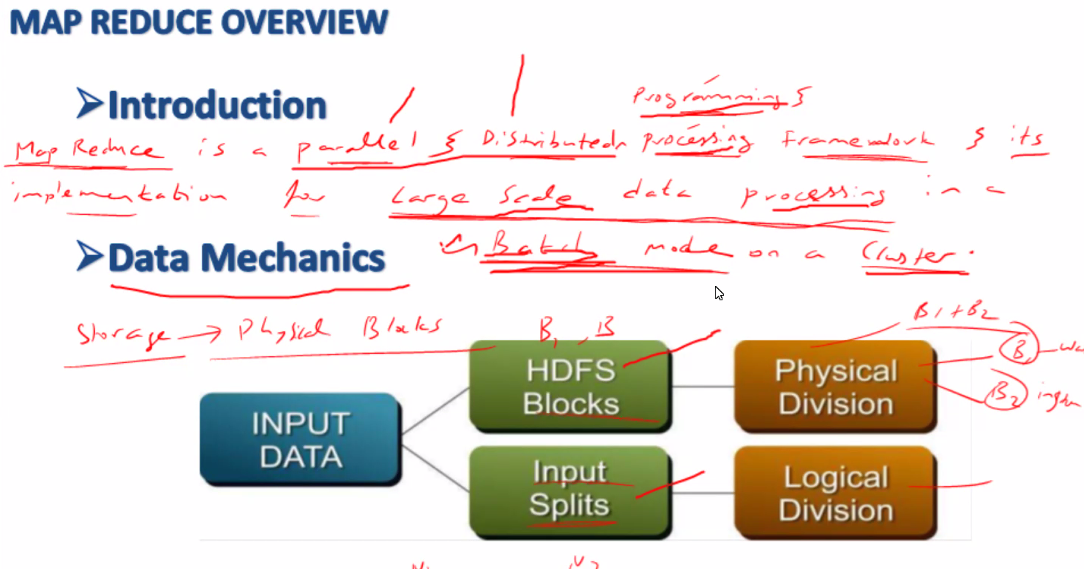
# MAP REDUCE and YARN OVERVIEW

Map reduce is a parallel and distributed programming and processing framework and its implementation for large scale data processing in a BATCH mode on a cluster.



* On Demand
* Scheduled
* Batch mode, online not supported

**Inputsplit**

Input splits are logical, an information given to Map reduce

We have below message of 130mb which is stored across 2 blocks

The last line falls in 2nd block as it exceeds 128mb, but considered to be only 1 single Inputsplit as it is a continuation of data (broken line) from last line of blk1

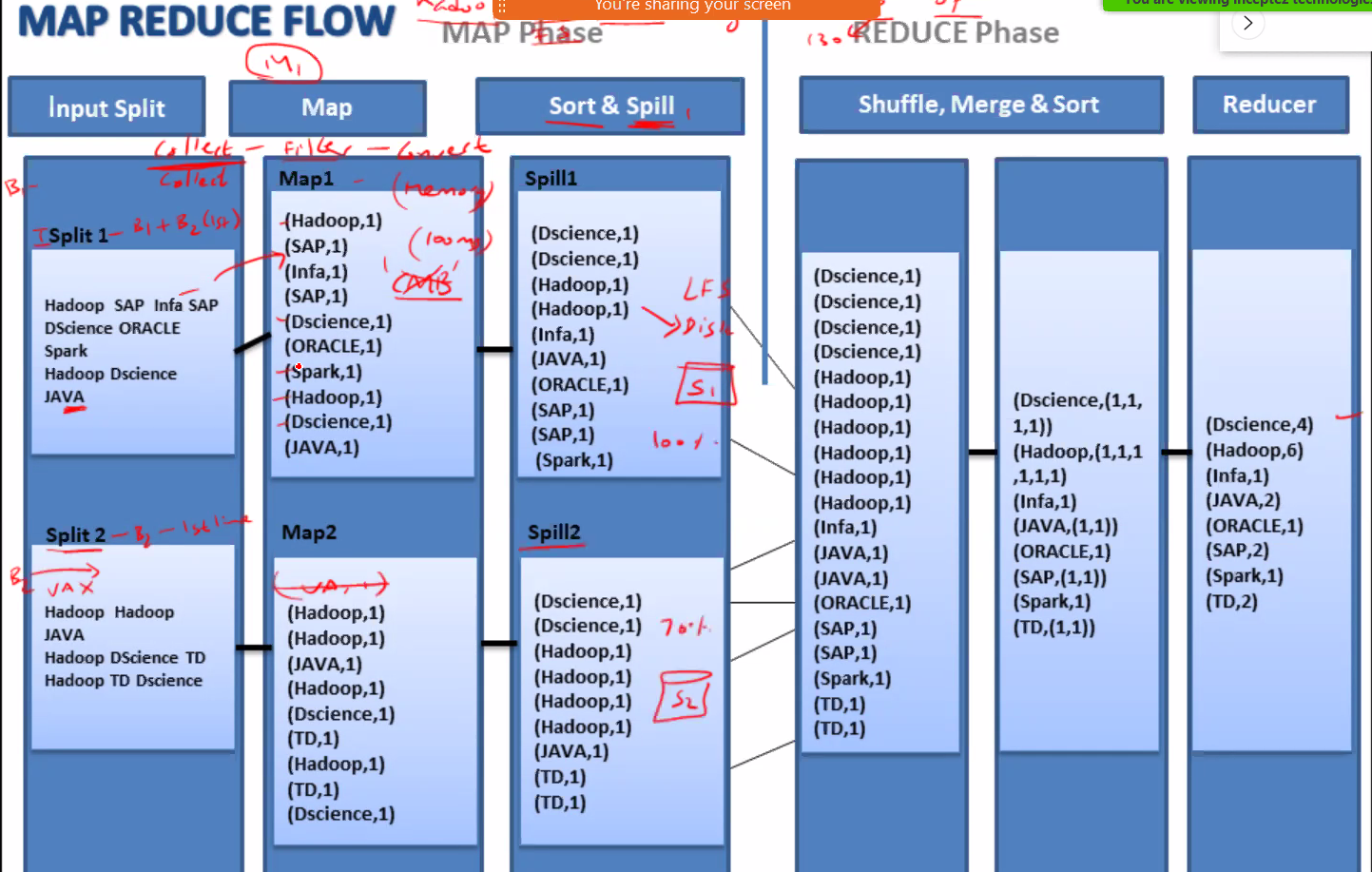
Regards,Basha 🡪 is a new line, not broken line so it is considered to be a new Inputsplit

|  |  |  |  |
| --- | --- | --- | --- |
| You make me feel like I can do anything and  I am so happy to be with you.  Thank you for being the wonderful,  amazing person that you are.  You surprise me every day and  you warm my heart every night.  I am the person I am today because  you've loved me ~~and helped me love~~ | new para with broken line | Input Split 1 | Blk1 |
| and helped me love | continuation of last line | Blk2 |
| Regards, Basha | New line | InputSplit2 |

No of Inputsplit <= No of Blocks

Inputsplit are logical splits of data whereas Blocks are physical split of data

Inputsplit are information given to mapper to know how to read data from blocks



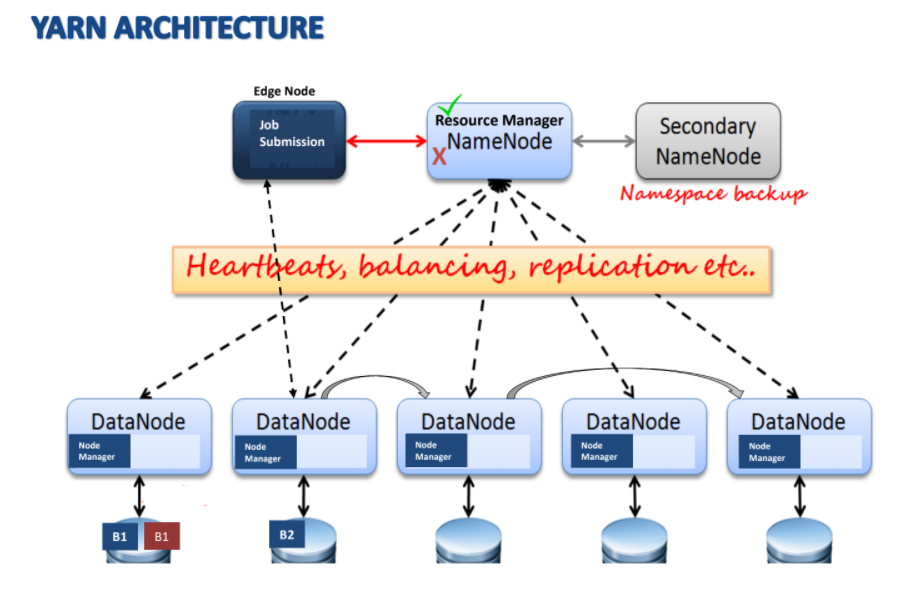
MAPPER – Collects

REDUCER – Aggregates

|  |  |
| --- | --- |
| **Map** | |
| Inputsplit | Inputsplit gives the logical information of data spread across the blocks to be collected to Map. Using this no of MAP are initiated |
| MAP 1 | Collects the data thru information provided by Inputsplit |
| SORT | Sort the collected data |
| Spill | Write the sorted data to local HDD |
| MAP2, SORT, Spill |  |
| MAP2, SORT, Spill |  |

|  |  |
| --- | --- |
| **Reduce** | |
| Shuffle | Collect the data from spill as soon as at least 1 MAP is 100% |
|  | Collect data from MAP2 |
|  | Collect data from MAP3 |
| Merge | Does not start as long as shuffle is complete |
|  | Merge all data from shuffle |
| Sort | sort the merged data |
| Reducer | Reduce the data (apply aggregate function) |

## YARN ARCHITECTURE



**YARN** is a Multi-tenant resource manager, not only used to manage sqoop but many layers

**Resource Manager** – is responsible to speak to Name Node, Client Node, NodeManager and Application Manager, coordinator application. When a job is submitted it is directed to RM not to NN

**Node Manager** – Is responsible to create the containers inside the datanode, requested by Application manager, every datanode contains a NodeManager

Once in 5 sec Node Manager sends the capacity information to Resource Manager

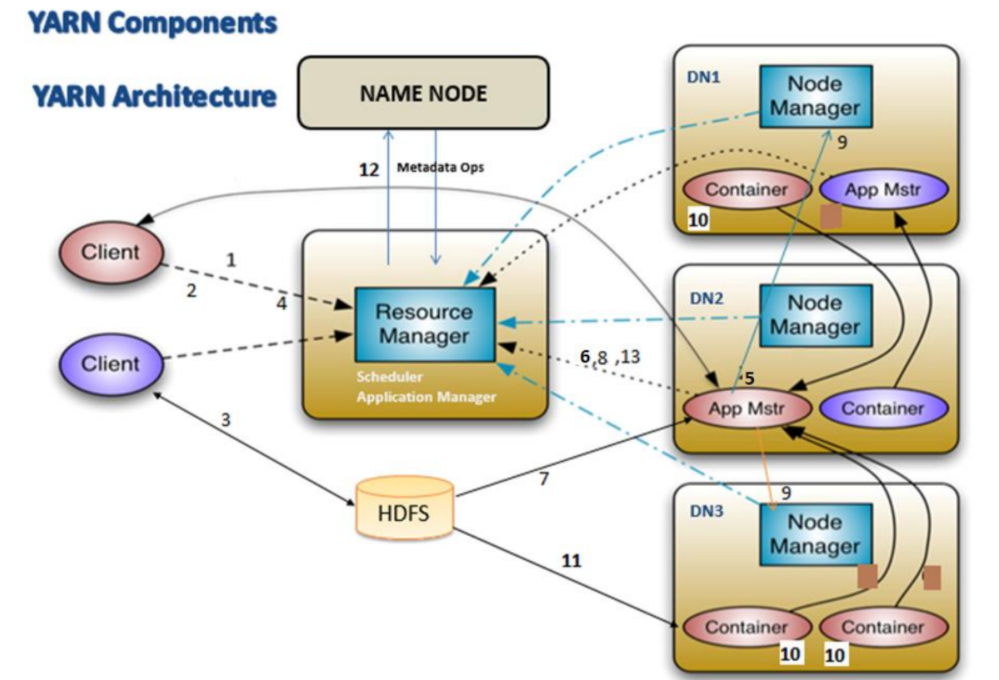
**Application Manager** – Is initiated by Resource Manager in one of the DataNodes, which get mapper data, speaks to resource manager again to get NodeManager capacity and permission to create containers inside Nodes using NodeManagers,

|  |  |
| --- | --- |
| Short Lived | |
| Application Manger | Created by Resource Manager |
|  | 1 job = 1 Application Manager |
| Containers / JVM | Any no of JVM depending on the job nature |

**When a job is submitted – scoop import -connect jdbc:\127.0.0.1\mysql table to hdfs**

* Primary contact point is Resource Manager
* Resource manager launches AM help of Node Manager
* App Master is created by resource Manager
* App manager is registered with resource Manager
* App master negotiates with Resource manager for no of mappers and reducers
* Application manager creates Mappers/reducers = JVM or containers
* When Mappers are done == 100%, then Application Manager kills JVM or container
* Application managers kills self

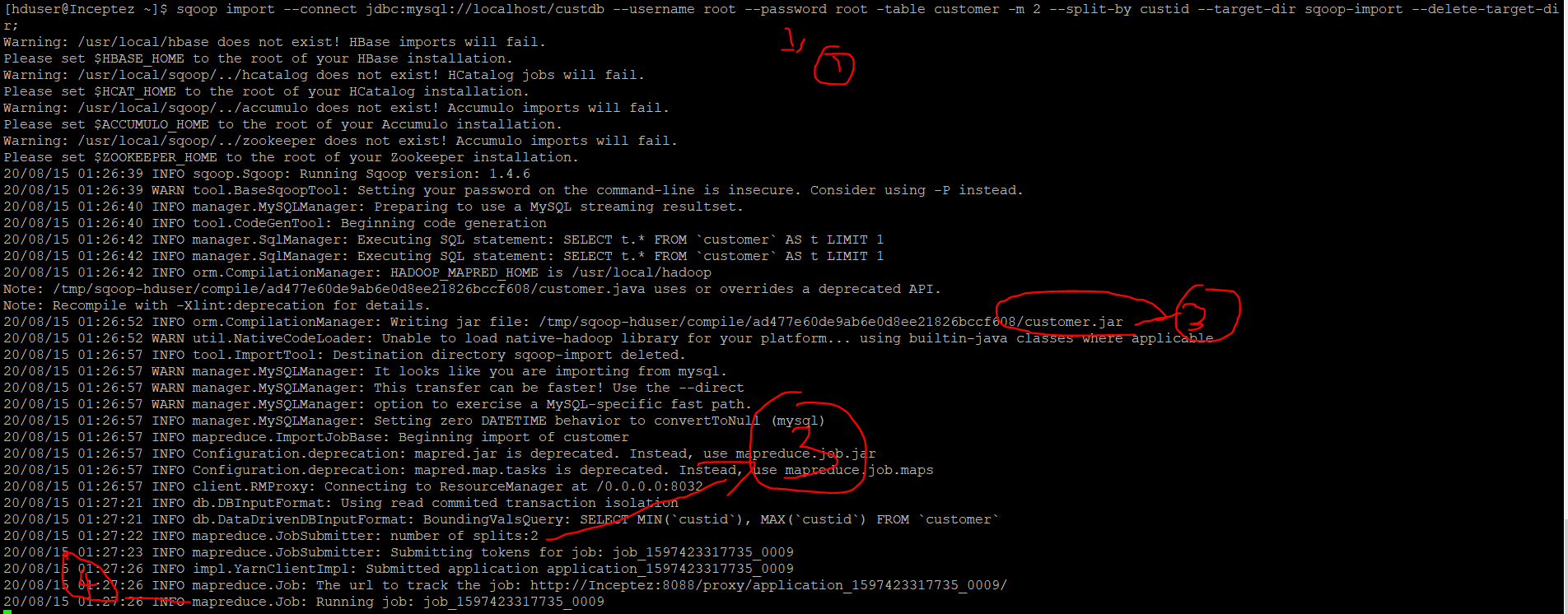
## YARN Components

Steps:

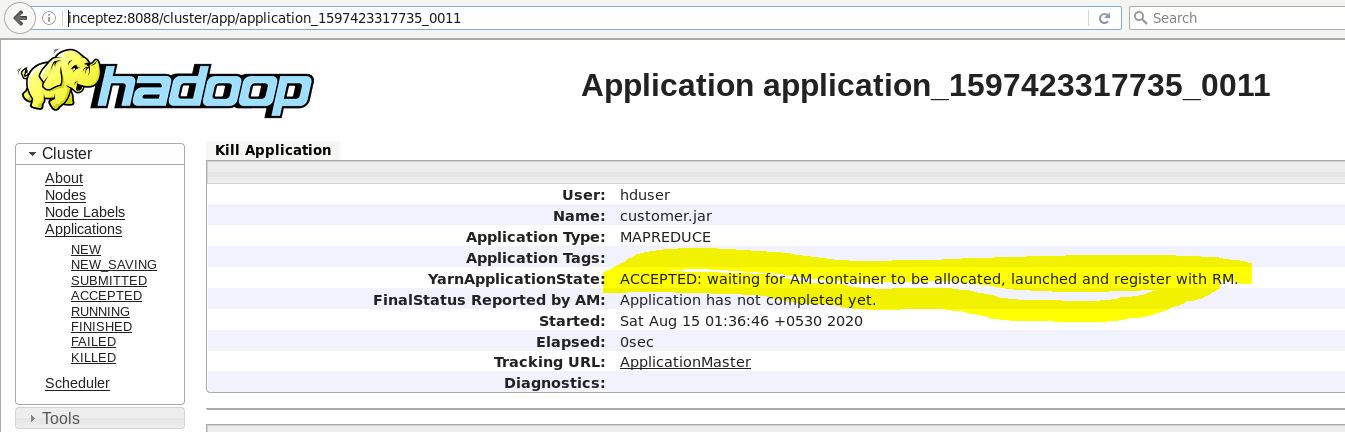
1. A scoop job is submitted from Client Node (converts to java program .jar)
   1. Contact Primary point of contact Resource Manager
   2. Resource Manager contacts NameNode to check the meta regarding the destination file/folder details, DN details
   3. Name Node approves the file name and signals the resource manager

***sqoop import --connect jdbc:mysql://localhost/custdb --username root -P -table customer -m 3 --split-by custid --target-dir sqoop\_import --delete-target-dir --direct;***

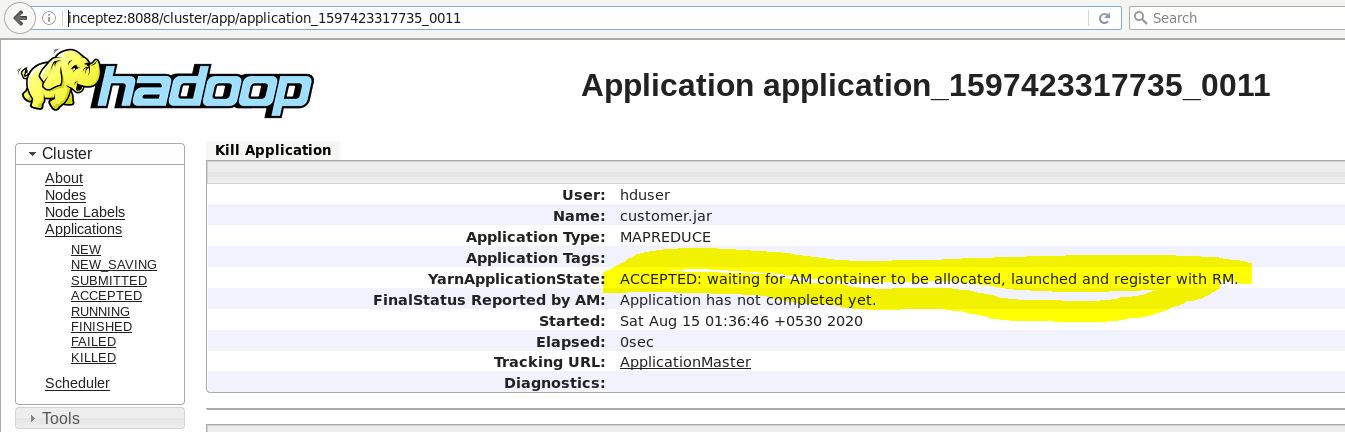
1. Resource manager creates a Job ID and provides it to Client Node alone with metadata
2. With the metadata information Client Computes INPUT SPLITs using **record reader**
   1. Record reader reaches data nodes to see last line of the block to verify broken line, then creates Input Split information
   2. Copies all the information from client node (.jar, configuration, metadata, Input splits) to a temp location in HDFS
3. Client node passes Job ID, Meta data, Input Split information as an application, to submit the job to Resource Manager



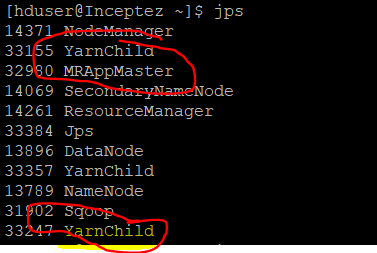
1. Resource Manager creates Application Master. It will be created (also called as container), with the help of NodeManager
   1. Resource Manager contain 2 components
      1. Scheduler 🡪 which contains the capacity information collected from NodeManager
      2. Application Manager 🡪 handle applications, launch application master as a container using NodeManger

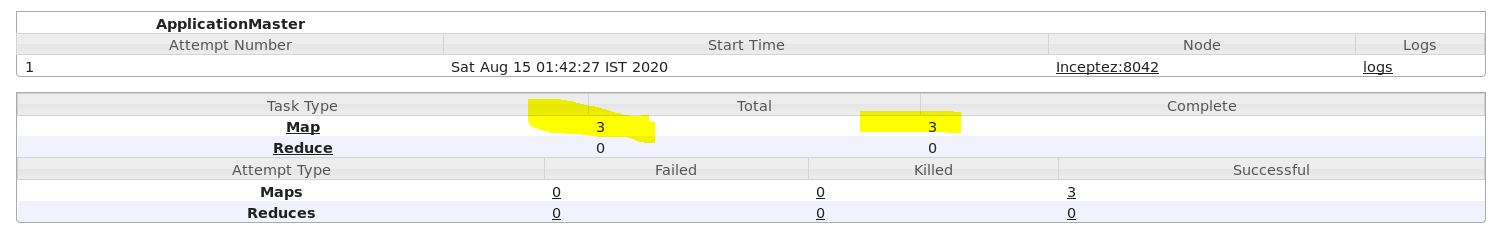


1. Application Master registers with Resource Manager

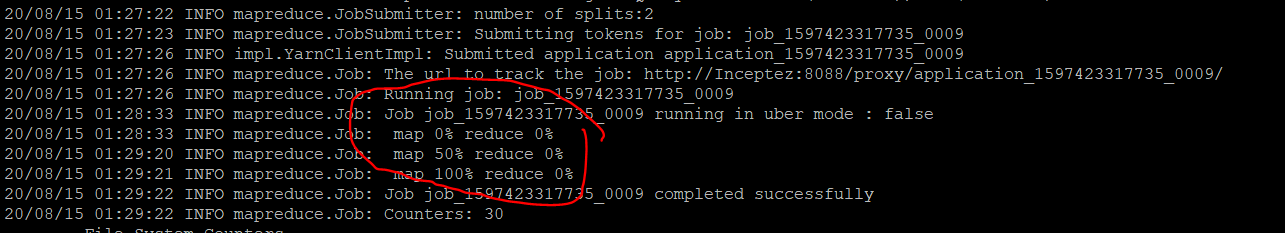


1. App master reads the job information (earlier stored in step4) all resource information required
2. App Master negotiates with Resource Manager with the information obtained from step7 for no of resources (eg.Map and Reduce )
   1. App master asks resource manager that he wants DN1 , DN2 with the information obtained to work with the NodeManger to launch the JVM in DataNodes
   2. Tries to launch the containers in respective datanodes
3. App Master works with Nodemanger to create JVM or container
4. NodeManager creates JVM or container on respective DataNodes

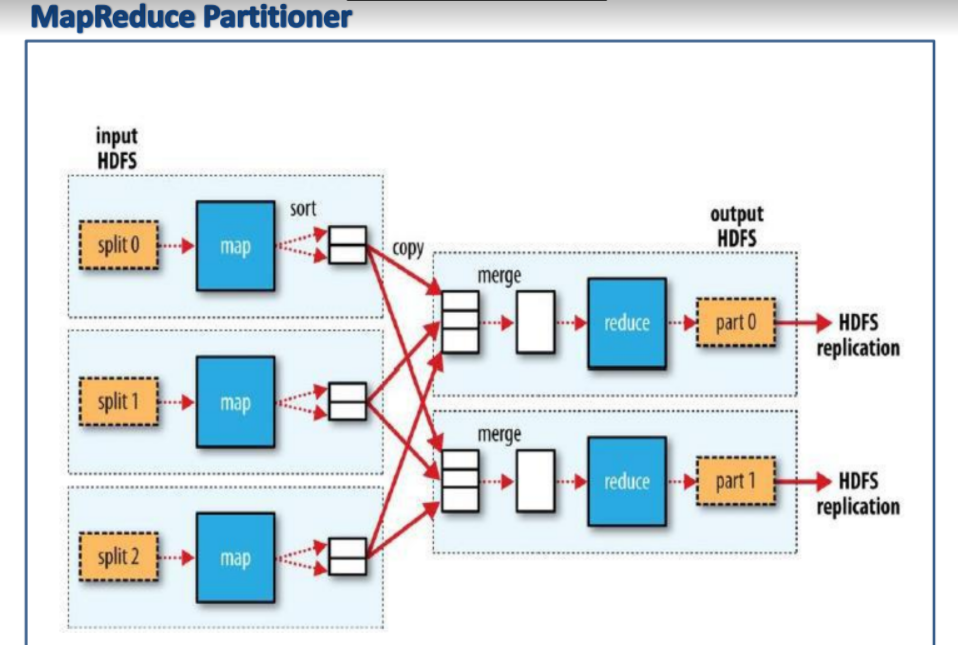


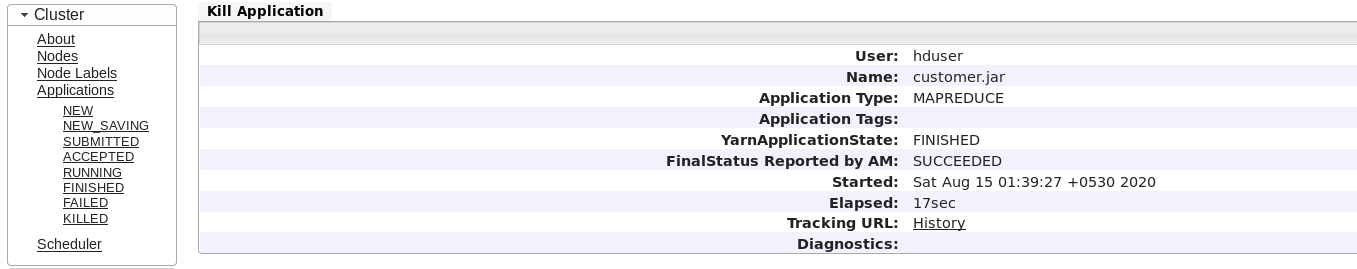


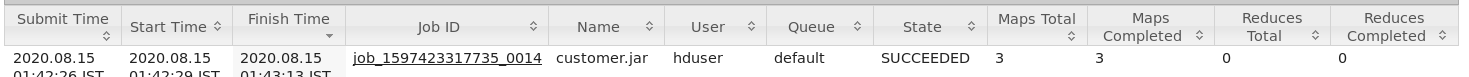
1. Copies all the job information created in HDFS temp location (step3), which is required for containers to do their job(for ex reduce program-aggregate copies in all containers – data locality) – all containers reads this information(program) to read the code from local datanode
2. Application manager sends the status of the job to client node
   1. Say we have Map Reduce job running with 2 map, 1 container
   2. When a map container is done, it informs app master
   3. App master asks reduce container to shuffle the data spilled from map
   4. Once all container works are over (map and reduce) 100%
3. Once all jobs are over jvm or container, deregister App Master from Resource Manager, and kill App Master self and all Containers created

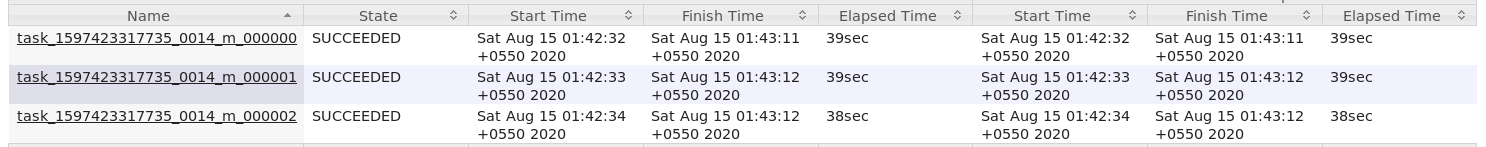


### Final result: When we have more than 1 mappers









## YARN – High availability – Fault tolerance mechanism

### UBER Mode

Also called as StandAlone - App master will take care of creation and running Map and reducer using a single Container – Provided if data is small in nature

### Speculative execution

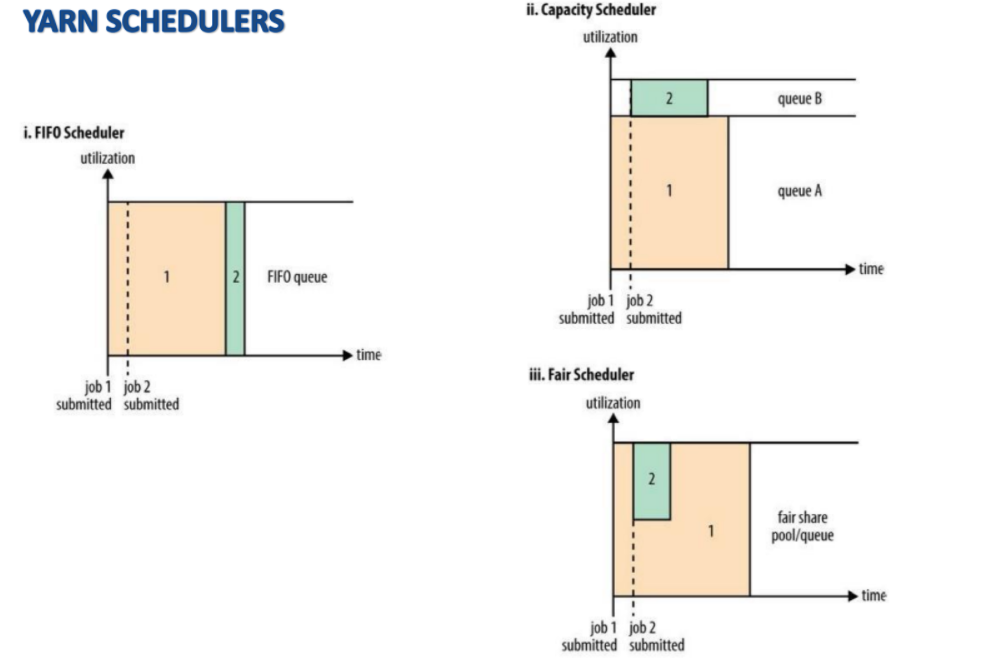
When there is a delay in one of the container, the same work is created in another container, wait for the both the container to finish.

When one of the container finishes the work, other container is killed automatically.

This is called speculative execution.

***Speculative execution*** in ***Hadoop*** MapReduce is an optimization where slow running map or reduce tasks are started on another node too

# Scheduler



Contains all resource capacity information. Scheduler decides which job to take and how to prioritize. Scheduler approves the jobs to be run.

### Types of Scheduler

**FIFO Scheduler** – First In first out, job2 waits for job1 to complete

**Capacity Scheduler** - Allocate % of CPU capacity in queue to every application 20% Marketing, 50% IT, others 30% When Market is 20%, but IT = 0, Market cannot use other capavity, it will be locked and waster – though capacity is available cannot be used

**Fair Scheduler** – same as capacity scheduler, but when Marketing queue is full, it can use IT queue, of its free. Another queue Capacity can be used