Big Data
- Job Scheduling

Dr. Qing “Matt” Zhang
ITU

Hadoop Job Scheduling

- A Hadoop job is composed of
  - An unordered set of Map tasks which have locality preferences
  - An unordered set of Reduce tasks
- Tasks are scheduled by the JobTracker
  - They are then scheduled/launched by TaskTrackers
  - One TaskTracker per node

Task Tracker

- Each TaskTracker has a fixed number of slots for Map and Reduce tasks
  - This may differ per node – a node with a powerful processor may have more slots than one with a slower CPU
- TaskTrackers report the availability of free task slots to the JobTracker on the Master node
- Scheduling a job requires assigning Map and Reduce tasks to available Map and Reduce task slots

Hadoop Job Scheduler

- The FIFO scheduler
- The Fair scheduler
- The Capacity scheduler

FIFO Scheduler

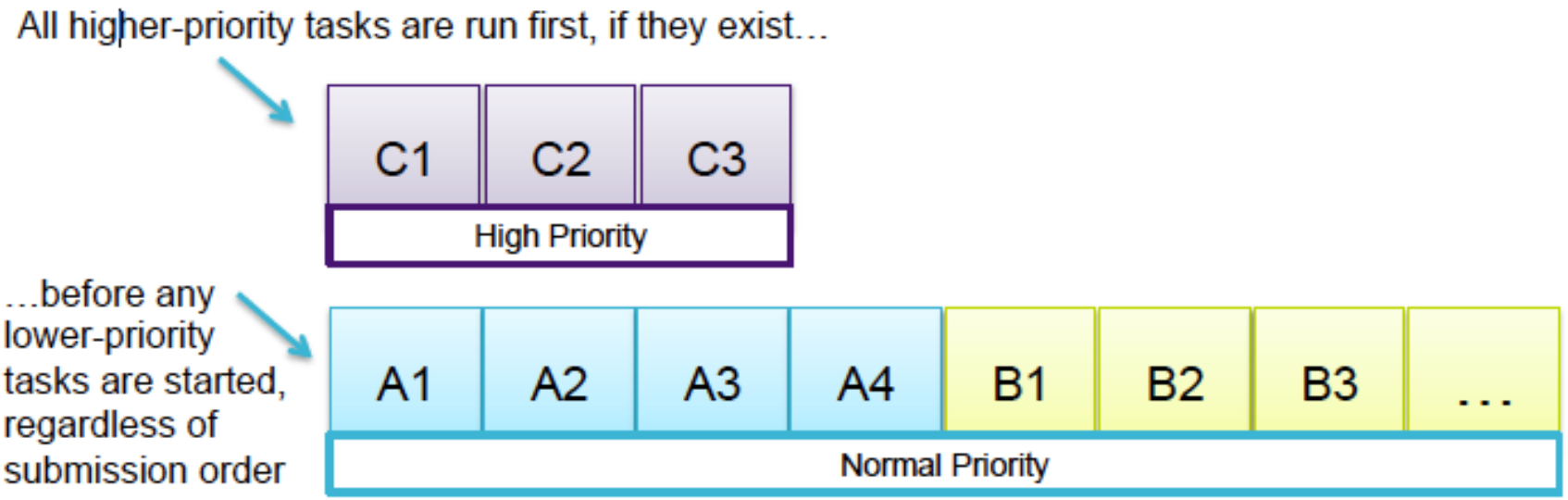
- It's the default Hadoop job scheduler
  - First In, First Out
- Given two jobs A and B, submitted in that order, all Map tasks from job A are scheduled before any Map tasks from job B are considered
  - Similarly for Reduce tasks
- Order of task execution within a job may be shuffled around

FIFO with Priorities

- The FIFO Scheduler supports assigning priorities to jobs
  - Priorities are *VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW*
  - Set with the mapred.job.priority property
  - May be changed from the command line as the job is running

    hadoop job -set-priority <job_id> <priority>
- All work in each queue is processed before moving on to the next

FIFO with Priorities



Problems

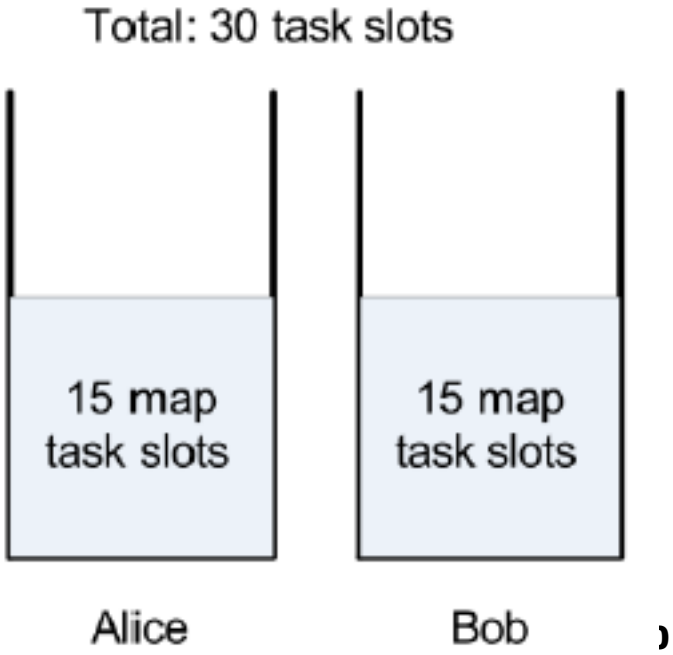
- Job A may have 2,000 tasks; Job B may have 20
  - Even if Job B has a higher priority than Job A, Job B will not make any progress until Job A has nearly finished
  - Completion time should be proportional to job size
- Users with poor understanding of the system may always flag all their jobs as HIGH_PRIORITY
  - Starving other jobs of processing time
- ‘All or nothing’ nature of the scheduler makes sharing a cluster between production jobs with SLAs and interactive users challenging

Fair Scheduler

- Fair Scheduler is designed to allow multiple users to share the cluster simultaneously
  - Should allow short interactive jobs to coexist with long production jobs
  - Should allow resources to be controlled proportionally
  - Should ensure that the cluster is efficiently utilized

Fair Scheduler: Concept

- Each job is assigned to a pool
  - Default assignment is one pool per username
- Jobs may be assigned to arbitrarily named pools
  - Such as production
- Physical slots are not bound to any specific pool
- Each pool gets an even share of the available task slots

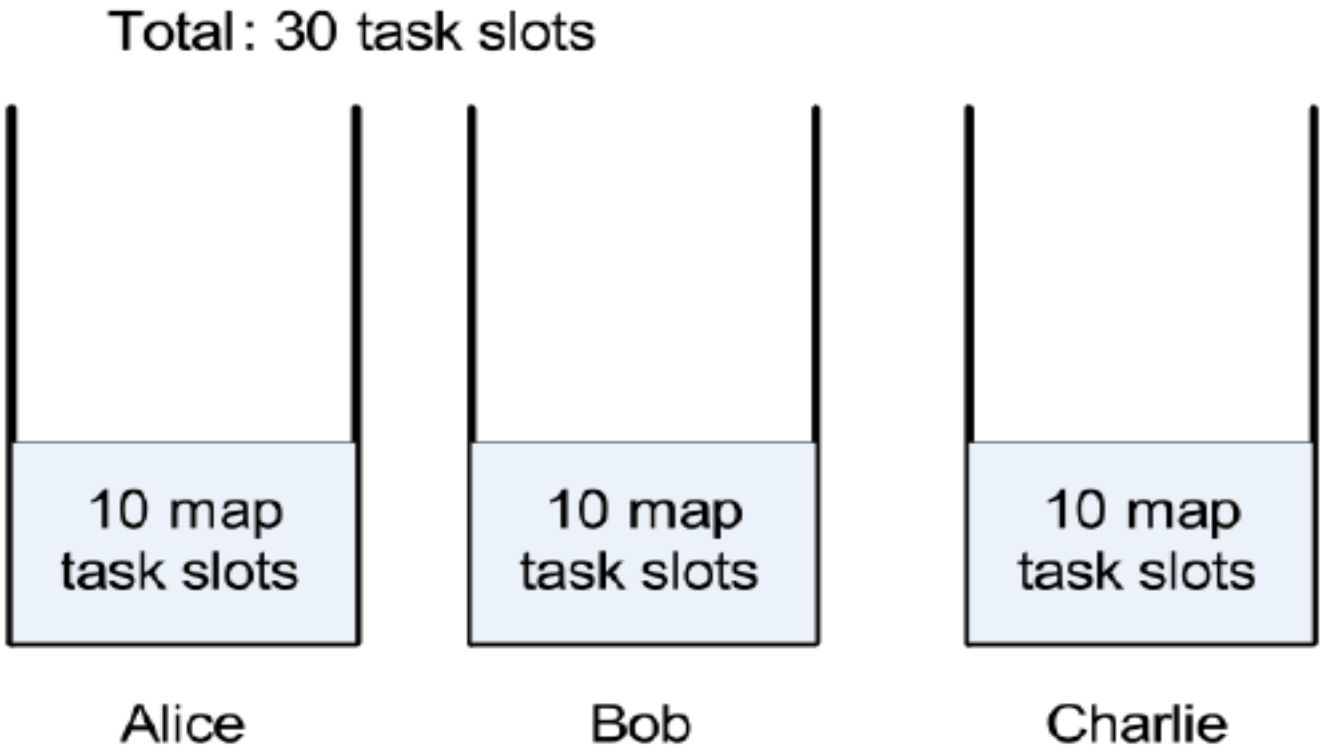


Pool Creation

- By default, pools are created dynamically based on the username submitting the job
  - No configuration necessary
- Jobs can be sent to designated pools (e.g., “production”)
- Pools can be defined in a configuration file
- Pools may have a minimum number of mappers and reducers defined

Dynamic adjustment

- If a new job is submitted to a new pool, slots for pools will be adjusted.



Determine the Fair Share

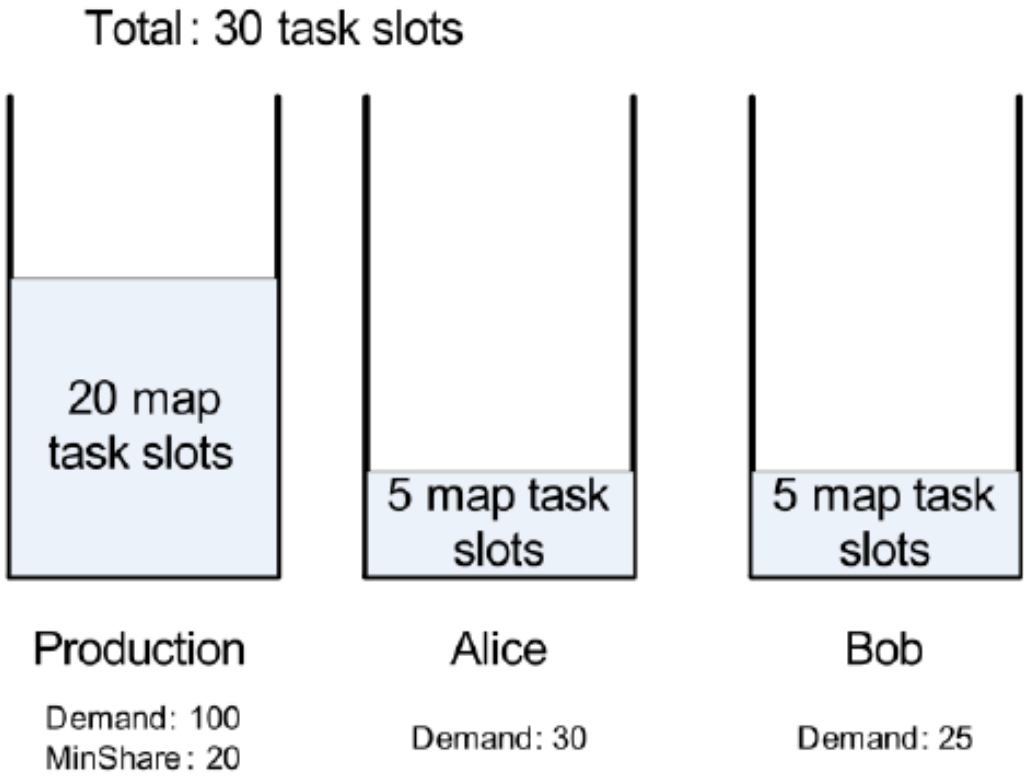
- The fair share of tasks slots assigned to the pool is based on:
  - The actual number of task slots available across the cluster
  - The demand from the pool
    - The number of tasks eligible to run
  - The minimum share, if any, configured for the pool
  - The fair share currently assigned to each of other active pools

Fair share for a pool

- The fair share for a pool will never be higher than the actual demand
- Pools are first filled up to their minimum share, assuming cluster capacity
- Excess cluster capacity is spread across all pools
  - Try to maintain the most even loading possible

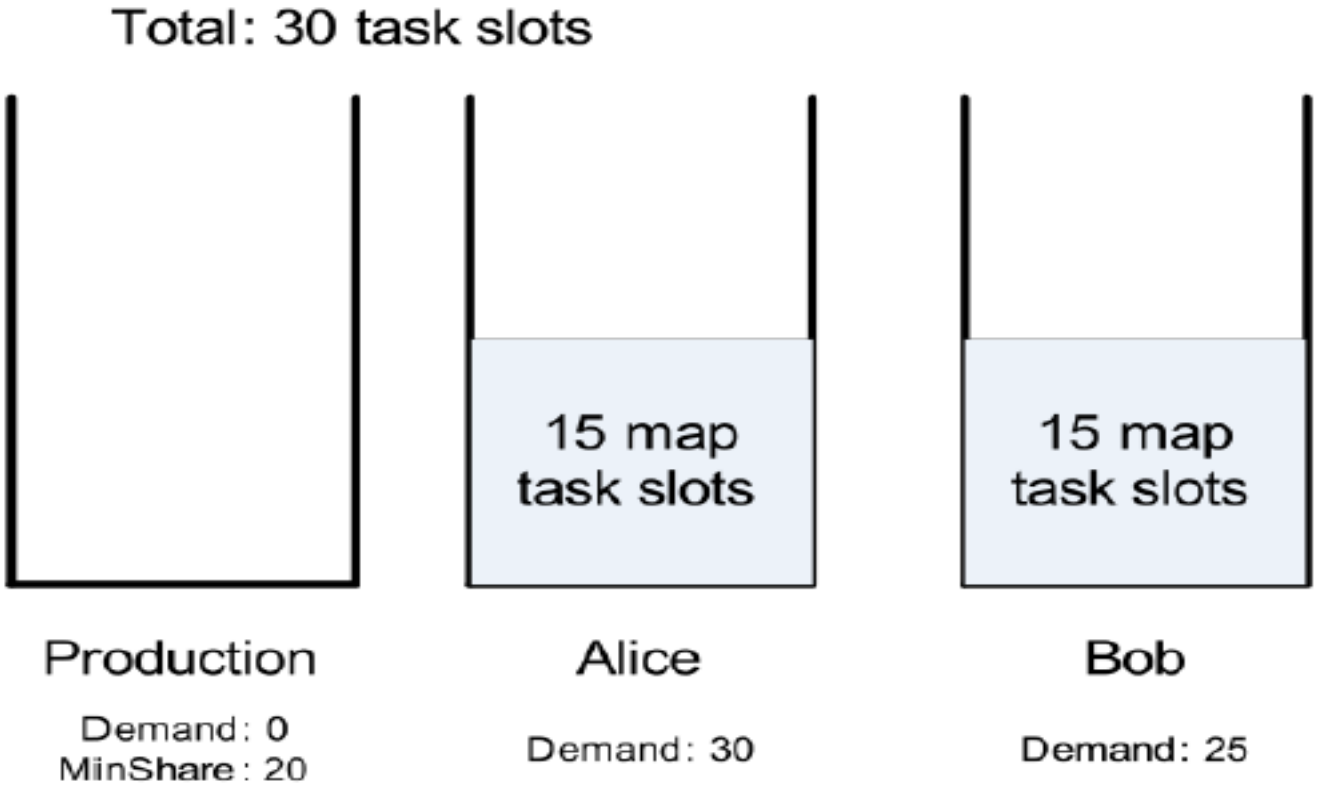
Example Share Allocation

- Each pool has a Demand and MinShare requirement
- First, fill Production up to 20 slot minimum guarantee
- Then distribute remaining 10 slots evenly across Alice and Bob



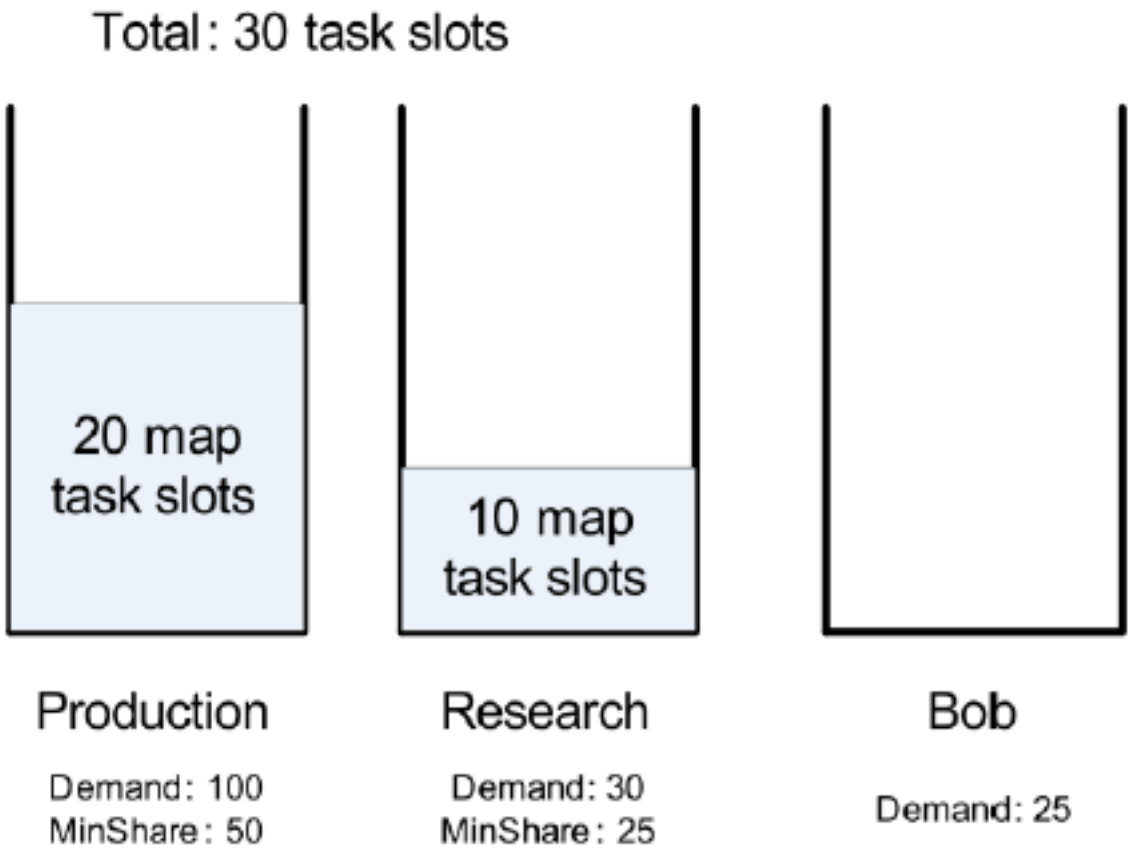
Example 2

- Production has no demand, so no slots reserved
  - Demand can be seen as an upper bound for allocation
- All slots are allocated evenly across Alice and Bob



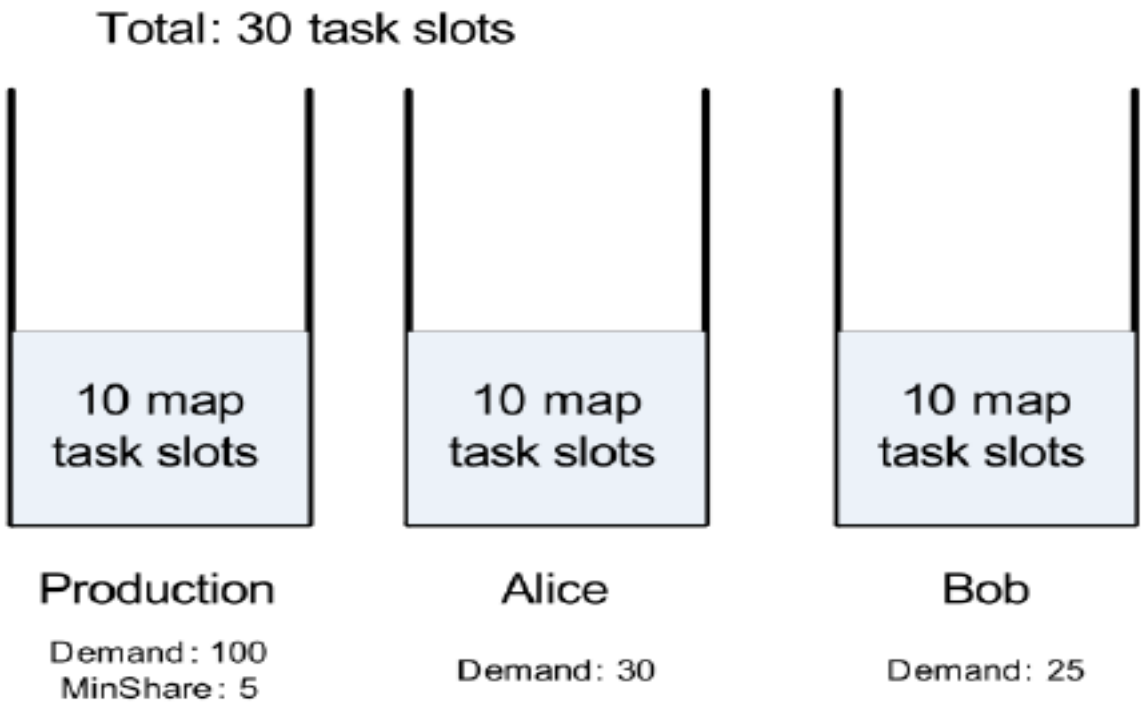
Example 3

- minShare of Production, Research exceeds available capacity
- minShares are scaled down (2:1) to match actual slots
- No slots remain for users without minShare (i.e., Bob)



Example 4:

- Production filled to minShare
- Remaining 25 slots distributed across all pools
  - First to make all pools even, then assign evenly the remaining pool
- Production pool gets more than minShare, to maintain fairness

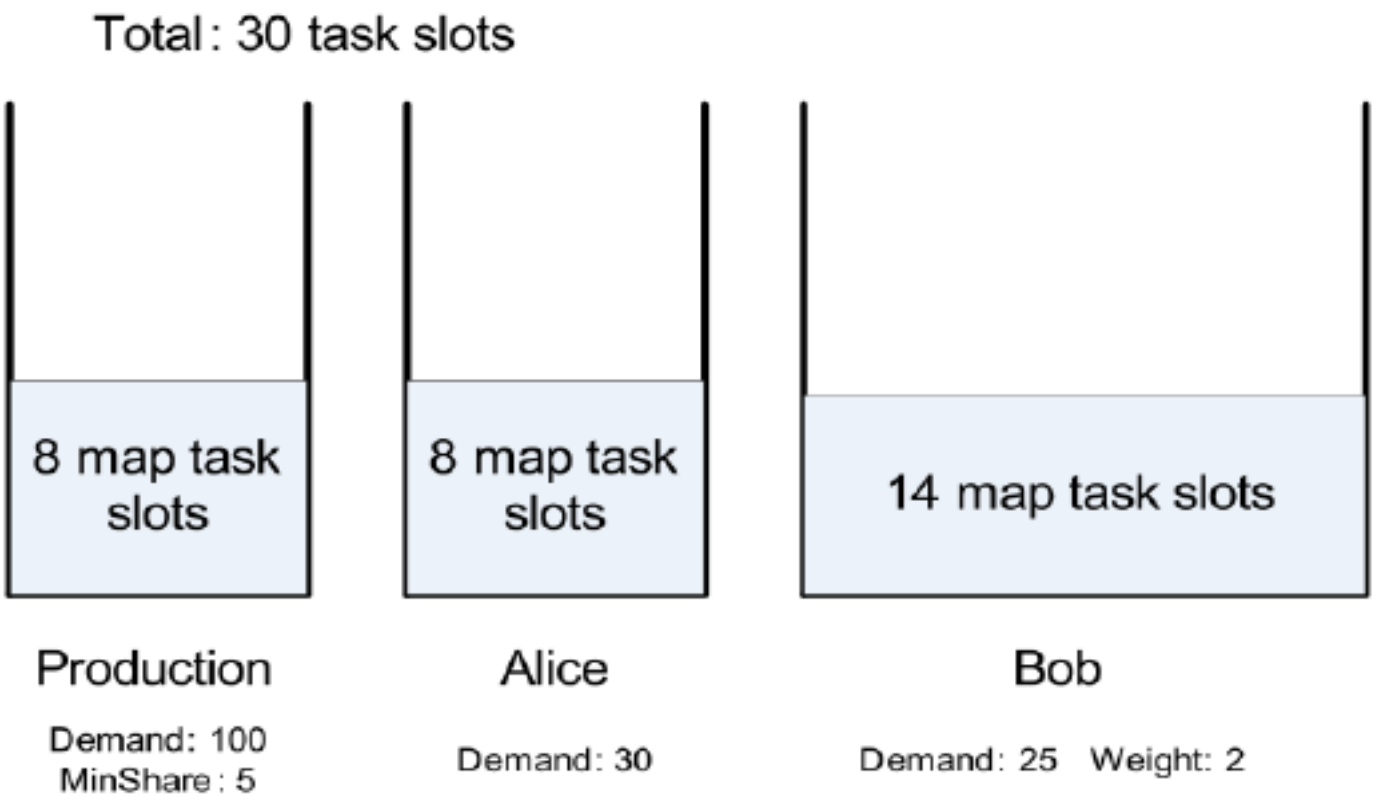


Pools with Weights

- In addition to setting minShare, pools can be assigned a weight
- Pools with higher weight get more slots during free slot allocation
- ‘Even water glass height’ analogy:
  - Think of the weight as controlling the ‘width’ of the glass

Example

- Bob has double weight
- Production filled to minShare
- Remaining 25 slots distributed across pools
- Bob's pool gets two slots instead of one during each round



Detailed computation

Production	Alice	Bob	total
Round 1: Production filled to minShare			
5	0	0	5
Round 2: Make all pools even			
5	5	10	20

Round 3: Assign remaining ones

10/4 =2.5, round up and assign from 1st pool, go through each of them, and put remaining to last pool.

Summary

- Demand is the max cap of resource assigned
- minShare is assigned first
- If minShare cannot be satisfied, assign according to ratio
- Final goal is to achieve ‘Even water glass height’
  - With weight (width)

Multiple Jobs Within A Pool

- A pool exists if it has one or more jobs in it
- So far, we've only described how slots are assigned to pools
- We need to determine how jobs are scheduled within a given pool

Job Scheduling Within a Pool

- Within a pool, resources are fair scheduled across all jobs
  - This is achieved via another instance of Fair Scheduler
- It is possible to enforce FIFO scheduling within a pool
  - May be appropriate for jobs that would compete for external bandwidth, for example

Job Scheduling Within a Pool

- Pools can have a maximum number of concurrent jobs configured
- The weight of a job within a pool is determined by its priority (NORMAL, HIGH etc)
- Assignment of a slot to a pool can be delayed
  - The Fair Scheduler lets free slots on a host remain open for a short time if no queued tasks prefer to run on that host
  - This increases the overall data locality hit ratio

Preemption in Fair Scheduler

- If shares are imbalanced, pools which are over their fair share may not assign new tasks when their old ones complete
  - Eventually, as tasks complete, free slots will become available
  - Those free slots will be used by pools which were under their fair share
  - This may not be acceptable in a production environment, where tasks take a long time to complete
- Two types of preemption are supported
  - minShare preemption
  - Fair Share preemption

minShare Preemption

- Pools with a minimum share configured are operating on an SLA (Service Level Agreement)
  - Waiting for tasks from other pools to finish may not be appropriate
- Pools which are below their minimum guaranteed share can kill the newest tasks from other pools to reap slots
  - Can then use those slots for their own tasks
  - Ensures that the minimum share will be delivered within a timeout window

Fair Share Preemption

- Pools not receiving their fair share can kill tasks from other pools
  - A pool will kill the newest task(s) in an over-share pool to forcibly make room for starved pools
- Fair share preemption is used conservatively
  - A pool must be operating at less than 50% of its fair share for 10 minutes before it can preempt tasks from other pools

The Capacity Scheduler

- Each job is assigned to a queue
  - Analogous to a FIFO Scheduler pool
- Queues are assigned a percentage of the cluster's slots
  - Similar to a Fair Scheduler pool's minShare
- A queue's unused capacity is not given away if it is not being used
  - Unlike the Fair Scheduler, in which pools with lower usage can give away their slots
- Jobs within queues are FIFO ordered
  - Similar to the FIFO Scheduler, you can enable prioritization within queues
- Slot allocation can be based on tasks' memory usage

Comparison

Requirement	FIFO Scheduler	Fair Scheduler	Capacity Scheduler
Learning tool or proof of concept	✔		
Pool utilization varies, so it is desirable that pools give away resources when they are not in use		✔	
Jobs within a pool need to make equal progress		✔	
Data locality makes a significant difference in job run-time performance		✔	
Pool utilization has little fluctuation			✔
Jobs have a high degree of variance in memory utilization			✔

Comparison

- In practice, the Fair Scheduler is far more widely used than the Capacity Scheduler
- Fair Scheduler assigns resources to jobs such that all jobs get, on average, an equal share of resources over time.
- Capacity Scheduler gives each organization a minimum capacity guarantee.
  - Available resources in the cluster are partitioned among multiple organizations who collectively fund the cluster based on computing needs.

Configure the Fair Scheduler

- Steps:
  - Enable the Fair Scheduler
  - Configure Scheduler parameters
  - Configure pools

Enable the Fair Scheduler

- In *$HADOOP_PREFIX/etc/hadoop/mapred-site.xml* on the JobTracker, specify the scheduler:

```
<property>
  <name>mapred.jobtracker.taskScheduler</name>
  <value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
```

- Identify the pool configuration file:

```
<property>
  <name>mapred.fairscheduler.allocation.file</name>
  <value>/etc/hadoop/conf/allocations.xml</value>
</property>
```

Configure Scheduler parameters

- In *$HADOOP_PREFIX/etc/hadoop/mapred-site.xml*

mapred.fairscheduler.poolnameproperty	Specifies which job configuration property is used to determine the pool that a job belongs in. Default is `user.name` (i.e., one pool per user). Other options include `group.name`, `mapred.job.queue.name`
mapred.fairscheduler.sizebasedweight	Makes a pool's weight proportional to log(demand) of the pool. Default: false.
mapred.fairscheduler.weightadjuster	Specifies a WeightAdjuster implementation that tunes job weights dynamically. Default is blank; can be set to `org.apache.hadoop.mapred.NewJobWeightBooster`.
mapred.fairscheduler.preemption	Enables preemption in the Fair Scheduler. Set to `true` if you have pools that must operate on an SLA. Default is `false`.

Configure pools

- The allocations configuration file must exist, and contain an <allocations> entity
- <pool> enties can contain *minMaps, minReduces, maxMaps, maxReduces, maxRunningJobs, weight, minSharePreemptionTimeout, schedulingMode*
- <user> enties (optional) can contain maxRunningJobs
  - Limits the number of simultaneous jobs a user can run
- <userMaxJobsDefault> entity (optional)
  - Maximum number of jobs for any user without a specified limit
- System wide and per-pool timeouts can be set

Example: basic format

- The allocations configuration file must exist, and contain at least this:

```
<?xml version="1.0"?>
<allocations>
</allocations>
```

Example: Limit Users to 3 Jobs

- Limit max jobs for any user: specify userMaxJobsDefault

```
<?xml version="1.0"?>
<allocations>
  <userMaxJobsDefault>3</userMaxJobsDefault>
</allocations>
```

Example: Allow One User More Jobs

- If a user needs more than the standard maximum number of jobs, create a <user> entity

```
<?xml version="1.0"?>
<allocations>
  <userMaxJobsDefault>3</userMaxJobsDefault>
  <user name="bob">
    <maxRunningJobs>6</maxRunningJobs>
  </user>
</allocations>
```

Example: Adding Fair Share Timeout

- Set a Preemption timeout to 300 seconds
  - If a job is below half its fair share for 5 minutes, it will be allowed to kill tasks from other jobs to achieve its share.

```
<?xml version="1.0"?>
<allocations>
  <userMaxJobsDefault>3</userMaxJobsDefault>
  <user name="bob">
    <maxRunningJobs>6</maxRunningJobs>
  </user>
<fairSharePreemptionTimeout>300</fairSharePreemptionTimeout>
</allocations>
```

Example: Create a specific Pool

- Pools are created by adding <pool> entities

```
<?xml version="1.0"?>
<allocations>
  <userMaxJobsDefault>3</userMaxJobsDefault>
  <pool name="production">
    <minMaps>20</minMaps>
    <minReduces>5</minReduces>
    <weight>2.0</weight>
  </pool>
</allocations>
```

Example: Add an SLA to the Pool

- Configure the minShare for the SLA

```
<?xml version="1.0"?>
<allocations>
  <userMaxJobsDefault>3</userMaxJobsDefault>
  <pool name="production">
    <minMaps>20</minMaps>
    <minReduces>5</minReduces>
    <weight>2.0</weight>
    <minSharePreemptionTimeout>60</minSharePreemptionTimeout>
  </pool>
</allocations>
```

Example: Create a FIFO Pool

- FIFO pools are useful for jobs which are bandwidth intensive

```
<?xml version="1.0"?>
<allocations>
  <pool name="bandwidth_intensive">
    <minMaps>10</minMaps>
    <minReduces>5</minReduces>
    <schedulingMode>FIFO</schedulingMode>
  </pool>
</allocations>
```

- To use the Fair Scheduler, use FAIR for schedulingMode

Scheduler web UI

- The Fair Scheduler exposes a status page in the JobTracker Web user interface at *http://<job_tracker_host>:50030/scheduler*
  - Allows you to inspect pools and allocations
  - In YARN, it's moved to the ResourceManager web UI at:

  http://<ResourceManager Host>:8088/cluster/scheduler
- Any changes to the pool configuration file (e.g., allocations.xml) will automatically be reloaded by the running scheduler
- Scheduler detects a timestamp change on the file
  - Waits five seconds aper the change was detected, then reloads the file
  - If the scheduler cannot parse the XML in the configuration file, it will log a warning and continue to use the previous configuration