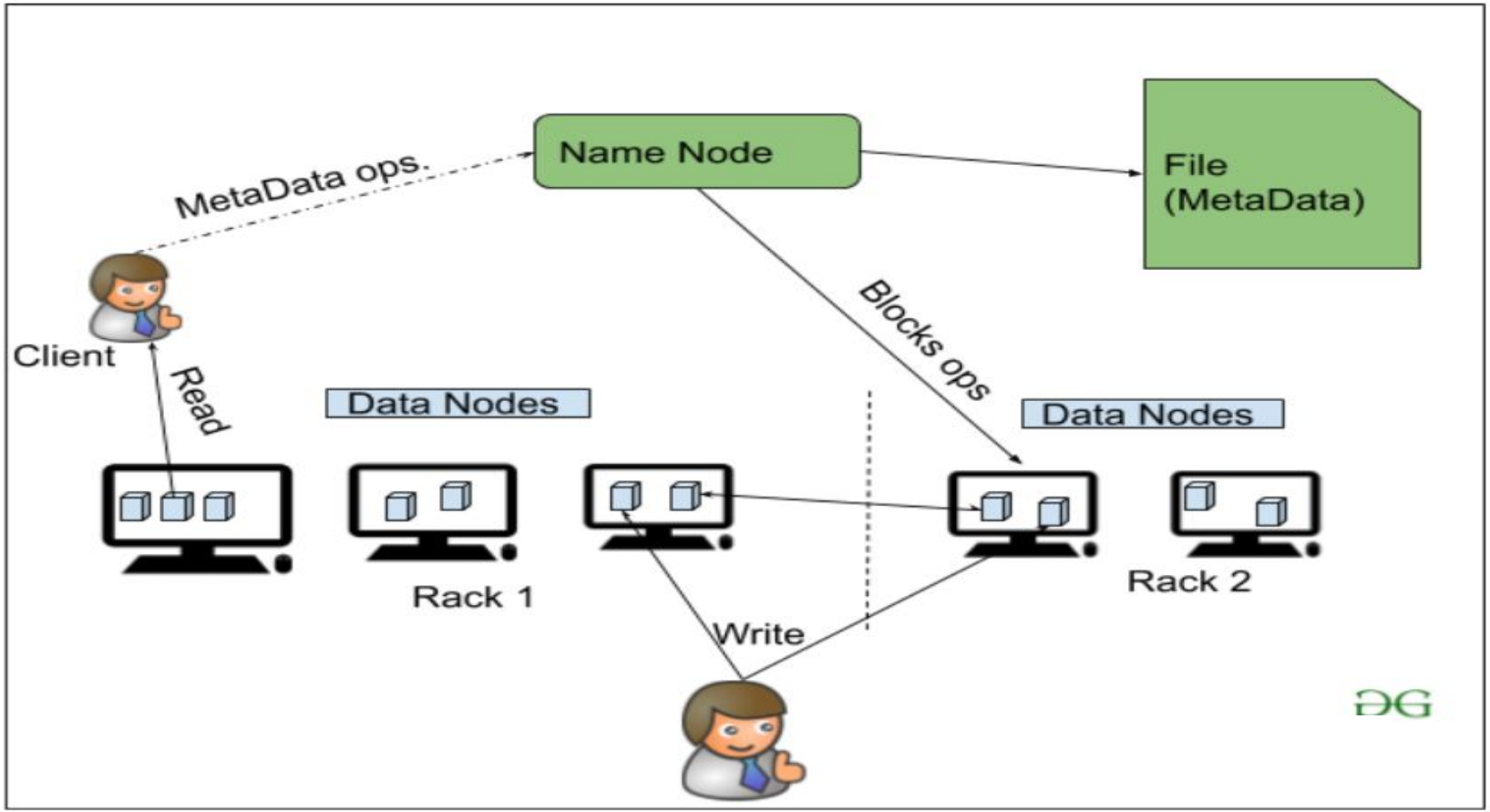
- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

Rack Awareness :

The rack is nothing but just the physical collection of nodes in our Hadoop cluster (maybe 30 to 40). A large Hadoop cluster is consists of so many Racks .

With the help of this Racks information Namenode chooses the closest Datanode to achieve the maximum performance while performing the read/write information which reduces the Network Traffic.

HDFS Architecture



HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.

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 NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.

The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

3. YARN(Yet Another Resource Negotiator)

YARN is a Framework on which MapReduce works. YARN performs 2 operations that are Job scheduling and Resource Management.

The Purpose of Job schedular is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be Maximized.

Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, etc. And the use of Resource Manager is to manage all the resources that are made available for running a Hadoop cluster.

Features of YARN

- Multi-Tenancy
- Scalability
- Cluster-Utilization
- Compatibility

4. Hadoop common or Common Utilities

Hadoop common or Common utilities are nothing but our java library and java files or we can say the java scripts that we need for all the other components present in a Hadoop cluster.

These utilities are used by HDFS, YARN, and MapReduce for running the cluster. Hadoop Common verify that Hardware failure in a Hadoop cluster is common so it needs to be solved automatically in software by Hadoop Framework.

Main Properties of HDFS

• **Large**: A HDFS instance may consist of thousands of server machines, each storing part of the

file system's data

• **Replication**: Each data block is replicated many times (default is 3)

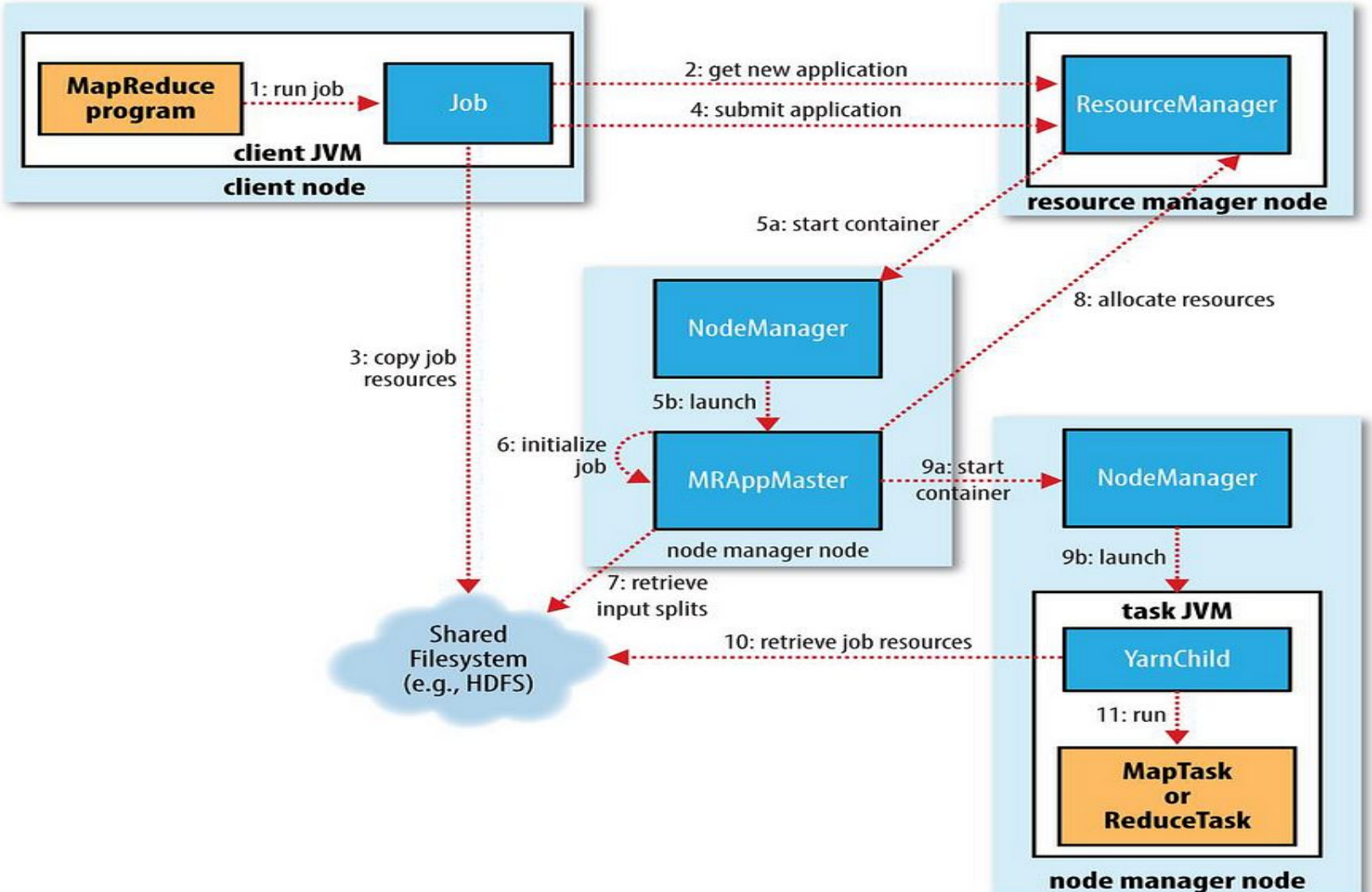
• **Failure**: Failure is the norm rather than exception

• **Fault Tolerance**: Detection of faults and quick, automatic recovery from them is a core

architectural goal of HDFS

– Namenode is consistently checking Datanodes

Anatomy of Map Reduce Job Run


MapReduce program
1: run job
Job
client JVM
client node
2: get new application
4: submit application
ResourceManager
resource manager node
5a: start container
8: allocate resources
NodeManager
5b: launch
3: copy job resources
6: initialize job
MRAppMaster
node manager node
9a: start container
NodeManager
9b: launch
task JVM
YarnChild
11: run
MapTask or ReduceTask
node manager node
7: retrieve input splits
Shared Filesystem (e.g., HDFS)
10: retrieve job resources

There are five independent entities:

- The **client**, which submits the MapReduce job.
- The **YARN resource manager**, which coordinates the allocation of compute resources on the cluster.
- The **YARN node managers**, which launch and monitor the compute containers on machines in the cluster.
- The **MapReduce application master**, which coordinates the tasks running the MapReduce job The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
- The **distributed filesystem**, which is used for sharing job files between the other entities.

1. Job Submission :

- The submit() method on Job creates an internal JobSubmitter instance and calls submitJobInternal() on it.
- Having submitted the job, waitForCompletion polls the job's progress once per second and reports the progress to the console if it has changed since the last report.
- When the job completes successfully, the job counters are displayed Otherwise, the error that caused the job to fail is logged to the console.

The job submission process implemented by JobSubmitter does the following:

- Asks the resource manager for a new application ID, used for the MapReduce job ID.
- Checks the output specification of the job For example, if the output directory has not been specified or it already exists, the job is not submitted and an error is thrown to the MapReduce program.
- Computes the input splits for the job If the splits cannot be computed (because the input paths don't exist, for example), the job is not submitted and an error is thrown to the MapReduce program.
- Copies the resources needed to run the job, including the job JAR file, the configuration file, and the computed input splits, to the shared filesystem in a directory named after the job ID.
- Submits the job by calling submitApplication() on the resource manager.

2. Job Initialization :

- When the resource manager receives a call to its submitApplication() method, it hands off the request to the YARN scheduler.
- The scheduler allocates a container, and the resource manager then launches the application master's process there, under the node manager's management.
- The application master for MapReduce jobs is a Java application whose main class is MRAppMaster .
- It initializes the job by creating a number of bookkeeping objects to keep track of the job's progress, as it will receive progress and completion reports from the tasks.
- It retrieves the input splits computed in the client from the shared filesystem.
- It then creates a map task object for each split, as well as a number of reduce task objects determined by the mapreduce.job.reduces property (set by the setNumReduceTasks() method on Job).

3. Task Assignment:

- If the job does not qualify for running as an uber task, then the application master requests containers for all the map and reduce tasks in the job from the resource manager .
- Requests for map tasks are made first and with a higher priority than those for reduce tasks, since all the map tasks must complete before the sort phase of the reduce can start.
- Requests for reduce tasks are not made until 5% of map tasks have completed.

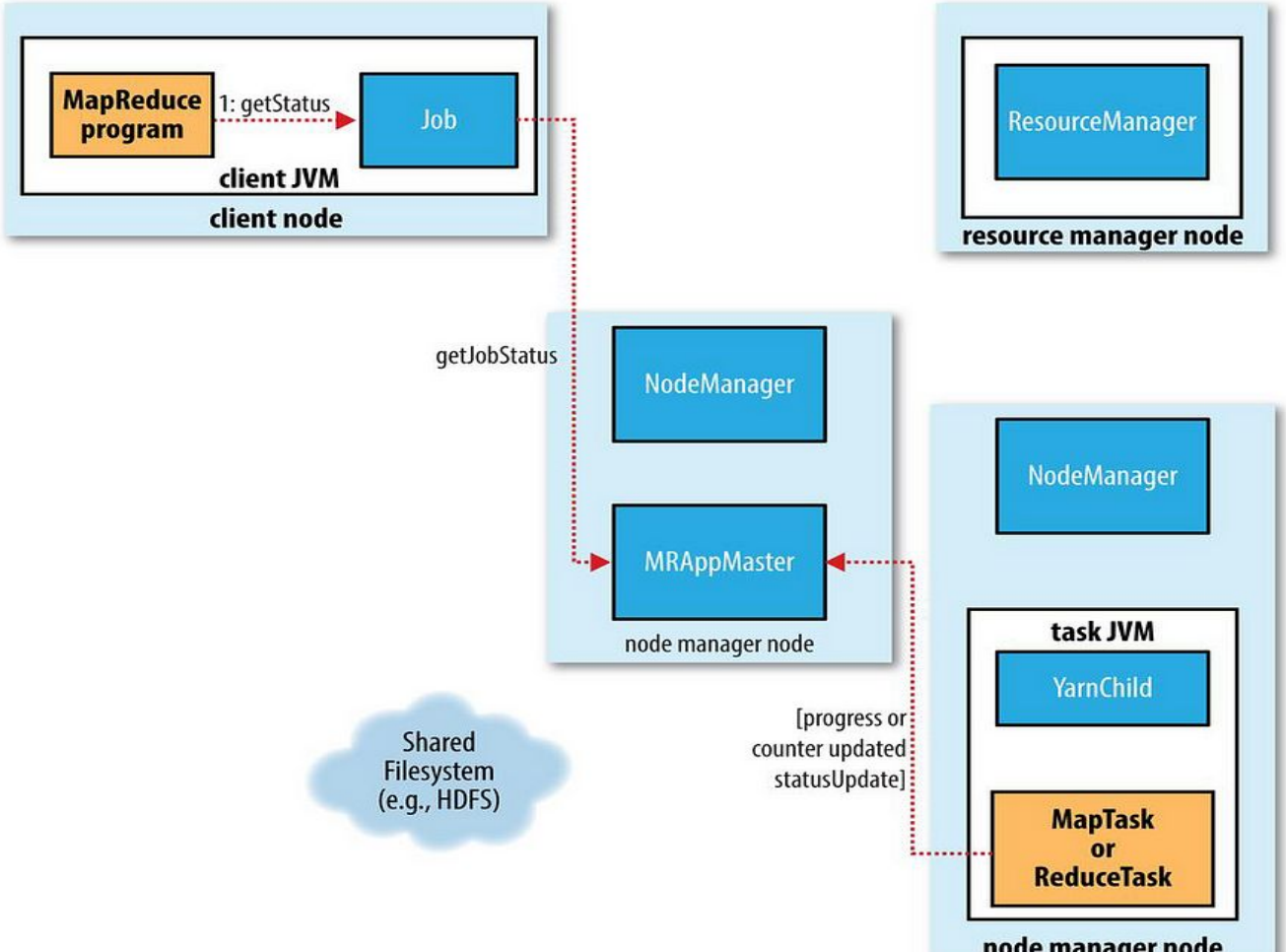
4. Task Execution:

- Once a task has been assigned resources for a container on a particular node by the resource manager's scheduler, the application master starts the container by contacting the node manager.
- The task is executed by a Java application whose main class is YarnChild. Before it can run the task, it localizes the resources that the task needs, including the job configuration and JAR file, and any files from the distributed cache.
- Finally, it runs the map or reduce task.

5. Progress and status updates :

- MapReduce jobs are long running batch jobs, taking anything fromtens of seconds to hours to run.
- A job and each of its tasks have a status, which includes such things as the state of the job or task (e g running, successfully completed, failed), the progress of maps and reduces, the values of the job's counters, and a status message or description (which may be set by user code).
- When a task is running, it keeps track of its progress (i e the proportion of task is completed).
- For map tasks, this is the proportion of the input that has been processed.
- For reduce tasks, it's a little more complex, but the system can still estimate the proportion of the reduce input processed.

How status updates are propagated through the MapReduce System



6. Job Completion:

- When the application master receives a notification that the last task for a job is complete, it changes the status for the job to Successful.
- Then, when the Job polls for status, it learns that the job has completed successfully, so it prints a message to tell the user and then returns from the waitForCompletion() .
- Finally, on job completion, the application master and the task containers clean up their working state and the OutputCommitter's commitJob () method is called.
- Job information is archived by the job history server to enable later interrogation by users if desired.

Failures in MapReduce

There are generally 3 types of failures in MapReduce.

- Task Failure
- TaskTracker Failure
- JobTracker Failure

1. Task Failure

In Hadoop, **task failure** is similar to an employee making a mistake while doing a task. Consider you are working on a large project that has been broken down into smaller jobs and assigned to different employees in your team. If one of the team members fails to do their task correctly, the entire project may be compromised.

Similarly, in Hadoop, if a job fails due to a mistake or issue, it could affect overall data processing, causing delays or faults in the final result.

Reasons for Task Failure

Limited memory: A task can fail if it runs out of memory while processing data.

Failures of disk: If the disk that stores data or intermediate results fails, tasks that depend on that data may fail.

Issues with software or hardware: Bugs, mistakes, or faults in software or hardware components can cause task failures.

How to Overcome Task Failure

Increase memory allocation: Assign extra memory to jobs to ensure they have the resources to process the data.

Implement fault tolerance mechanisms: Using data replication and checkpointing techniques to defend against disc failures and retrieve lost data.

Regularly update software and hardware: Keep the Hadoop framework and supporting hardware up to date to fix bugs, errors, and performance issues that can lead to task failures.

2. TaskTracker Failure

A **TaskTracker** in Hadoop is similar to an employee responsible for executing certain tasks in a large project.

If a TaskTracker fails, it signifies a problem occurred while an employee worked on their assignment. This can interrupt the entire project, much as when a team member makes a mistake or encounters difficulties with their task, producing delays or problems with the overall project's completion.

To avoid TaskTracker failures, ensure the TaskTracker's hardware and software are in excellent working order and have the resources they need to do their jobs successfully.

Reasons for TaskTracker Failure

Hardware issues: Just as your computer's parts can break or stop working properly, the TaskTracker's hardware (such as the processor, memory, or disc) might fail or stop operating properly. This may prohibit it from carrying out its duties.

Software problems or errors: The software operating on the TaskTracker may contain bugs or errors that cause it to cease working properly. It's similar to when an app on your phone fails and stops working properly.

Overload or resource exhaustion: It may struggle to keep up if the TaskTracker becomes overburdened with too many tasks or runs out of resources such as memory or processing power. It's comparable to being overburdened with too many duties or running out of storage space on your gadget.

How to Overcome TaskTracker Failure

Update software and hardware on a regular basis: Keep the Hadoop framework and associated hardware up to date to correct bugs, errors, and performance issues that might lead to task failures.

Upgrade or replace hardware: If TaskTracker's hardware is outdated or insufficiently powerful, try upgrading or replacing it with more powerful components. It's equivalent to purchasing a new, upgraded computer to handle jobs more efficiently.

Restart or reinstall the program: If the TaskTracker software is causing problems, a simple restart or reinstall may be all that is required. It's the same as restarting or reinstalling an app to make it work correctly again.

3. JobTracker Failure

A **JobTracker** in Hadoop is similar to a supervisor or manager that oversees the entire project and assigns tasks to TaskTrackers (employees).

If a JobTracker fails, it signifies the supervisor is experiencing a problem or has stopped working properly. This can interrupt the overall project's coordination and development, much as when a supervisor is unable to assign assignments or oversee their completion.

To avoid JobTracker failures, it is critical to maintain the JobTracker's hardware and software, ensure adequate resources, and fix any issues or malfunctions as soon as possible to keep the project going smoothly.

Reasons for JobTracker Failure

Database connectivity: The JobTracker stores job metadata and state information in a backend database (usually Apache Derby or MySQL). JobTracker failures can occur if there are database connectivity issues, such as network problems or database server failures.

Security problems: JobTracker failures can be caused by security issues such as authentication or authorization failures, incorrectly configured security settings or key distribution and management issues.

How to Overcome JobTracker Failure

Avoiding Database Connectivity: To avoid database connectivity failures in the JobTracker, ensure optimized database configuration, robust network connections, and high availability techniques are implemented. Retrying connections, monitoring, and backups are all useful.

To overcome security-related problems: implement strong authentication and authorization, enable SSL/TLS for secure communication, keep software updated with security patches, follow key management best practices, conduct security audits, and seek expert guidance for vulnerability mitigation and compliance with security standards.

Job Scheduling

- In Hadoop, we can receive multiple jobs from different clients to perform. The Map-Reduce framework is used to perform multiple tasks in parallel in a typical Hadoop cluster to process large size datasets at a fast rate. This Map-Reduce Framework is responsible for scheduling and monitoring the tasks given by different clients in a Hadoop cluster. But this method of scheduling jobs is used prior to **Hadoop 2**.
- Now in Hadoop 2, we have YARN (Yet Another Resource Negotiator). In YARN we have separate Daemons for performing Job scheduling, Monitoring, and Resource Management as Application Master, Node Manager, and Resource Manager respectively.

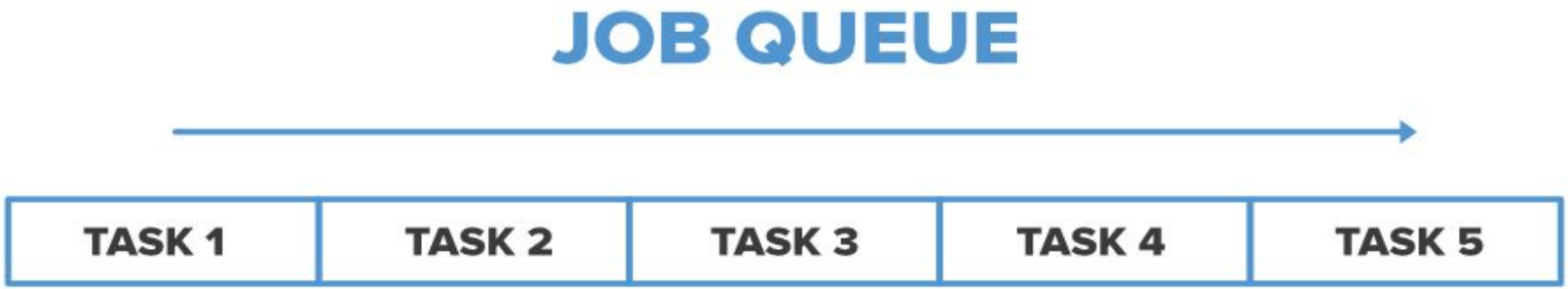
- Here, Resource Manager is the Master Daemon responsible for tracking or providing the resources required by any application within the cluster, and Node Manager is the slave Daemon which monitors and keeps track of the resources used by an application and sends the feedback to Resource Manager.
- **Schedulers** 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.

There are mainly 3 types of Schedulers in Hadoop:

1. FIFO (First In First Out) Scheduler.
2. Capacity Scheduler.
3. Fair Scheduler.

These Schedulers are actually a kind of algorithm that we use to schedule tasks in a Hadoop cluster when we receive requests from different-different clients.

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



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.

FIFO SCHEDULER

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.

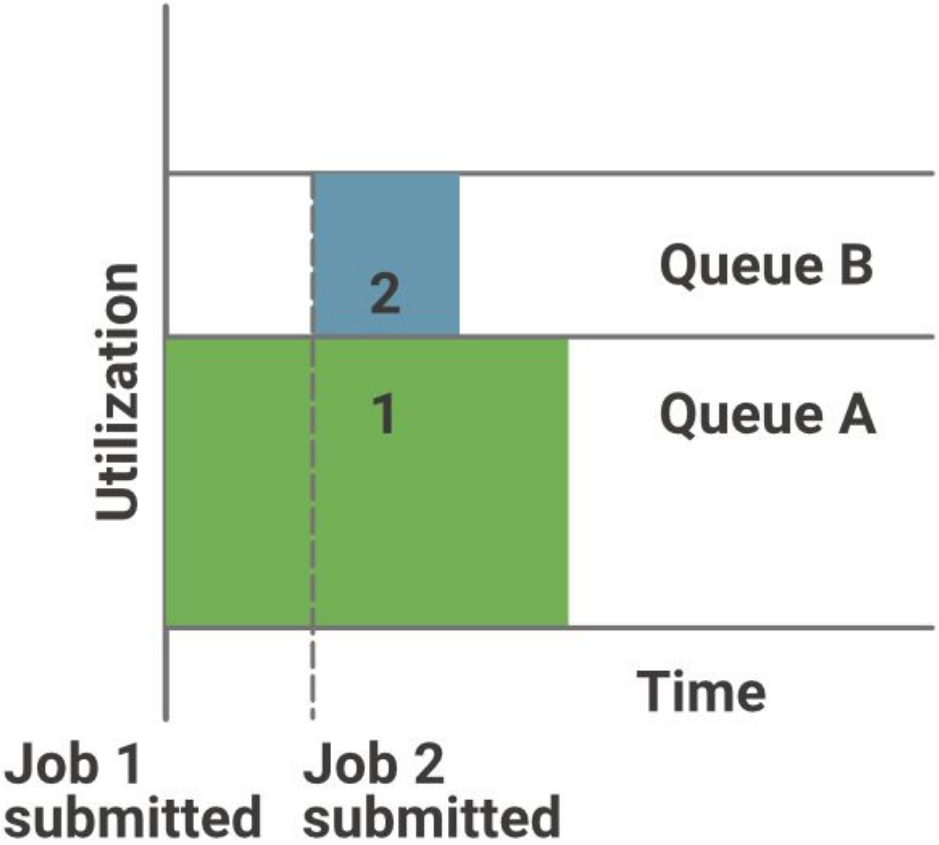
In Capacity Scheduler corresponding for each job queue, we provide some slots or cluster resources for performing job operation. Each job queue has it’s own slots to perform its task.

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.

CAPACITY SCHEDULER



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

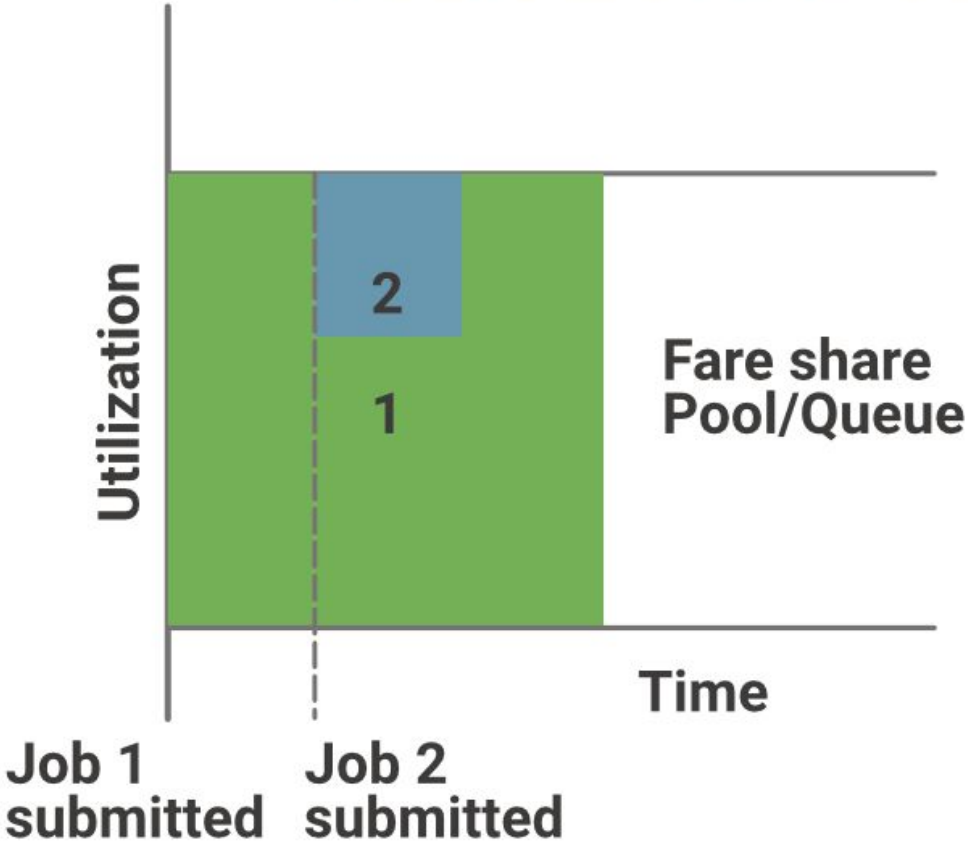
3. Fair Scheduler

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.

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.

FAIR SCHEDULER



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.

When to use Each Job Scheduling in MapReduce

- The capacity scheduler is the correct choice because we want to secure guaranteed access with the potential in order to reuse unused capacity.

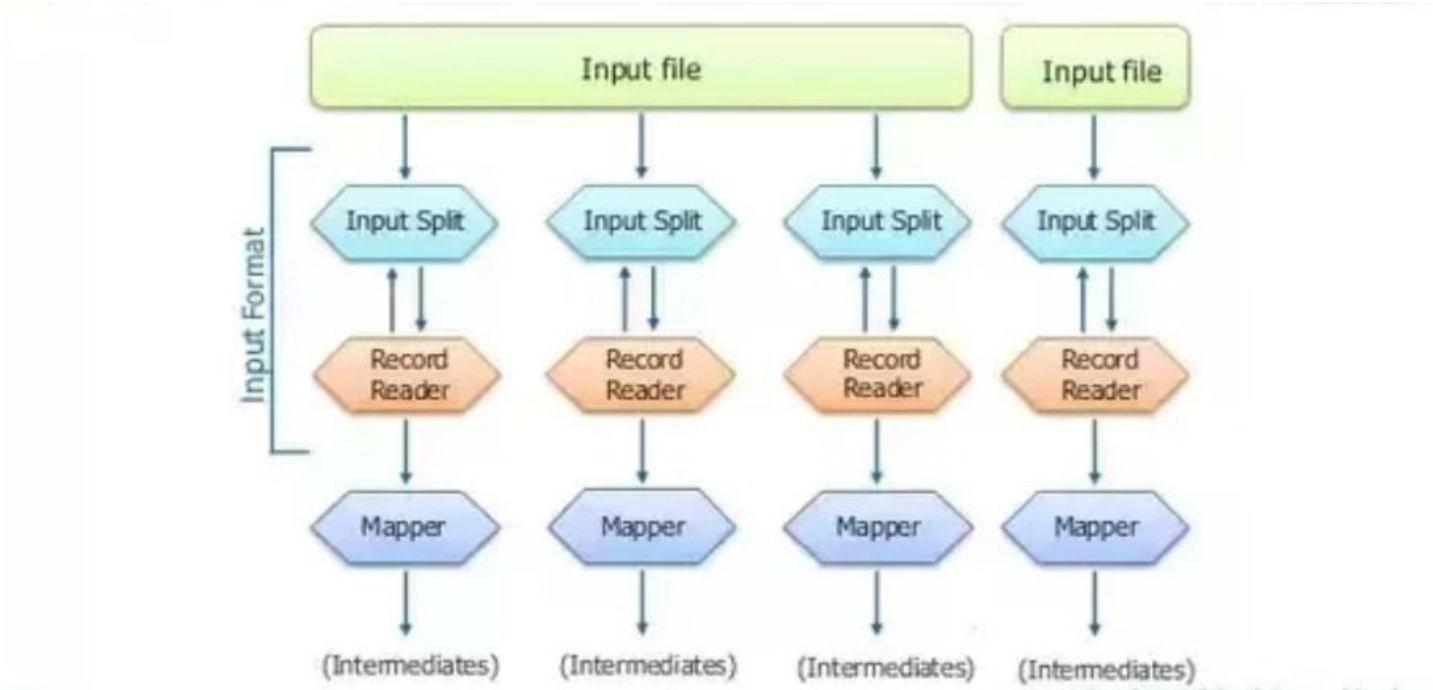
- The fair scheduler works well when we use large and small clusters for the same organization with limited workloads. Also, it is helpful in the presence of various jobs.

Map Reduce Types and Format

Input Format

- InputFormat takes care about how input file is split and read by Hadoop.
- It uses input format interface and TextInputFormat is the default.
- Each Input file is broken into splits and each map processes a single split. Each Split is further divided into records of key/value pairs which are processed by map tasks one record at a time.
- Record reader creates key/value pairs from input splits and writes on context, which will be shared with Mapper class.
- The InputFormat class is one of the fundamental classes in hadoop mapreduce framework which provide following functionality.
    - The file
    - Data splits
    - RecorReader

Input Format



Types of Input File Format

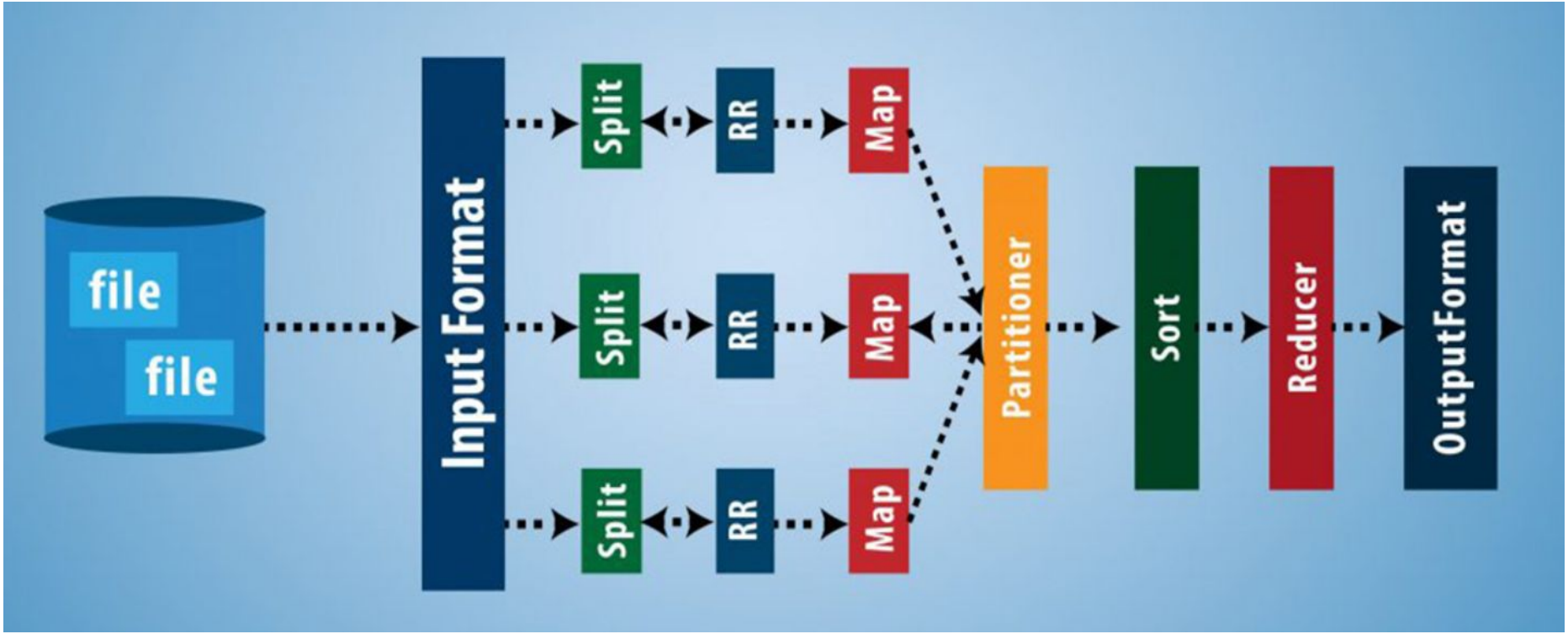
- **FileInputFormat:** It is the base class for all file-based Input Formats. It specifies input directory where data files are located. It will read all files and divides these files into one or more Input Splits.
- **TextInputFormat:** Each line in the text file is a record.Key:Byte offset of line Value: Content of the line.
- **KeyValueTextInputFormat:** Everything before the separator is the key, and everything after is value.
- **SequenceFileInputFormat:** To read any sequence files. Key and values are user defined.
- **SequenceFileasTextInputFormat:** Similar to SequenceFileInputFormat. It converts sequence file key values to text objects.

- **SequenceFileasBinaryInputFormat:** To read any sequence files. It is used to extract sequence files keys and values as opaque binary object.
- **NLineInputFormat:** Similar to TextInputFormat, But each split is guaranteed to have exactly N lines.
- **DBInputFormat:** To read data from Relational Database Service (RDS) . Key is LongWrittables and values are DB Writable.

Output Formats

- The OutputFormat checks the Output-Specification for execution of the Map- Reduce job. For e.g check that the output directory doesn’t already exist.
- It determines how RecordWriter Implementation is used to write output to output files. Output Files are stored in a File System.
- The OutputFormat decides the way the output key-value pairs are written in the output files by RecordWriter.

Output Format



Types of Output Formats

- **TextOutputFormat** : It is the MapReduce default Hadoop reducer Output Format which writes key,value pairs on individual lines of text files.
- **SequenceFileOutputFormat** : It writes sequences files for its output and it is the intermediate format use between MapReduce jobs.
- **MapFileOutputFormat**: It writes output as map files.The key in the MapFile must be added in order to ensure that the reducer emits keys in sorted order.
- **MultipleOutputs** : It allows writing data to files whose names are derived from the output keys and values.
- **LazyOutputFormat :** It is a wrapper OutputFormat which ensures that the output file will be created only when the record is emitted for a given partition
- **DBOutputFormat** : It writes to the relational database and HBase and sends the reduce output to a SQL Table.

Map Reduce Features

- **Counters** - They are a useful channel for gathering statistics about the job like quality control.
- Hadoop maintains some built in counters for every job that report various metrics for your job.
- *Types of counters* - Task counters, Job counters, User-Defined Java Counters.
- **Sorting** - Ability to sort data is at the heart of MapReduce.
- *Types of sorts* - Partial sort, Total sort, Secondary sort
- For any particular key, values are not sorted.
- **Joins** - MapReduce can perform joins between large datasets, but writing code to do joins from scratch is fairly involved. Ex: Map-side joins, Reduced-side join.
- Basic idea is that the mapper tags each record with its source and uses the join key as map output key, so that the records with same key are brought together in the reducer.

- **Side Data Distribution** - It can be defined as extra read only data needed by job to process the main dataset.
- Challenge is to make side data available to all the map or reduce tasks in convenient and efficient fashion.
- **Using the Job Configuration -** We can set arbitrary key-value pairs in the job configuration using various setter methods on Configuration.
- **Distributed Cache -** Rather than serializing side data in job configuration, it is preferable to distribute datasets using Hadoop's distributed cache mechanism.
- **Distributed Cache API -** Most applications don’t need to use distributed cache API as they can use the cache via GenericOptionsParser.
- **MapReduce Library Classes -** Hadoop comes with library of mappers and reducers for commonly used functions.

Spark

• Spark is a fast and general-purpose cluster computing system.

• It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.

• It also supports a rich set of higher-level tools including:

- MLlib for machine learning,
- GraphX for graph processing,
- Spark Streaming,
- Spark SQL for SQL and structured data processing.

- Spark uses Hadoop which is a popular open-source implementation of MapReduce.
- Hadoop works in a master/slave architecture for both distributed storage and distributed computation.
- Hadoop Distributed File System (HDFS) is responsible for distributed storage

Programming Implementation with Spark

Although Spark comes from MapReduce and appear to solve traditional Hadoop MapReduce slow disc reading problem, it have little different programming logic.

In Hadoop the typical points are:

1. Usually using key-value pair to be map and reduce function input and output

2. One key is for one reduce in Reducer class process

3. Every mapper or reducer may process every kind of key or value instead of using a standard data format.

While in Spark, the input is a resilient distributed dataset. And also there is no obvious key-value pair, instead, we use Scala's tuple. This kind of tuple is made by (a, b) language logic such as (line.length, 1).

One of a traditional map function in Hadoop is expressed into and **Resilient Distributed Dataset (**RDD), (line.length, 1) tuple. If one RDD owns tuples, it relay other function like reduceByKey() to handle it.

The first task to make this programming is to create a sparkContext object to tell Spark how to access the clusters.

Then use RDD function API which is called JavaRDD and JavaPairRDD to transform the input file and call mapToPair function to write our word detect and count method to process new Tuples with words (key) and numbers (value). After that use reduceByKey to run total and count the numbers and give result