Hadoop Administration

Managing & Scheduling Jobs, Cluster Maintenance & Logging

MapReduce Schedulers

- Scheduler is responsible for assigning tasks to open slots on tasktrackers
- Maps tasks have a locality preference
- Options: FIFO, Fair, Capacity

1) MR job sent to JT
2) List of input splits computed
3) JT creates a M task for each split
4) Each M task is designated to a queue
5) Scheduler plugin decides which tasks from which queue to process first

FIFO Scheduler

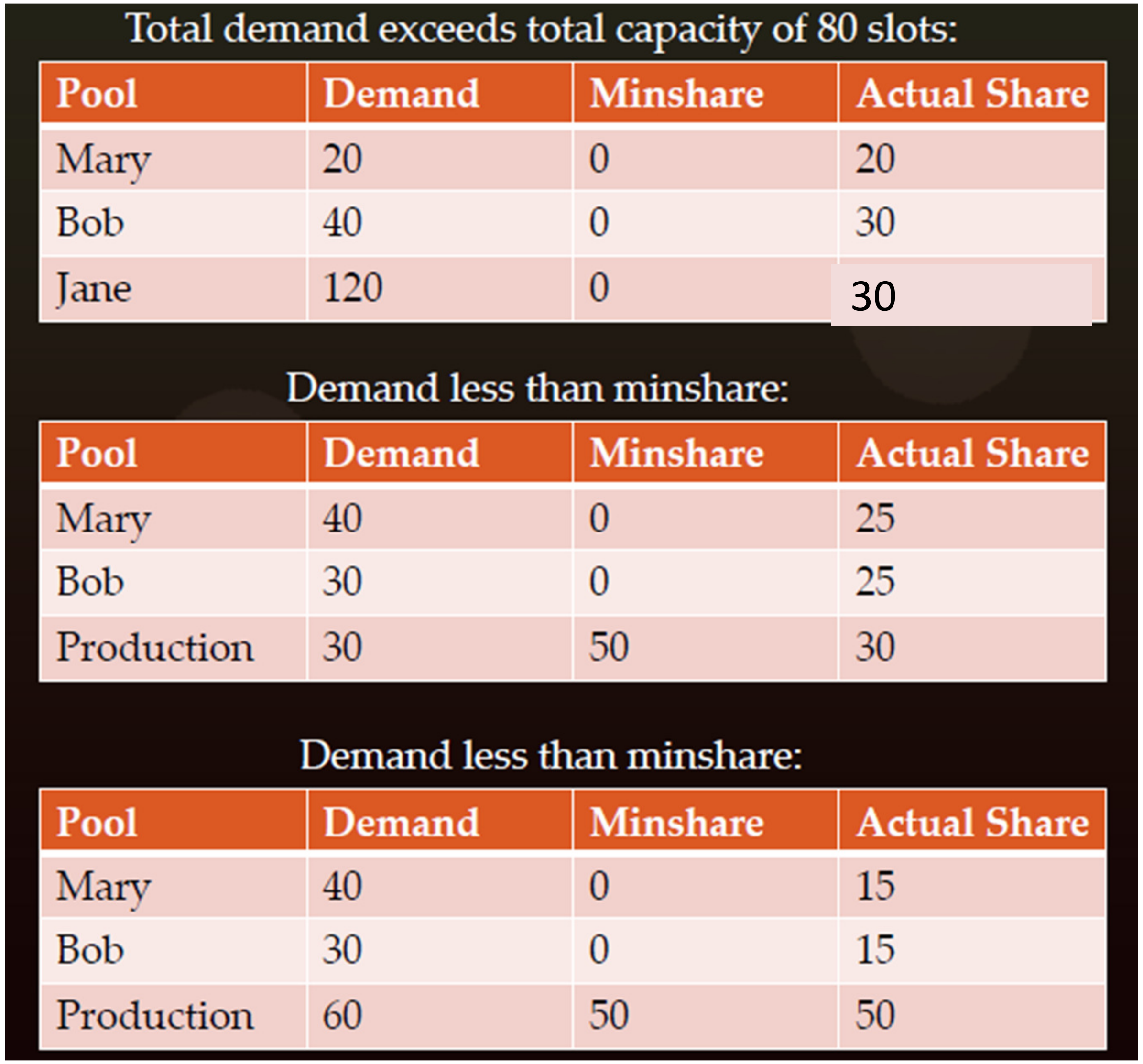
- Default scheduler, good for bootstrapping small, prototype clusters
- All tasks in job A will run before tasks from job B
- 5 levels of job prioritization: very low, low, normal, high, very high
- Each priority is a separate FIFO queue

- **Example:**
  - Data Scientist wants to do log analysis on 30 TB of data
  - This will be 245,760 map tasks (10TB / 128MB)
  - A 400 node cluster with 10 Map slots per node allows for 4000 map tasks to be processes in parallel
  - Each Map JVM runs for about 30 seconds
  - 245,000 maps/ 4,000 maps per run = 61 turns
  - 61 * 30 seconds = ~ 30 minutes

Fair Scheduler

- Jobs are assigned to pools
- Each pool is assigned a # of task slots, after considering total slot capacity, current demand, minimum slot guarantees and available slot capacity
- Pools have an SLA
- Beyond minimum slot guarantees, each pool gets an equal # of remaining available slots in the cluster (fair share)
- A MR job property defines how the scheduler determines which pool a job will be tagged to
- **Terminology:** Total capacity, Total available capacity, Pool, Demand, Fair share, minimum share

Fair Scheduler



Total demand exceeds total capacity of 80 slots:

Pool	Demand	Minshare	Actual Share
Mary	20	0	20
Bob	40	0	30
Jane	120	0	30

Demand less than minshare:

Pool	Demand	Minshare	Actual Share
Mary	40	0	25
Bob	30	0	25
Production	30	50	30

Demand less than minshare:

Pool	Demand	Minshare	Actual Share
Mary	40	0	15
Bob	30	0	15
Production	60	50	50

Fair Scheduler

- If a SLA specified job is submitted to a queue with min share, and those slots have already been allocated to another queue, the scheduler will kill the busy slots for other jobs and steal them back for the SLA job
  - **Minimum share preemption:** occurs when a pool is operating below its configured minimum share; happens aggressively
  - **Fair share preemption:** occurs when a pool is operating below its fair share; pool must be below half of its fair share for the preemption to kick in

Capacity Scheduler

- Different philosophy toward task scheduling algorithms and some feature differences
- Simpler and more deterministic than Fair Scheduler.
- Fair Scheduler works exclusively with slots; Cap Scheduler understands memory consumption of a job's tasks as well.
- Administrator configures one or more queues, each with a capacity (like FS's min share, however it is always reserved for this queue and not given away)
- Any additional slots beyond the sum of the queue capacities may be freely assigned to any queue as needed
- There is no preemption/killing
- Within a queue, jobs for the same user are FIFO ordered, but jobs can be prioritized within a queue

Cluster Maintenance

Starting and stopping a process

- *You may need to restart a process to enact config changes or after adding/removing worker nodes*

1) Sudo to root
2) Run: /etc/init.d/*script operation (operation can be start, stop, restart)*
3) Confirm the process restarted with: ps –ef | grep *process*

HDFS: Adding a DataNode

- *You may need to restart a process to enact config changes or after adding/removing worker nodes*

1) Add IP of DN to the include file specified by dfs.hosts
2) Run: dfsadmin -refreshNodes
3) If needed, update the rack awareness script
4) Start the datanode process
5) Verify with NN web UI or hadoop dfsadmin -report

HDFS: Decommission a DataNode

- *You may need to restart a process to enact config changes or after adding/removing worker nodes*

1) Shut down NameNode
2) Create empty exclude file in NameNode's local file system
3) Set the dfs.hosts.exclude parameter to the exclude file
4) Restart NameNode
5) Add the node you want to remove's full hostname or IP to exclude file
6) Run: bin/hadoop dfsadmin -refreshNodes
7) Check NN log files for "decommission complete"
8) If decommissioned node may rejoin cluster later, remove it from exclude file

HDFS: Dealing with Failed Disks

- Technically, Hadoop doesn't detect bad disks. A path is reported as healthy from the dfs.data.dir or mapred.local.dir directories if all of the following are true:
  1) Specified path is a directory
  2) The directory exists
  3) The directory is readable
  4) The directory is writable

- DN will shut down if more disks than *dfs.datanode.failed.volumes.tolerated* fail.
- Default is zero, so a single disk failure shuts down DN!

MR: Adding a TaskTracker

1) Follow procedure for adding a DN to HDFS
2) Run balancer utility
3) Start TT
4) Check # of TTs in the JT web user interface

- It is not a technical requirement to run the balancer, but if you don’t, ensures that when the TT is assigned work, there is local block data.

MR: Killing a MR Job

1) Become HDFS superuser or user
2) Run: hadoop job –list or use JT web UI to find job ID
3) Run: hadoop job –kill jobID
4) Confirm with: hadoop job –list or in JT web UI

- Any temp map data will be discarded and job will immediately be terminated
- It is possible that a task on its final attempt is running on the worker node

MR: Blacklisted TTs

- Hadoop can temporarily blacklist TaskTrackers by removing them from the global pool of workers, either for a specific job or globally
- Any TT with 3 or more failed tasks from a single job is ineligible to receive further tasks for that job.
- If tasks from other jobs continue to fail on the bad TT, the JT adds the host to a global blacklist for 24 hours by default.
- There is currently no graceful way to remove a machine from the global blacklist since it’s in the JT’s memory. One way is to restart the bad TT, which registers as a new instance with the JT. Restarting the JT is also possible, but intrusive.

Block Scanner

- dfs.datanode.scan.period.hours → Default = 504 hours=3 weeks
- http://datanode:50075/blockScannerReport

```
Total Blocks : 39422
Verified in last hour : 84
Verified in last day : 2329
Verified in last week : 9012
Verified in last four weeks : 38240
Verified in SCAN_PERIOD : 38240
Not yet verified : 1182
Verified since restart : 45394
Scans since restart : 7492
Scan errors since restart : 0
Transient scan errors : 0
Current scan rate limit KBps : 1024
Progress this period : 108%
Time left in cur period : 2.42%
```

Block Scanner

- http://datanode:50075/blockScannerReport?listblocks

```
blk_6035596358209321442 : status : ok type : none scan time
: 0 not yet verified blk_3065580480714947643 : status : ok
type : remote scan time : 1215755306400 2008-07-11
05:48:26,400 blk_8729669677359108508 : status : ok type :
local scan time : 1215755727345 2008-07-11 05:55:27,345
```

Balancer

- Balancer runs in background, default speed limit 1 MB/s hdfs-site.xml ---> dfs.balance.bandwidthPerSec

- % start-balancer.sh

```
Time Stamp Iteration# Bytes Already Moved Bytes Left To
Move Bytes Being Moved
Mar 14, 2012 5:23:42 PM 0 0 KB 219.21 MB 150.29 MB
Mar 14, 2012 5:27:14 PM 1 124.24 MB 22.45 MB 150.29 MB
The cluster is balanced. Exiting...
Balancing took 32.03452 minutes
```

Thank You