Resource Request

- Request for container - amount of computer res. and locality constraint
- locality is critical - to use cluster bandwidth efficiently
- if locality cannot be met - YARN start container on node in same rack or any node in cluster
- YARN application can make res. req.
  - ↳ all request up front (Spark)
  - ↳ Dynamic approach (MR - Map → Reduce)

Application Lifespan

- Can be short lived app (few sec) or long running app (runs for days or months)
- Categorize app - how they map to the job user runs.

↳ 1) one application per use job (MR)

2) one app per workflow or user session (unrelated jobs)
   - ↳ more efficient | container reused b/w jobs | Cache intermediate data b/w jobs (Spark)

3) long running app shared by diff users
   - ↳ coordination role
   - ↳ Apache Slider - has AM for launching other app on cluster. (Impala)

YARN Applications Intersted in

→ Running (DAG) Job - Spark | Tez
→ Stream Processing - Spark Streaming, Samza | Storm
→ Simplify process of building YARN - Apache Slider - Possible to run existing distapp on YARN - Different user can run diff version of same app (HBase)

	MapReduce 1	YARN
1-	JobTracker	- RM, AM, timeline Server
2-	TaskTracker	- Node Mgr
3-	Slot	- Container

timeline server - store application history

4-	Batch Processing	- Real time, Interactive
5-	1 nameSpace	- Multiple nameSpace
6-	Only MR app	- MapReduce and non MR app
7-	Average Cores optimization (Fixed MR Slot)	- Excellent cluster res. optimization (Central)

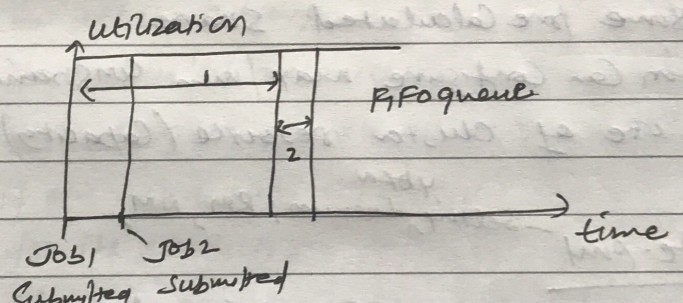
YARN Scheduling

Date: Page: 27

- Scheduler is a Component of RM
- Responsible for allocating resources to app.
- Scheduler
  - perform no monitoring or tracking status of App
  - offer no guarantees about restarting failed task

1) FIFO Scheduler

- First in First out
- Jobs run in the order of submission
- Job submitted first will get priority to execute
- queue based scheduler
- Simple, doesnot guarantee performance efficiency

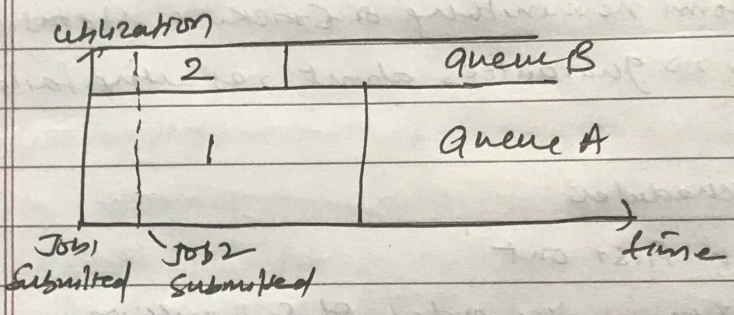


- Simple to understand - not needing any configuration
- not suitable for shared cluster.
- large app will use all cluster resources - Small one to wait

2) Capacity Scheduler

- Allow app to share resource.
- Job queues
- Seperate queue - allow small jobs to start as soon as it is submitted

- As Capacity queue is reserved for job - at the cost of overall cluster utilization
- large job finishes later than FIFO



- allow user to share resources - User or group of user assigned a certain capacity of cluster.
- To enable - admin configs one or more queue with some pre calculated share
- Admin can configure max/min constraint on the use of cluster resource (Capacity) on each queue

- setting YARN daemons: RM, NM,

yarn-site.xml

name - yarn.resourcemanager.scheduler.class
value - org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler (Capacity)
/fifo.FifoScheduler (FIFO)
/fair.FairScheduler (Fair)

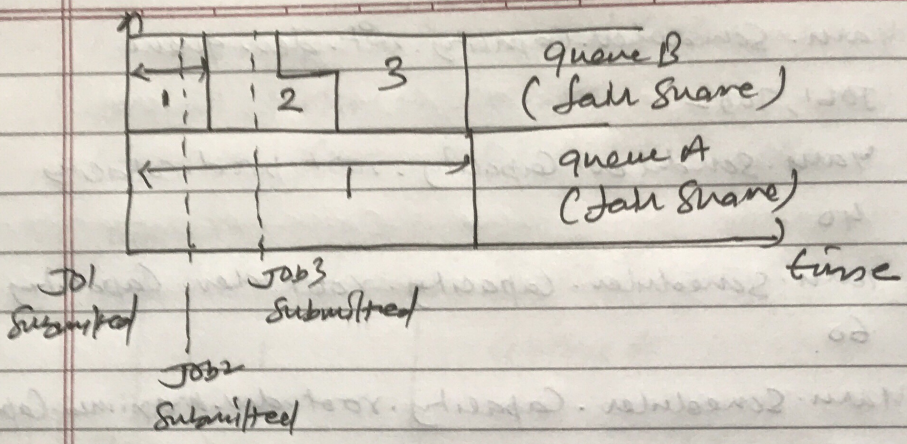
Capacity-scheduler.xml

name - yarn.scheduler.capacity.root.queues
value - prod, dev

name - Yarn.Scheduler.Capacity.root.dev.queue
value - Job1, Job2
name - Yarn.Scheduler.Capacity.root.prod.Capacity
value - 40
name - Yarn.Scheduler.Capacity.root.dev.Capacity
value - 60
name - Yarn.Scheduler.Capacity.root.dev.Maximum-Capacity
value - 75

Fair Scheduling

- Enable memory intensive app to share cluster resource in efficient way.
- Allocation of resources such that all applications get, on average an equal share of resource
- if one app is running - might request all resource for its execution if needed.
- if other app are submitted policy distribute free resource among app such that each app get a fairly equal share of resource.
- allow preemption - RM request container back from AM depends of Job Configuration
- Adv - Every queue get min share of cluster res.
- App waiting in queue - get min resource share
- Resources are more than for app - excess amount would be distributed.


utilization
1
2
3
queue B
(fair share)
queue A
(fair share)
time
Job1 Submitted
Job3 Submitted
Job2 Submitted