Resource Allocation using YARN and MESOS

To run multiple frameworks in a single cluster, we need to maximise resource utilisation and allow data sharing between frameworks in the cluster.

An implementation of dynamic sharing is feasible than sequential static sharing.

YARN

YARN, or YARN Application Resource Negotiator or Yet Another Resource Negotiator, is a cluster resource management framework running on top of HDFS.

Its main components are

1. Resource Manager
    a. Arbitrates resources amongst all applications of the system
    b. Keeps track of live node managers and available resources
2. Node Manager
    a. Responsible for launching application containers
    b. Monitors resource usage
3. Application Master
    a. Negotiates appropriate resource containers from the scheduler
    b. Tracks and monitors the progress of containers
4. Container
    a. Unit of allocation incorporating resources

Job Scheduling is done by using a default Capacity Scheduler or a FIFO Scheduler.

The Fair Scheduler

1. Aims to give every user fair share of cluster capacity over time
2. If a single job is running, it gets the whole cluster
3. As more jobs are submitted, free slots given to jobs such that fair share of the cluster is maintained
4. Does not cause starvation as both long and short jobs keep making progress
5. Jobs are placed in pools and each user gets their own pool
6. Two users get the same amount of cluster resources on average
7. Custom pools can be defined for specific users
8. Supports preemption, so if a pool is not receiving a fair share, the scheduler kills tasks in the pools running over capacity to give the slots to the pool running under capacity

The Capacity Scheduler

1. Cluster made of a number of queues which may be hierarchical with their own allocated capacity
2. Each job in a queue scheduled using FIFO with priorities
3. Allows separate MR cluster with FIFO for each user, removing fair sharing of a pool's resources

Tasks failures can occur as

1. Runtime exceptions
    a. Error is logged and task is marked as failed
2. Hanging tasks
    a. Timeouts set
3. Killed Tasks
    a. Speculative duplicates can be killed
    b. When progress is slow, a speculative task is created and the slower of the two is killed

MESOS

Mesos is a common resource sharing layer over which diverse frameworks can run.

The main goals of Mesos are

1. High utilisation of resources
2. Support diverse frameworks
3. Scalability
4. Reliability

The resulting design is a small microkernel-like core that pushes scheduling logic to frameworks.

The main design elements are:

1. Fine grained sharing
    a. Allocation at the level of tasks within a job
    b. Improves utilisation, latency and locality
2. Resource offers
    a. Simple, scalable, application controlled scheduling

A way to implement resource offers would be global scheduling, where frameworks request resources in a specification language, and the scheduler matches them to resources.

This can make optimal decisions, but the drawbacks are

1. Complex language to be developed
2. Difficult to scale and make robust
3. Future frameworks may have unanticipated needs

The way Mesos does it is, it offers all the available resources to frameworks, allowing them to pick which ones they need. This makes Mesos simple, but optimality is lost as the framework is decentralised.

YARN vs MESOS

Yarn flow vs Mesos Flow

1. App manager makes request to YARN and YARN delivers resources. Mesos scheduler makes resource offer to app scheduler and the App scheduler accepts or rejects it.
2. YARN decides where to run the job, which may be more optimal for cluster utilisation. In Mesos, the App scheduler makes the decision on where to run the job based on criteria.