**Difference between FIFO scheduler and Capacity scheduler**

**FIFO scheduler**

The original scheduling algorithm that was integrated within the JobTracker was called *FIFO.* In FIFO scheduling, a JobTracker pulled jobs from a work queue, oldest job first. This schedule had no concept of the priority or size of the job, but the approach was simple to implement and efficient.

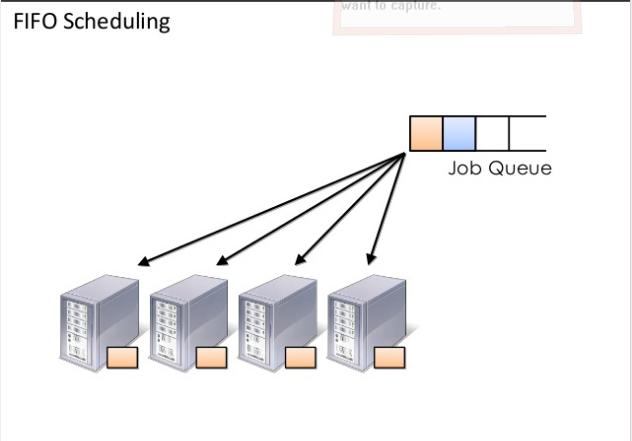
Hadoop’s Default scheduler

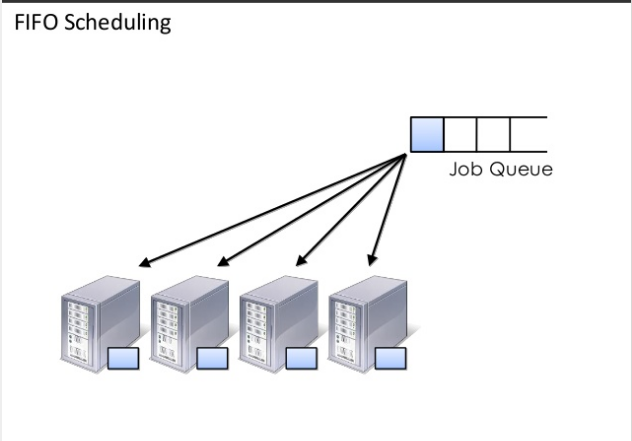
Problem :

1.Short job get struck behind long one .

2.Poor utilization.

3.Costly data replication



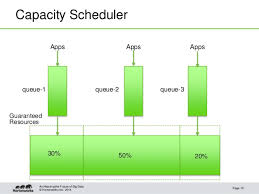


**Capacity scheduler**

In capacity scheduling, instead of pools, several queues are created, each with a configurable number of map and reduce slots. Each queue is also assigned a guaranteed capacity (where the overall capacity of the cluster is the sum of each queue's capacity).

Queues are monitored; if a queue is not consuming its allocated capacity, this excess capacity can be temporarily allocated to other queues. Given that queues can represent a person or larger organization, any available capacity is redistributed for use by other users.

It organise jobs into queues and shares as %’s of cluster. There will be FIFO scheduling within each queue and supports pre-emption. Queues are always monitored and if it exceeds the capacity, temporary queue will be allocated.



2. **Limitations of Hadoop 1.x**

1. Only one NameNode is possible to configure i.e If NameNode fails entire cluster goes down, that is why NameNode is called as Single Point of Failure (SPOF)
2. Secondary NameNode was just to take hourly backup of MetaData from NameNode.
   * Lets say SecondaryNameNode has taken backup at 10:00 AM, 10:45 NameNode fails then the transaction done during 10:00 to 10:45 is gone.
3. It is only suitable for Batch Processing of Huge amount of Data, which is already in Hadoop System.
4. It is not suitable for Real-time Data Processing.
5. It supports upto 4000 Nodes per Cluster.
6. It has a single component : JobTracker to perform many activities like Resource Management, Job Scheduling, Job Monitoring, Re-scheduling Jobs etc.
7. JobTracker is the single point of failure.
8. It supports only one Name Node and One Namespace per Cluster.
9. It does not support Horizontal Scalability of NameNode.
10. It runs only Map/Reduce jobs.
11. It is only suitable for Batch Processing of Huge amount of Data, which is already in Hadoop System.
12. It follows Slots concept in HDFS to allocate Resources (Memory, RAM, CPU). It has static Map and Reduce Slots. That means once it assigns resources to Map/Reduce jobs, it cannot re-use them even though some slots are idle.

For Example:- Suppose, 10 Map and 10 Reduce Jobs are running with 10 + 10 Slots to perform a computation. All Map Jobs are doing their tasks but all Reduce jobs are idle. We cannot use these Idle jobs for other purpose.

**Overcome of Hadoop 2.x**

If we observe the components of Hadoop 1.x and 2.x, Hadoop 2.x Architecture has one extra and new component that is : YARN (Yet Another Resource Negotiator).

* Hadoop 1.x supports only one namespace for managing HDFS filesystem whereas Hadoop 2.x supports multiple namespaces.
* Hadoop 1.x supports one and only one programming model: MapReduce. Hadoop 2.x supports multiple programming models with YARN Component like MapReduce, Interative, Streaming, Graph, Spark, Storm etc.
* Hadoop 1.x has lot of limitations in Scalability. Hadoop 2.x has overcome that limitation with new architecture.
* Hadoop 2.x has Multi-tenancy Support, but Hadoop 1.x doesn’t.
* Hadoop 1.x HDFS uses fixed-size Slots mechanism for storage purpose whereas Hadoop 2.x uses variable-sized Containers.
* Hadoop 1.x supports maximum 4,000 nodes per cluster where Hadoop 2.x supports more than 10,000 nodes per cluster.

**Hadoop 2.x has resolved most of the Hadoop 1.x limitations by using new architecture.**

* By decoupling MapReduce component responsibilities into different components.
* By Introducing new YARN component for Resource management.
* By decoupling component’s responsibilities, it supports multiple namespace, Multi-tenancy, Higher Availability and Higher Scalability.

**Hadoop 2.x YARN has the following benefits**.

* Highly Scalability
* Highly Availability
* Supports Multiple Programming Models
* Supports Multi-Tenancy
* Supports Multiple Namespaces
* Improved Cluster Utilization
* Supports Horizontal Scalability