Unit 5 MapReduce

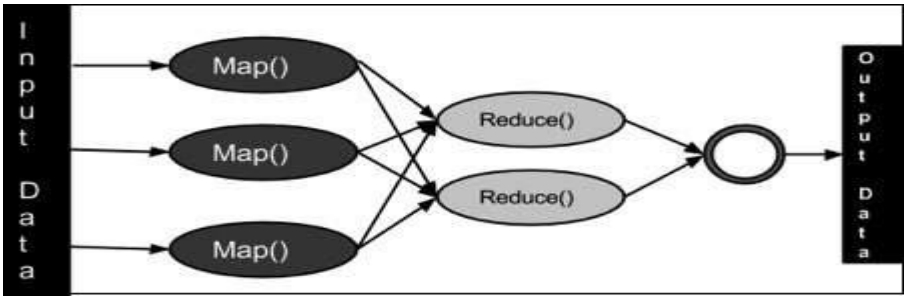
MapReduce

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in two stages, namely map stage and reduce stage.
    - **Map stage** − The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
    - **Reduce stage** − This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



Matrix-Vector Multiplication Algorithm by MapReduce

Suppose we have an n×n matrix M, whose element in row i and column j will be denoted mij . Let us consider the matrix multiplication example to visualize MapReduce. Consider the following matrix:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Here matrix A is a 2×2 matrix which means the number of rows(i)=2 and the number of columns(j)=2. Matrix B is also a 2×2 matrix where number of rows(j)=2 and number of columns(k)=2. Each cell of the matrix is labelled as Aij and Bij. Ex. element 3 in matrix A is called A21 i.e. 2nd-row 1st column. Now One step matrix multiplication has 1 mapper and 1 reducer. The Formula is:

Mapper for Matrix A (k, v)=((i, k), (A, j, Aij)) for all k
Mapper for Matrix B (k, v)=((i, k), (B, j, Bjk)) for all i

Therefore computing the mapper for Matrix A:

k, i, j computes the number of times it occurs.

Here all are 2, therefore when k=1, i can have

2 values 1 & 2, each case can have 2 further

values of j=1 and j=2. Substituting all values

in formula

k=1 i=1 j=1 ((1, 1), (A, 1, 1))
j=2 ((1, 1), (A, 2, 2))
i=2 j=1 ((2, 1), (A, 1, 3))
j=2 ((2, 1), (A, 2, 4))

k=2 i=1 j=1 ((1, 2), (A, 1, 1))
j=2 ((1, 2), (A, 2, 2))
i=2 j=1 ((2, 2), (A, 1, 3))
j=2 ((2, 2), (A, 2, 4))

Computing the mapper for Matrix B

i=1 j=1 k=1 ((1, 1), (B, 1, 5))

k=2 ((1, 2), (B, 1, 6))

j=2 k=1 ((1, 1), (B, 2, 7))

k=2 ((1, 2), (B, 2, 8))

i=2 j=1 k=1 ((2, 1), (B, 1, 5))

k=2 ((2, 2), (B, 1, 6))

j=2 k=1 ((2, 1), (B, 2, 7))

k=2 ((2, 2), (B, 2, 8))

The formula for Reducer is:

Reducer(k, v)=(i, k)=>Make sorted Alist and Blist
*(i, k) => Summation (Aij * Bjk)) for j*
Output =>((i, k), sum)

Therefore computing the reducer:

We can observe from Mapper computation

that 4 pairs are common (1, 1), (1, 2),

(2, 1) and (2, 2)

Make a list separate for Matrix A &

B with adjoining values taken from

Mapper step above:

(1, 1) =>Alist ={(A, 1, 1), (A, 2, 2)}

Blist ={(B, 1, 5), (B, 2, 7)}

Now Aij x Bjk: [(1*5) + (2*7)] =19 -------(i)

(1, 2) =>Alist ={(A, 1, 1), (A, 2, 2)}

Blist ={(B, 1, 6), (B, 2, 8)}

Now Aij x Bjk: [(1*6) + (2*8)] =22 -------(ii)

(2, 1) =>Alist ={(A, 1, 3), (A, 2, 4)}

Blist ={(B, 1, 5), (B, 2, 7)}

Now Aij x Bjk: [(3*5) + (4*7)] =43 -------(iii)

(2, 2) =>Alist ={(A, 1, 3), (A, 2, 4)}

Blist ={(B, 1, 6), (B, 2, 8)}

Now Aij x Bjk: [(3*6) + (4*8)] =50 -------(iv)

From (i), (ii), (iii) and (iv) we conclude that

((1, 1), 19)

((1, 2), 22)

((2, 1), 43)

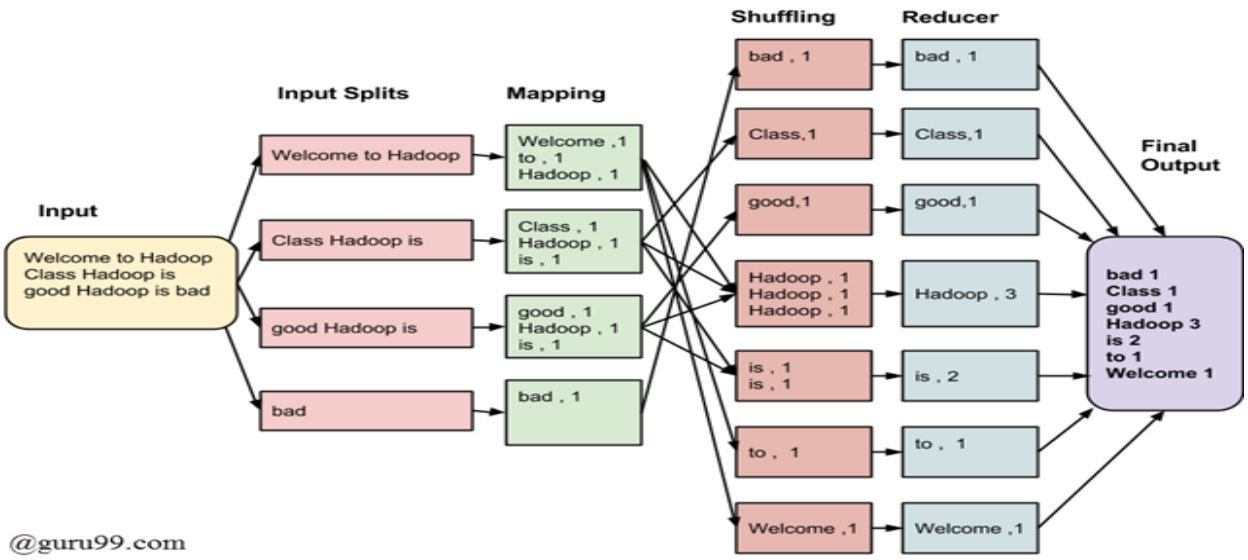
((2, 2), 50)

Therefore the Final Matrix is:

$$\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Word Count

MapReduce Word Count is a framework which splits the chunk of data, sorts the map outputs and input to reduce tasks. A File-system stores the output and input of jobs. Re-execution of failed tasks, scheduling them and monitoring them is the task of the framework.



The data goes through the following phases of MapReduce in Big Data

Input Splits:

An input to a MapReduce in Big Data job is divided into fixed-size pieces called **input splits** Input split is a chunk of the input that is consumed by a single map

Mapping

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

Shuffling

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.

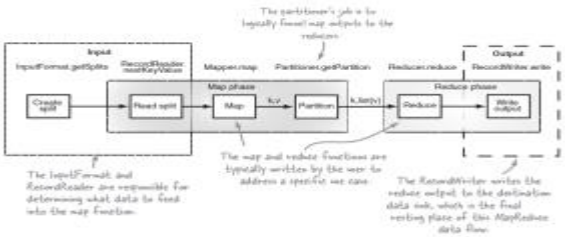
Reducing

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

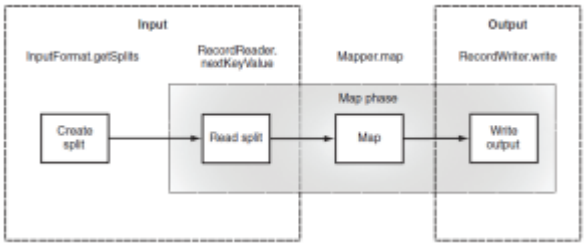
UNDERSTANDING INPUTS AND OUTPUTS IN MAPREDUCE

Your data might be XML files sitting behind a number of FTP servers, text log files sitting on a central web server, or Lucene indexes1 in HDFS. How does MapReduce support reading and writing to these different serialization structures across the various storage mechanisms? You'll need to know the answer in order to support a specific serialization format.



Data input :-

The two classes that support data input in MapReduce are InputFormat and Record-Reader. The InputFormat class is consulted to determine how the input data should be partitioned for the map tasks, and the RecordReader performs the reading of data from the inputs.



INPUT FORMAT :-
Every job in MapReduce must define its inputs according to contracts specified in the InputFormat abstract class. InputFormat implementers must fulfill three contracts: first, they describe type information for map input keys and values; next, they specify how the input data should be partitioned; and finally, they indicate the RecordReader instance that should read the data from source.

RECORDREADER :-
The RecordReader class is used by MapReduce in the map tasks to read data from an input split and provide each record in the form of a key/value pair for use by mappers. A task is commonly created for each input split, and each task has a single RecordReader that's responsible for reading the data for that input split.

Data output :-
MapReduce uses a similar process for supporting output data as it does for input data.Two classes must exist, an OutputFormat and a RecordWriter. The OutputFormat performs some basic validation of the data sink properties, and the RecordWriter writes each reducer output to the data sink.

OUTPUT FORMAT:-
Much like the InputFormat class, the OutputFormat class, as shown in figure 3.5, defines the contracts that implementers must fulfill, including checking the information related to the job output, providing a RecordWriter, and specifying an output committer, which allows writes to be staged and then made "permanent" upon task and/or job success.

RECORD WRITER:-
You'll use the RecordWriter to write the reducer outputs to the destination data sink.

Data Serialization

Serialization is the process of translating data structures or objects state into binary or textual form to transport the data over network or to store on some persisten storage. Once the data is transported over network or retrieved from the persistent storage, it needs to be deserialized again. Serialization is termed as **marshalling** and deserialization is termed as **unmarshalling**.

Storage formats are a way to define how to store information in the file. Most of the time, assume this information is from the extension of the data. Both structured and unstructured data can store on HADOOP-enabled systems. Common HDFS file formats are -

- Plain text storage
- Sequence files
- RC files
- AVRO
- Parquet

How to enable data serialization in it?

- Data serialization is a process that converts structure data manually back to the original form.
- Serialize to translate data structures into a stream of data. Transmit this stream of data over the network or store it in DB regardless of the system architecture.
- Isn't storing information in binary form or stream of bytes is the right approach.
- Serialization does the same but isn't dependent on architecture.

Consider CSV files contains a comma (,) in between data, so while Deserialization, wrong outputs may occur. Now, if metadata is stored in XML form, a self- architected form of data storage, data can easily deserialize.

Areas of Serialization for Storage Formats

To maintain the proper format of a data serialization system must have the following four properties -

- **Compact -** helps in the best use of network bandwidth
- **Fast -** reduces the performance overhead
- **Extensible -** can match new requirements
- **Inter-operable -** not language-specific

It has two areas -

Interprocess communication
When a client calls a function or subroutine from one pc to the pc in-network or server, that calling is a remote procedure call.

Persistent storage
It is better than java's inbuilt serialization as java serialization isn't compact. Serialization and Deserialization of data help maintain and manage corporate decisions for effective use of resources and data available in Data warehouse or any other database -writable - language specific to java.

Why Data Serialization?

- To process records faster (Time-bound).
- When proper data formats need to maintain and transmit over data without schema support on another end.
- Now when in the future, data without structure or format needs to process, complex Errors may occur.
- Serialization offers data validation over transmission.

Introduction to YARN

YARN is an Apache Hadoop technology and stands for **Y**et **A**nother **R**esource **N**egotiator. It is the Cluster management component of Hadoop 2.0. YARN is a large-scale, distributed operating system for big data applications. YARN is a resource manager created by separating the processing engine and the management function of MapReduce. It monitors and manages workloads, maintains a multi-tenant environment, manages the high availability features of Hadoop, and implements security controls. Different Yarn applications can co-exist on the same cluster so MapReduce, HBase, Spark all can run at the same time bringing great benefits for manageability and cluster utilization.

Need for YARN/Limitations of Hadoop 1.0/ Why to use YARN in Hadoop ?

Despite being thoroughly proficient at data processing and computations, Hadoop 1.x had some shortcomings like delays in batch processing, scalability issues, availability issues, and Multi tenancy issues as listed below..

- ✓ Single Name Node is responsible for managing entire namespace for Hadoop clusters.
- ✓ It has restricted processing model which is suitable for batch oriented MapReduce jobs
- ✓ Hadoop MapReduce is not suitable for interactive analysis.
- ✓ Hadoop 1.0 is not suitable for Machine Learning algorithms, graphs and other memory intensive algorithms.
- ✓ MapReduce is responsible for cluster resource management and data processing as it relied on MapReduce for processing big datasets.
- ✓ With YARN, Hadoop is able to support a variety of processing approaches and has a larger array of applications.
- ✓ Hadoop YARN clusters are able to run stream data processing and interactive querying side by side with MapReduce batch jobs.

YARN framework runs even the non-MapReduce applications, thus overcoming the shortcomings of Hadoop 1. Hadoop 2.x is YARN based architecture. It is general processing platform. YARN is not constrained to MapReduce only. One can run multiple applications in Hadoop 2.x in which all applications share common resource management.

YARN Features: YARN gained popularity because of the following features-

- ✓ **Scalability:** The scheduler in Resource manager of YARN architecture allows Hadoop to extend and manage thousands of nodes and clusters.
- ✓ **Compatibility:** Applications created use the MapReduce framework that can be run easily on YARN. YARN supports the existing map-reduce applications without disruptions thus making it compatible with Hadoop 1.0 as well.
- ✓ **Cluster Utilization:** YARN allocates all cluster resources efficiently and dynamically, which leads to better utilization of Hadoop as compared to the previous version of it.
- ✓ **Multi-tenancy:** It allows multiple engine access that can efficiently work together all because of YARN as it is a highly versatile technology.
- ✓ **High availability:** High availability of NameNode is obtained with the help of Passive Standby NameNode.

MapReduce: MapReduce is the Hadoop layer that is responsible for data processing. It writes an application to process unstructured and structured data stored in HDFS. It is responsible for the parallel processing of high volume of data by dividing data into independent tasks. The processing is done in two phases Map and Reduce. The Map is the first phase of processing that specifies complex logic code and the Reduce is the second phase of processing that specifies lightweight operations.

The key aspects of Map Reduce are:

- ✓ Computational frame work
- ✓ Splits a task across multiple nodes
- ✓ Processes data in parallel

MapReduce Vs YARN

MapReduce is Programming Model, YARN is architecture for distribution cluster. Hadoop 2 using YARN for resource management. Besides that, Hadoop support programming model which support parallel processing that we known as MapReduce. Difference between MapReduce and YARN are given below..

Criteria	YARN	MapReduce
Responsibility	YARN is responsible for managing the resources among applications in the cluster.	MapReduce is the processing framework for processing vast data in the Hadoop cluster in a distributed manner.
Type of processing	Real-time, batch, and interactive processing with multiple engines	Silo and batch processing with a single-engine
Cluster resource optimization	Excellent due to central resource management	Average due to fixed Map and Reduce slots
Suitable for	MapReduce and non-MapReduce applications	Only MapReduce applications
Managing cluster resource	Done by YARN	Done by Job Tracker
Namespace	Hadoop supports multiple namespaces	Supports only one namespace, i.e., HDFS

YARN Architecture:

The fundamental idea behind the YARN(Yet Another Resource Negotiator) architecture is to splitting the JobTracker responsibility of resource management and job scheduling/monitoring into separate daemons.

Basic concepts of YARN are Application and Container.

- ✓ **Application** is a job submitted to system. **Eg:** MapReduce job.
- ✓ **Container:** Basic unit of allocation. Replaces fixed map/reduce slots. Finegrained resource allocation across multiple resource type. **Eg**. Container_0: 2GB, 1CPU Container_1: 1GB, 6CPU.

The main components of YARN architecture includes:

Client: For submitting MapReduce jobs.

Resource Manager: To manage the use of resources across the cluster.The main responsibility of Global Resource Manager is to distribute resources among various applications. It has two main components: *Scheduler and Application Manager.*

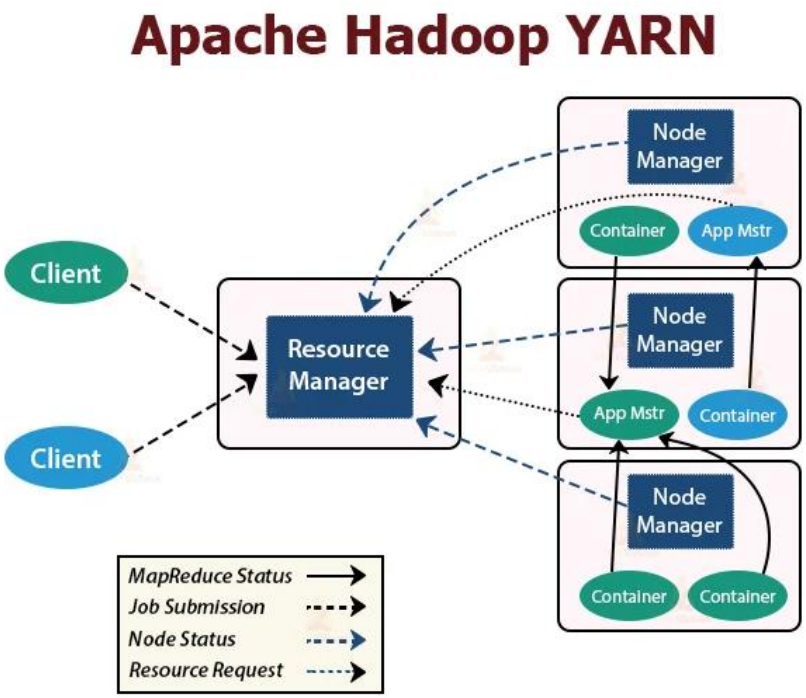
Scheduler: The pluggable scheduler of Resource Manager decides allocation of resources to various running applications. The scheduler is just that, a pure scheduler, meaning it does NOT monitor or track the status of the application.

Application Manager: It does:

- ✓ Accepting job submissions.
- ✓ Negotiating resources(container) for executing the application specific Application Master.
- ✓ Restarting the Application Master in case of failure.

Node Manager:For launching and monitoring the computer containers on machines in the cluster. Node Manager monitors the resource usage such as memory, CPU, disk, network, etc. It then reports the usage of resources to the global Resource Manager.

Map Reduce Application Master: Checks 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. It's responsibility is to negotiate required resources for execution from the Resource Manager. It works along with the Node Manager for executing and monitoring component tasks.



YARN Architecture

The steps involved in YARN architecture are:

1. The client program submits an application.

2. The Resource Manager launches the Application Master by assigning some container.

3. The Application Master registers with the Resource manager.

4. On successful container allocations, the application master launches the container by providing the container launch specification to the Node Manager.

5. The Node Manager executes the application code.

6. During the application execution, the client that submitted the job directly communicates with the Application Master to get status, progress updates.

7. Once the application has been processed completely, the application master deregisters with the Resource Manager and shuts down allowing its own container to be repurposed.

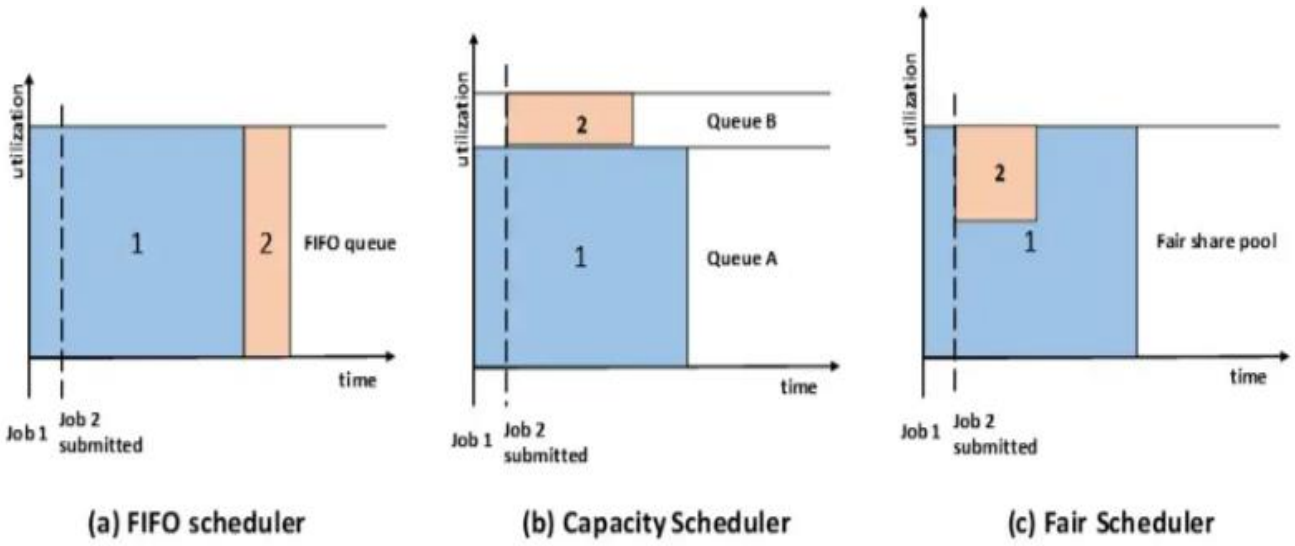
Scheduling in YARN:

YARN have 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.

A scheduler typically handles the resource allocation of the jobs submitted to YARN. Schedulers in YARN Resource Manager is a pure scheduler which is responsible for allocating resources to the various running applications.

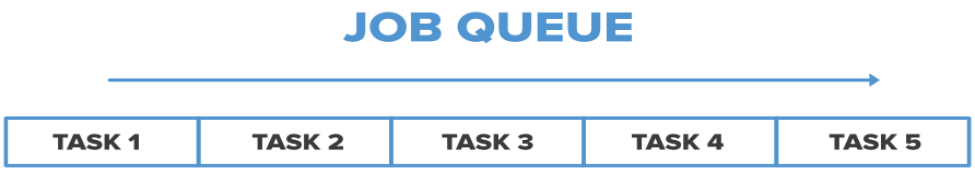
Example — if a computer app/service wants to run and needs 1GB of RAM and 2 processors for normal operation — it is the job of YARN scheduler to allocate resources to this application in accordance to a defined policy.

There are ***three types of schedulers*** available in YARN: 1. FIFO Scheduler, 2. Capacity Scheduler and 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 (First In First Out) Scheduler:** FIFO (first in, first out) is the simplest to understand and does not need any configuration. First In First Out is the default scheduling policy used in Hadoop. It runs the applications in submission order by placing them in a queue. Application submitted first, gets resources first and upon

completion, the scheduler serves next application in the queue. FIFO is not suited for shared clusters as large applications will occupy all resources and queues will get longer due to lower serving rate.

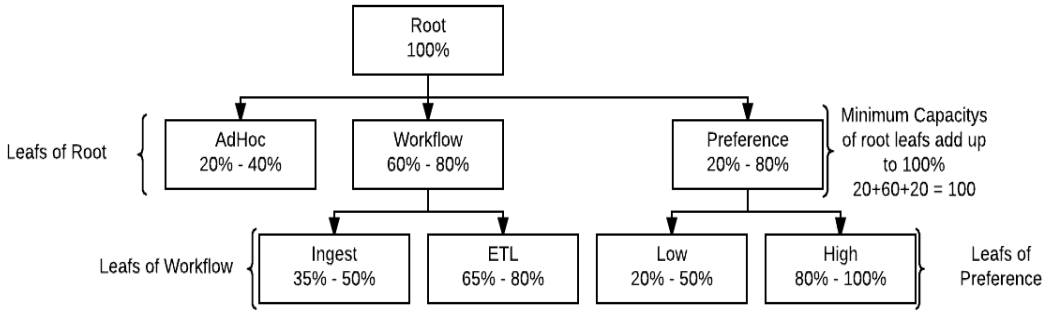
Advantages:

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

Disadvantages:

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

2. **Capacity Scheduler:**Capacity scheduler maintains a separate queue for small jobs in order to start them as soon a request initiates.However, this comes at a cost as we are dividing cluster capacity hence large jobs will take more time to complete. In Capacity Scheduler corresponding for each job queue, it provide some slots or cluster resources for performing job operation. Each job queue has it's own slots to perform its task. 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.



The fundamental idea of the Capacity Scheduler are around how queues are laid out and resources are allocated to them. Queues are laid out in a hierarchical design with the

topmost parent being the 'root' of the cluster queues, from here leaf (child) queues can be assigned from the root, or branches which can have leafs on themselves. Capacity is assigned to these queues as min and max percentages of the parent in the hierarchy. The minimum capacity is the amount of resources the queue should expect to have available to it if everything is running maxed out on the cluster. The maximum capacity is an elastic like capacity that allows queues to make use of resources which are not being used to fill minimum capacity demand in other queues.For example, with the Preference branch the Low leaf queue gets 20% of the Preference 20% minimum capacity while the High lead gets 80% of the 20% minimum capacity. Minimum Capacity always has to add up to 100% for all the leafs under a parent.

Advantages:

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

Disadvantages:

- 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. Fair scheduler does not have any requirement to reserve capacity. It dynamically balances the resources into all accepted jobs. When a job starts — if it is the only job running — it gets all the resources of the cluster. When the second job starts it gets the resources as soon as some containers, (a container is a fixed amount of RAM and CPU) get free. After the small job finishes, the scheduler assigns resources to large one. This eliminates both drawbacks as seen in FIFO and capacity scheduler i.e. overall effect is timely completion of small jobs with high cluster utilization.

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.