Institut TIC de Barcelona

BIG DATA

EJEMPLOS SOBRE EL CLUSTER REAL

EDUARD LARA

1. EJEMPLO MAPREDUCE EN CLUSTER

Paso 1. Una vez hemos arrancado nuestro cluster con un maestro y dos esclavos, ejecutaremos un proceso mapreduce hadoop contra el.

Primero activaremos el proceso history que nos permite guardar la información histórica de todos los procesos que vamos ejecutando.

mr-jobhistory-daemon.sh start historyserver → v2

mapred --daemon start historyserver → v3

```
hadoop@nodo1:~$ mapred --daemon start historyserver
hadoop@nodo1:~$
```

NOTA: Lo razonable es lanzar el history desde el maestro. En caso de disponer de muchas máquinas, se podrían desplegar los distintos procesos maestros por distintos sitios, es decir tener un nodo para el namenode, otro nodo para el secundarynamenode, otro nodo para el History, de manera que podamos un poco representar de una manera correcta los procesos hadoop

1. EJEMPLO MAPREDUCE EN CLUSTER

Paso 2. Utilizaremos el programa wordcount de mapreduce para contar las palabras del fichero access_log.

Creamos el directorio HDFS /prueba y subimos el fichero access_log a este directorio

hdfs dfs –mkdir /prueba

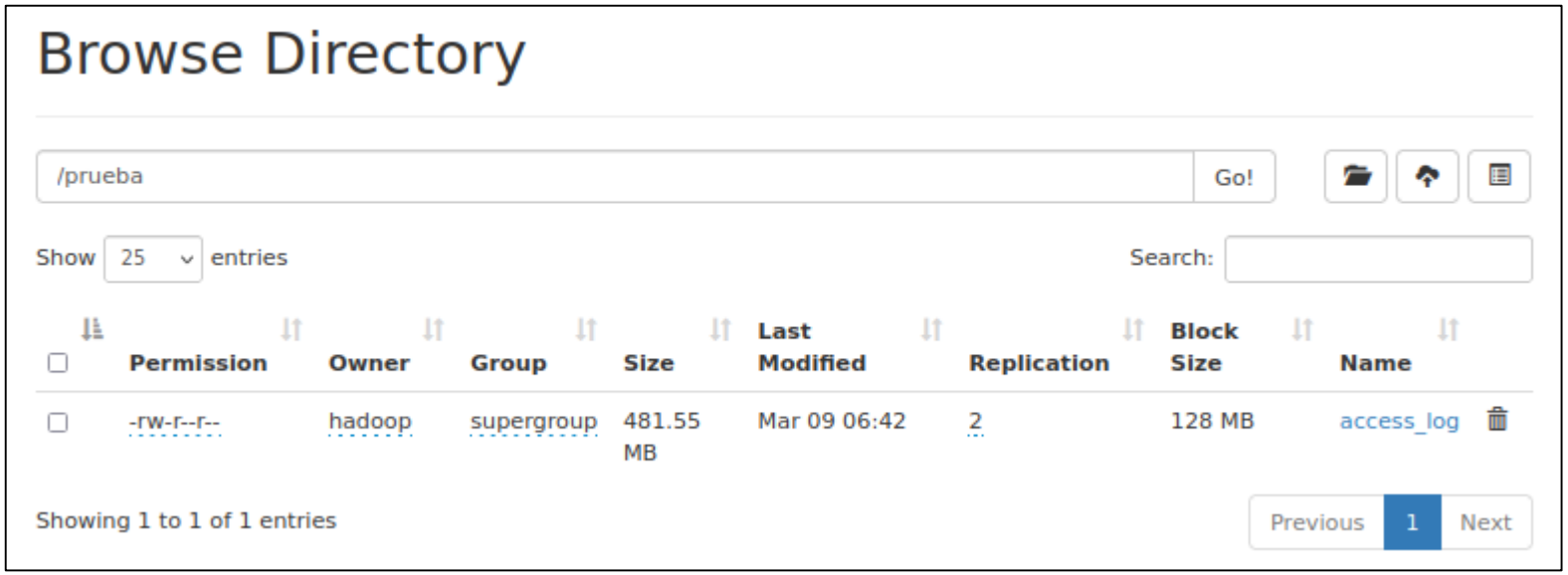
hdfs dfs –put access_log /prueba

hdfs dfs –ls /prueba

```
hadoop@nodo1:~/Downloads$ hdfs dfs -mkdir /prueba
hadoop@nodo1:~/Downloads$ hdfs dfs -put access_log /prueba
hadoop@nodo1:~/Downloads$ hdfs dfs -ls /prueba
Found 1 items
-rw-r--r--   2 hadoop supergroup  504941532 2023-03-09 06:42 /prueba/access_log
hadoop@nodo1:~/Downloads$
```

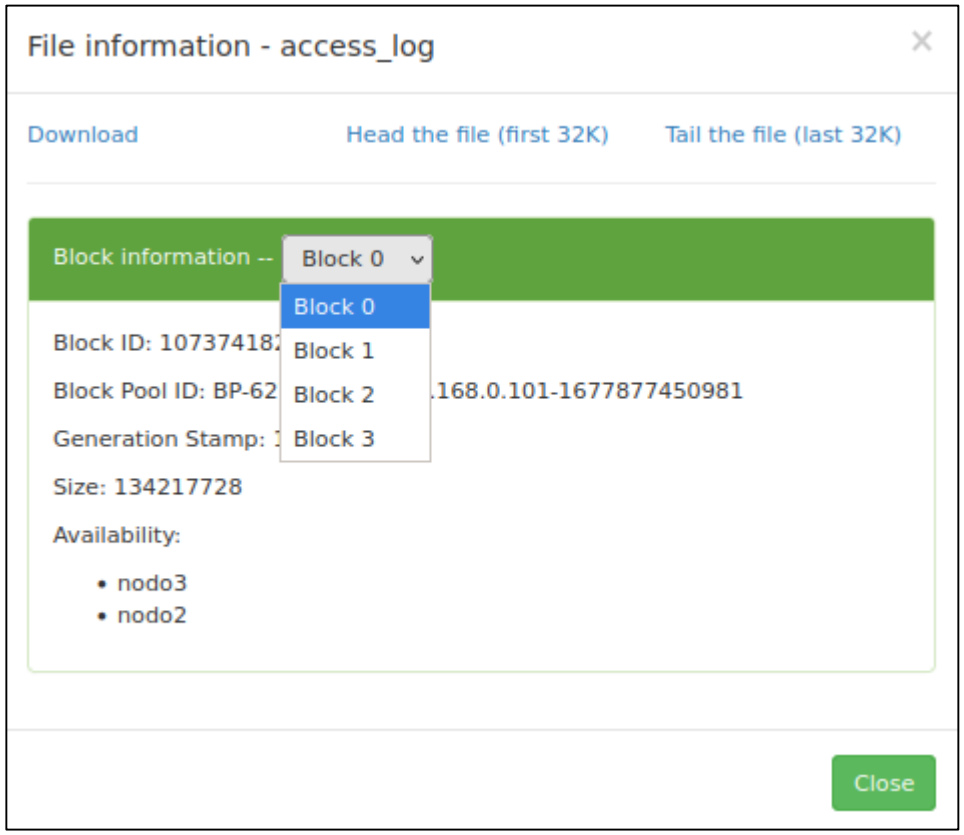
1. EJEMPLO MAPREDUCE EN CLUSTER

Paso 3. Si abrimos la wed de HDFS nodo1:9870, en Utilities/Browse the file system, dentro del directorio /prueba vemos el fichero access_log. En Replication hay un 2 indicando que sus bloques van a estar replicados en 2 nodos



1. EJEMPLO MAPREDUCE EN CLUSTER

Paso 4. Si pinchamos dentro del fichero access_log, ya sabemos que va a estar dividido en tengo 4 bloques de 128 MB (ocupa unos 500MB). Pero ahora observamos que cada bloque está reflejado en 2 nodos: nodo2 y nodo3.



1. EJEMPLO MAPREDUCE EN CLUSTER

Paso 5. En el directorio /home/hadoop/practicas tenemos el jar ContarPalabras.jar, realizado en Java para un ejemplo con mapreduce:

```
hadoop@nodo1:~/practicas$ ls -l
total 20
-rw-rw-r-- 1 hadoop hadoop 1754 feb 20 08:34 'ContarPalabras$IntSumReducer.class'
-rw-rw-r-- 1 hadoop hadoop 1751 feb 20 08:34 'ContarPalabras$TokenizerMapper.class'
-rw-rw-r-- 1 hadoop hadoop 1846 feb 20 08:34  ContarPalabras.class
-rw-rw-r-- 1 hadoop hadoop 3276 feb 20 09:08  ContarPalabras.jar
-rw-rw-r-- 1 hadoop hadoop 2934 feb 20 08:31  ContarPalabras.java
```

La arquitectura de un maestro y dos esclavos no es muy optima. Lo ideal sería tener 8, 9, 10 esclavos como mínimo para ver cómo realmente el proceso Hadoop se distribuye a través de todos ellos.

1. EJEMPLO MAPREDUCE EN CLUSTER

Paso 6. Lanzaremos el siguiente programa:

hadoop jar ContarPalabras.jar ContarPalabras /prueba/access_log /prueba_salida

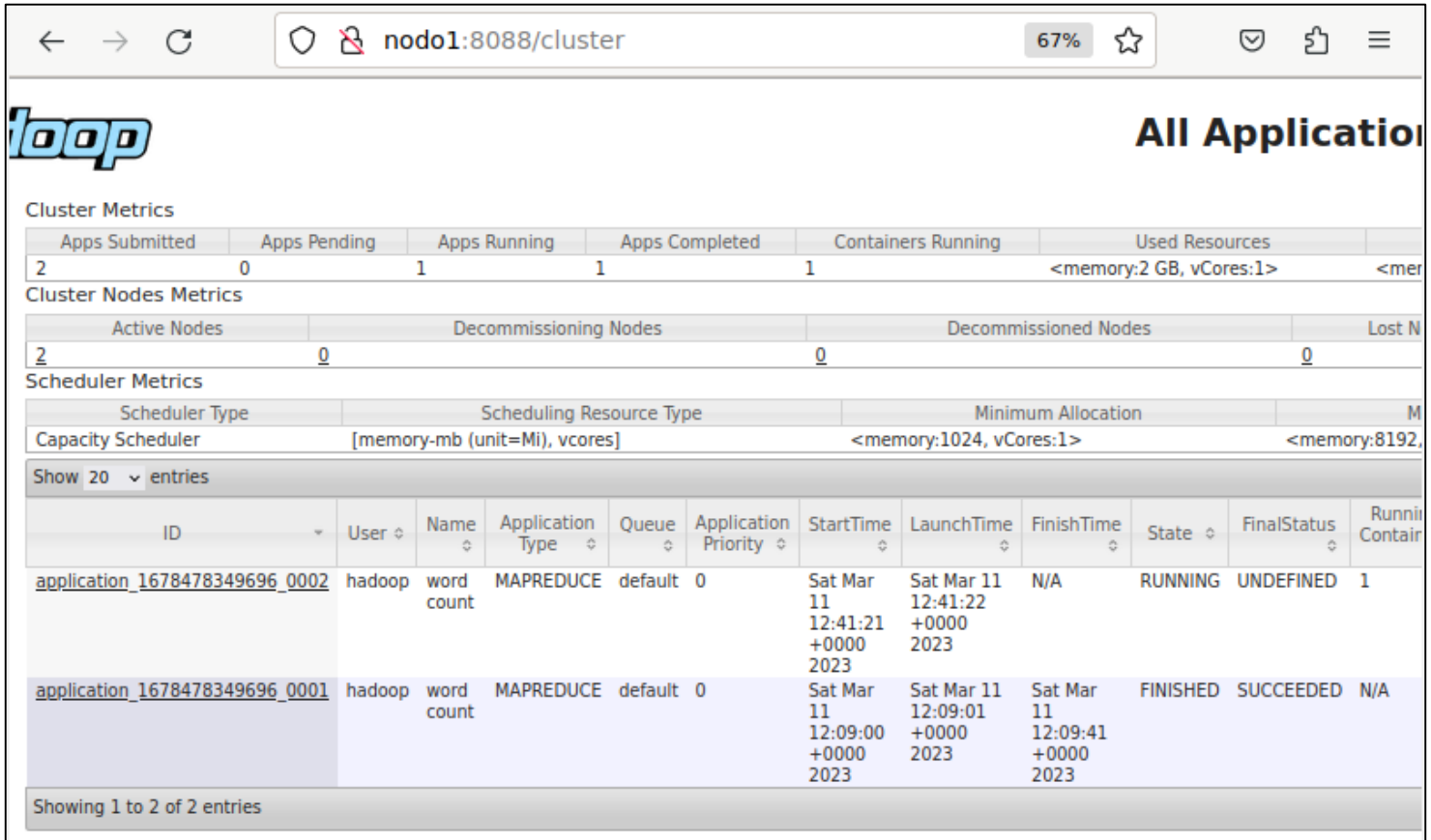
Al empezar a trabajar vemos que hace un Connecting al ResourceManager en el Nodo1, también lanza un job tipo MapReduce

```
hadoop@nodo1:~/practicas$ hadoop jar ContarPalabras.jar ContarPalabras /prueba/access_l
og /prueba_salida
2023-03-11 12:08:59,304 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192
.168.0.101:8032
2023-03-11 12:08:59,572 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding fo
r path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1678478349696_0001
2023-03-11 12:09:00,288 INFO input.FileInputFormat: Total input files to process : 1
2023-03-11 12:09:00,364 INFO mapreduce.JobSubmitter: number of splits:4
2023-03-11 12:09:00,448 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_167
8478349696_0001
2023-03-11 12:09:00,449 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-11 12:09:00,581 INFO conf.Configuration: resource-types.xml not found
2023-03-11 12:09:00,581 INFO resource.ResourceUtils: Unable to find 'resource-types.xml
'.
2023-03-11 12:09:00,917 INFO impl.YarnClientImpl: Submitted application application_167
8478349696_0001
2023-03-11 12:09:00,957 INFO mapreduce.Job: The url to track the job: http://nodo1:8088
/proxy/application_1678478349696_0001/
2023-03-11 12:09:00,958 INFO mapreduce.Job: Running job: job_1678478349696_0001
2023-03-11 12:09:06,030 INFO mapreduce.Job: Job job_1678478349696_0001 running in uber
mode : false
2023-03-11 12:09:06,031 INFO mapreduce.Job:  map 0% reduce 0%
```

1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Running

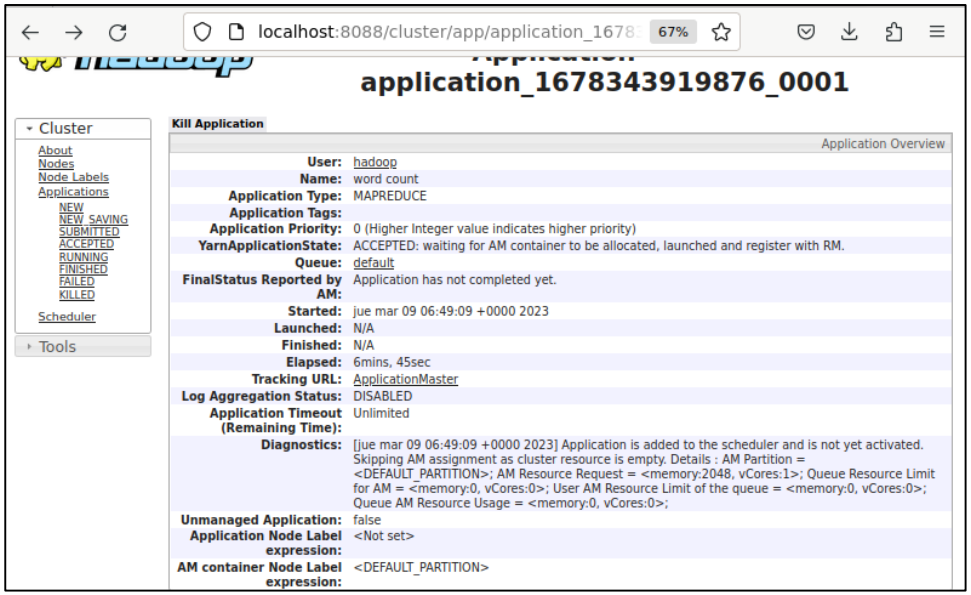
Paso 7. Cuando empieza con el proceso MapReduce, vamos a la web de administración nodo1:8088. Vemos que nuestro proceso lanzado esta en estado RUNNING



1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Running

Paso 8. Entramos dentro del proceso y aquí tenemos el tracking URL ApplicationMaster que es el proceso que consume



1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Running

Paso 9. Si hacemos un jps en nodo1, vemos que junto al proceso resourcemanager, esta runJar que es la aplicación que ejecutamos

```
hadoop@nodo1:~$ jps
41521 Jps
4520 ResourceManager
34267 JobHistoryServer
4043 SecondaryNameNode
3741 NameNode
38493 RunJar
hadoop@nodo1:~$
```

Paso 10. A través de un ssh vamos al nodo2 y hacemos un jps os distintos procesos que va abriendo

1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Running

Paso 11. En otra solapa nos vamos al nodo3 y vemos que le está costando esta vez. Si hacemos un jps veremos que más pronto más tarde voy a poder ver los procesos que se van a ir arrancando según vayan arrancando los mapear y los reducen. Podemos ver aquí el AppMaster en el nodo3 se llama así MRAppMaster Por cada proceso aplicación que lanzábamos se creaba un APP Master y luego creábamos los procesos Yarn

Paso 12. Hacemos otro jps en el nodo3 y ahora vemos un AppMaster y un YarnChild. En ndo2 podemos tener seguramente algo parecido. Si no lo vemos es porque no ha terminado o todavía no ha llegado a hacer ningún proceso.

1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

Paso 13. La ejecución finaliza rápidamente en una maquina de 32 GB

```
                Spilled Records=134335500
                Shuffled Maps =4
                Failed Shuffles=0
                Merged Map outputs=4
                GC time elapsed (ms)=3025
                CPU time spent (ms)=73610
                Physical memory (bytes) snapshot=2694070272
                Virtual memory (bytes) snapshot=12736823296
                Total committed heap usage (bytes)=2578448384
                Peak Map Physical memory (bytes)=597168128
                Peak Map Virtual memory (bytes)=2547679232
                Peak Reduce Physical memory (bytes)=314335232
                Peak Reduce Virtual memory (bytes)=2549448704
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=504953820
        File Output Format Counters
                Bytes Written=57779668
hadoop@nodo1:~/practicas$
```

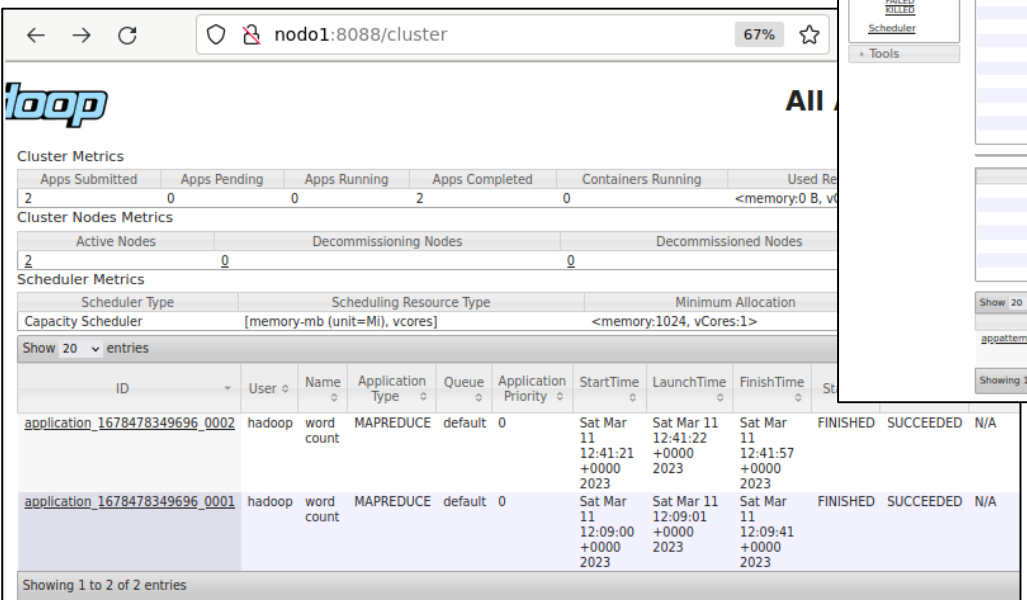
Paso 14. En el directorio de los resultados vemos el fichero SUCCESS

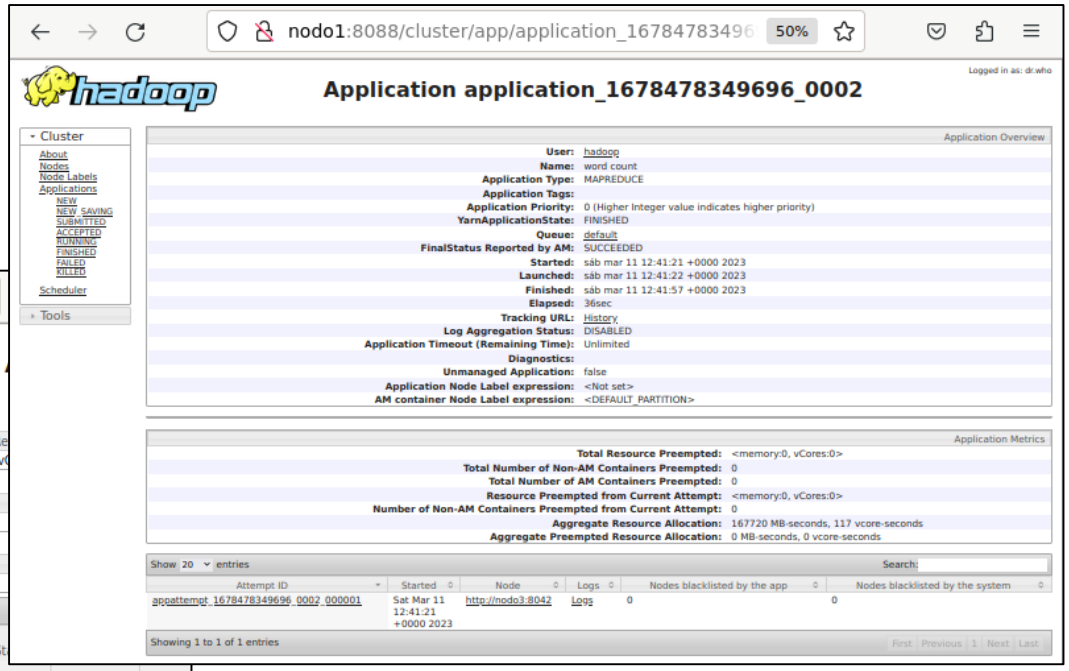
```
hadoop@nodo1:~/practicas$ hdfs dfs -ls /prueba_salida
Found 2 items
-rw-r--r--   2 hadoop supergroup          0 2023-03-11 12:09 /prueba_salida/_SUCCESS
-rw-r--r--   2 hadoop supergroup   57779668 2023-03-11 12:09 /prueba_salida/part-r-0000
0
hadoop@nodo1:~/practicas$
```

1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

Paso 15. En nodo1:8088 vemos que los procesos lanzados están Finished. Si hacemos click ApplicationID, llegamos a la estadísticas del proceso lanzado, donde podemos acceder al History y al attempt

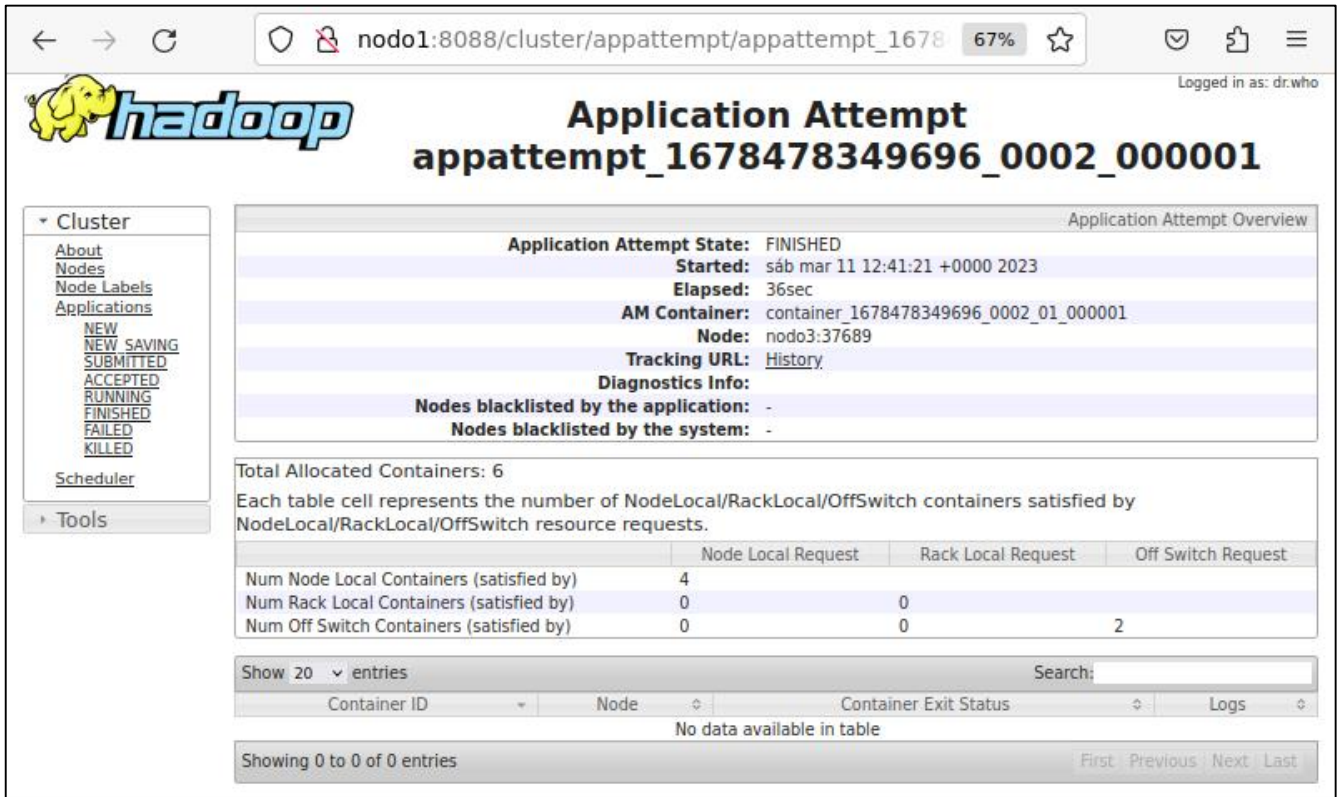




1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

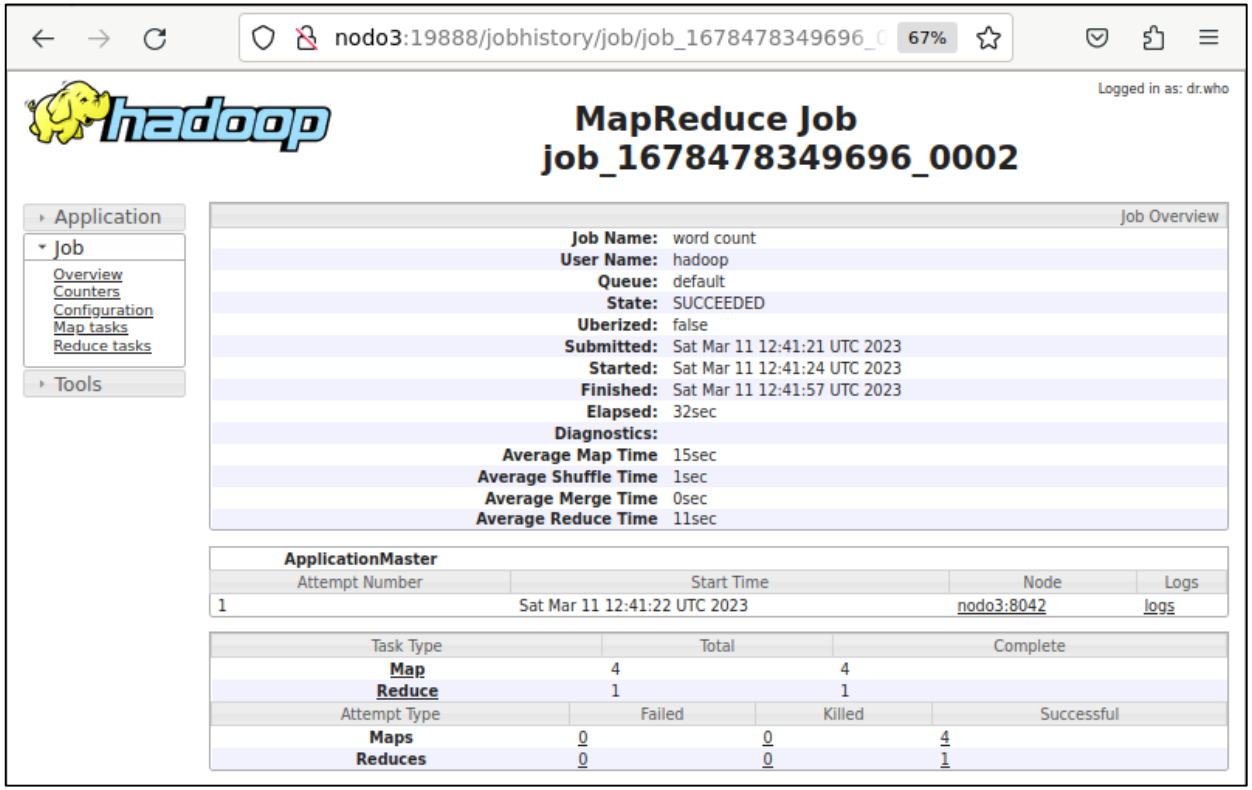
Paso 16. Si hacemos click en attempt del proceso, vemos que ha pedido 4 containers (o procesos) y ha realizado dos peticiones, en el nodo3. Ha arrancado 1 APPMaster y 6 containers. Esto lo decide él, según los recursos de que dispone



1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

Paso 17. Si hacemos click en Tracking URL: History, vemos que el AppMaster lo ha arrancado en nodo3. Ha creado 4 mapers y un reduce de los cuales ha conseguido terminar todos. No ha tenido ningún fallo y no ha tenido que eliminar ninguno (puede reiniciarlo en otro nodo).

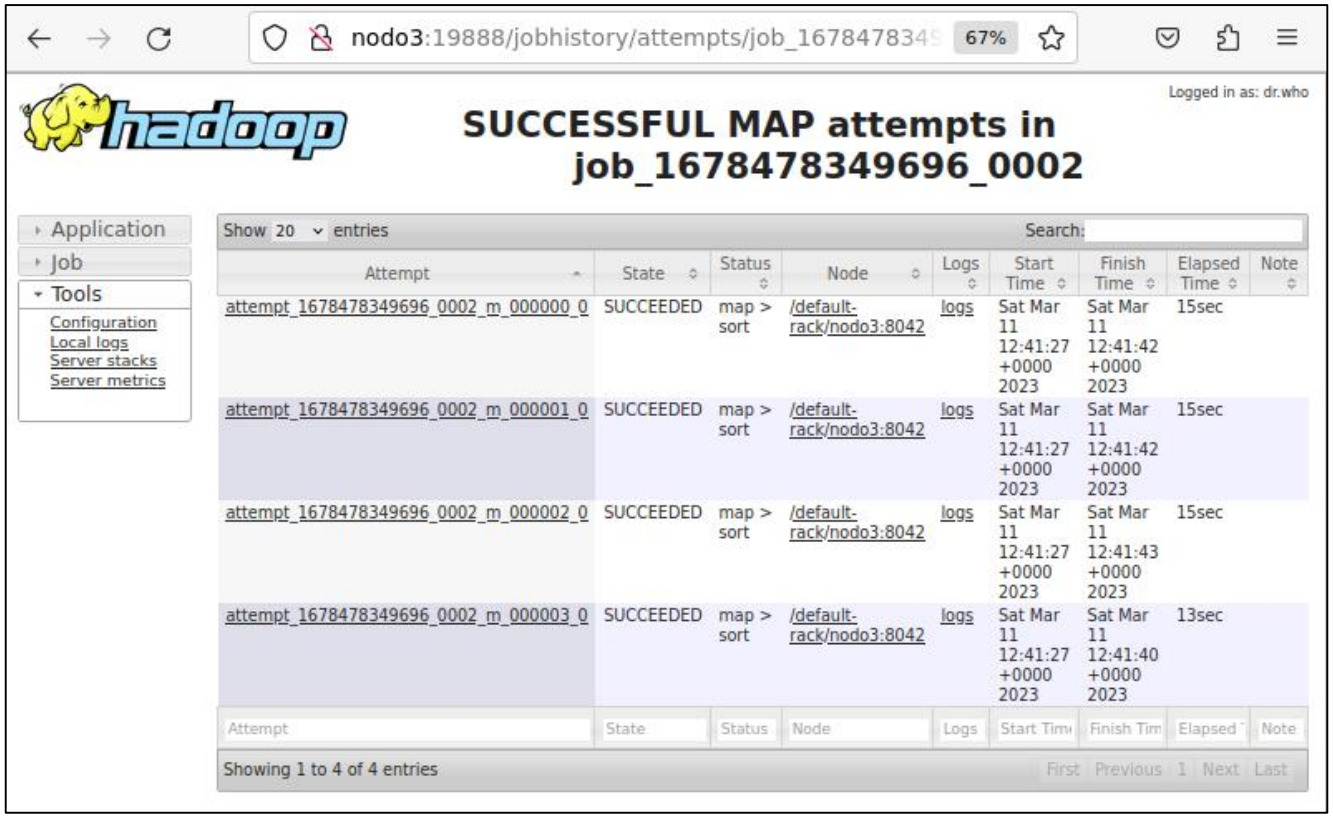


Tambien indica que todo ha sido satisfactorio y el tiempo que ha tardado.

1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

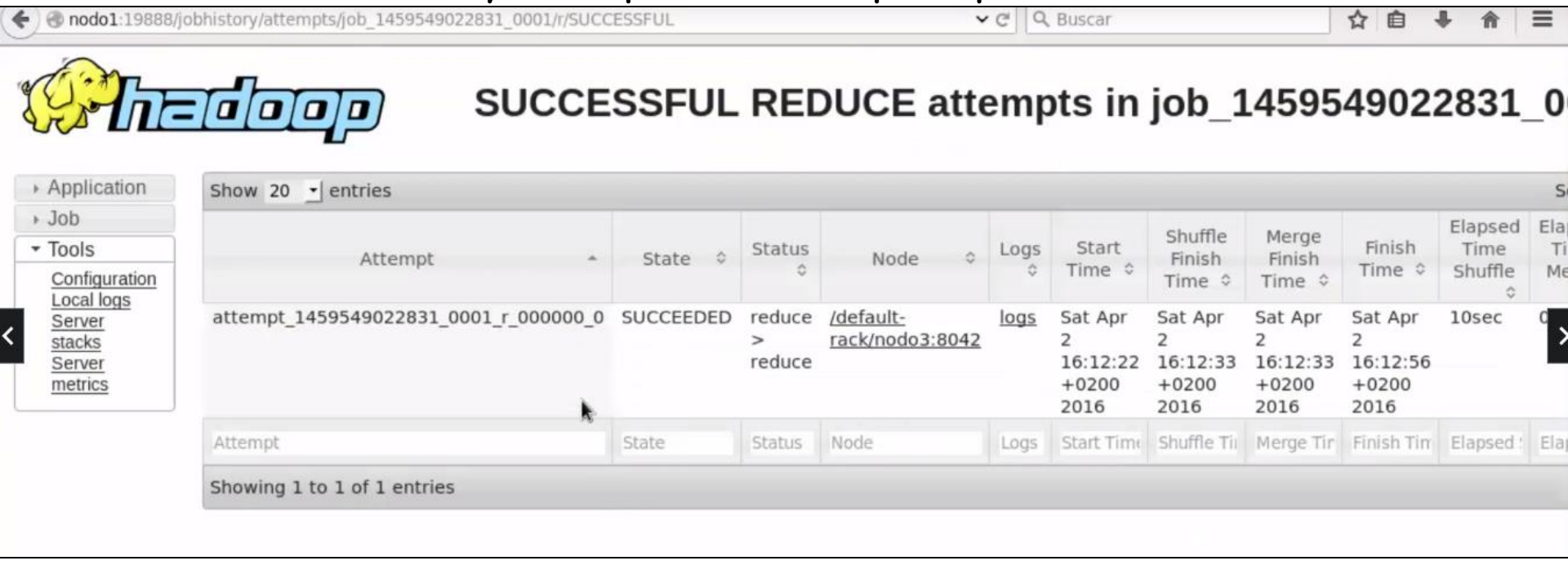
Paso 18. Si entramos dentro de los procesos MapReduce (sobre el enlace 4) podemos ver como han ido actuando los procesos Reduce, cuánto han tardado etc.



1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

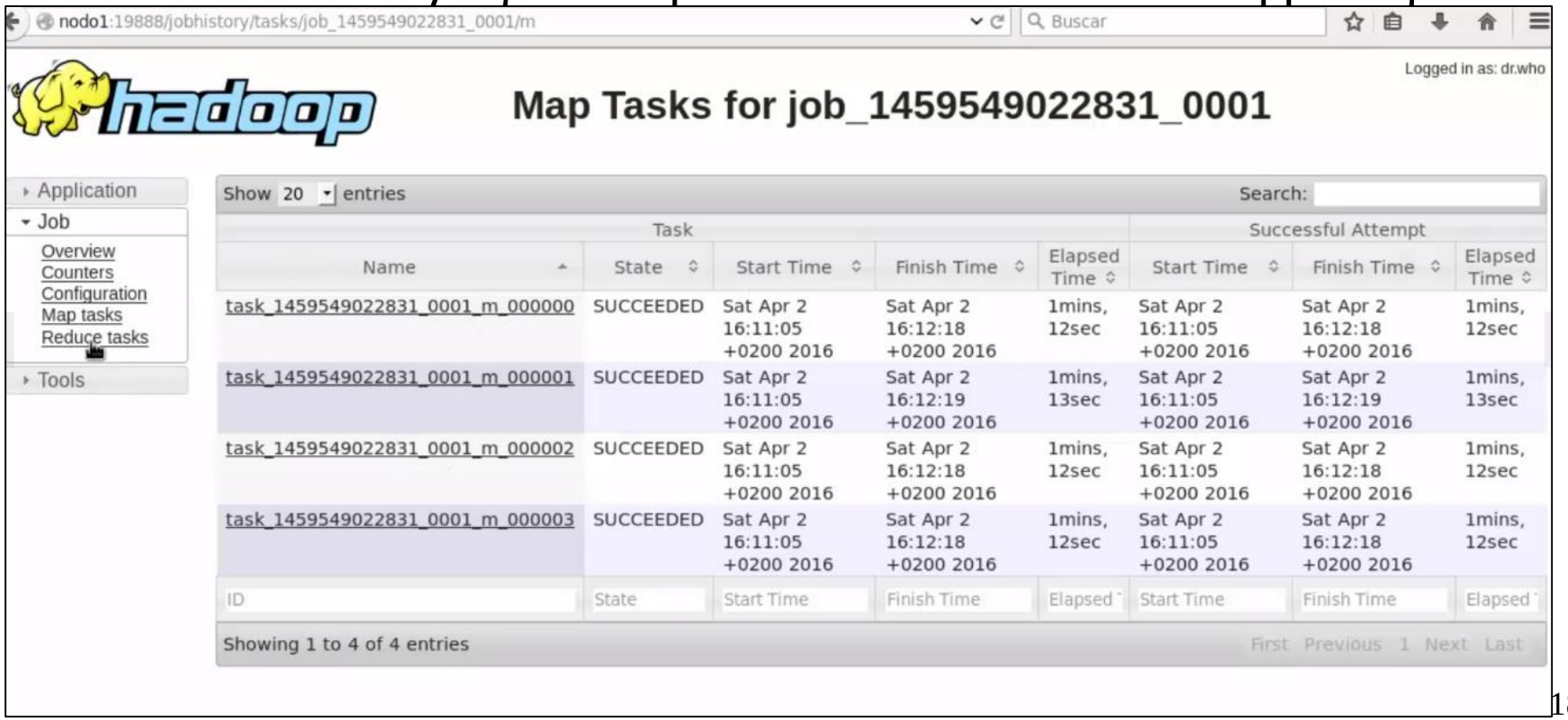
Paso 19. Si me voy a los procesos mapers puedo ver como ha ido



1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

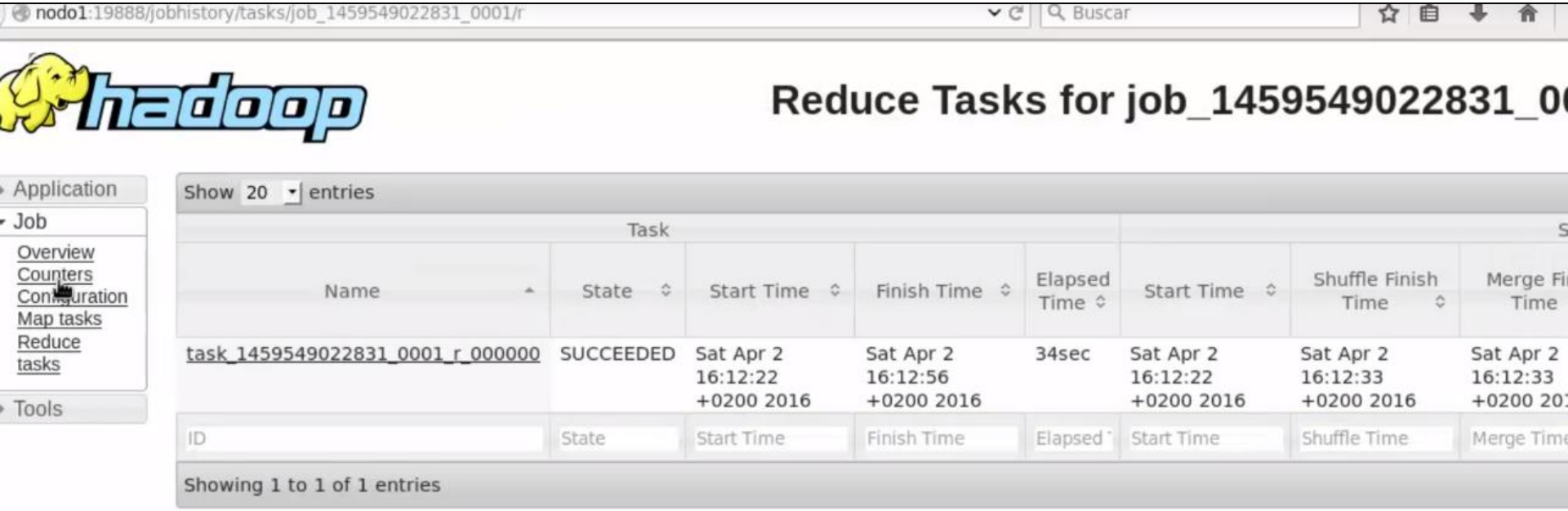
Paso 20. Esta arquitectura que estamos probando no es la más satisfactoria porque si tengo sólo dos nodos lo puede tener difícil para decidir. Si me voy aquí a MapTask vemos las cuatro Mappers que ha



1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

Paso 21. Si vamos a Reduce Tasks, vemos los reduced que ha hecho



1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

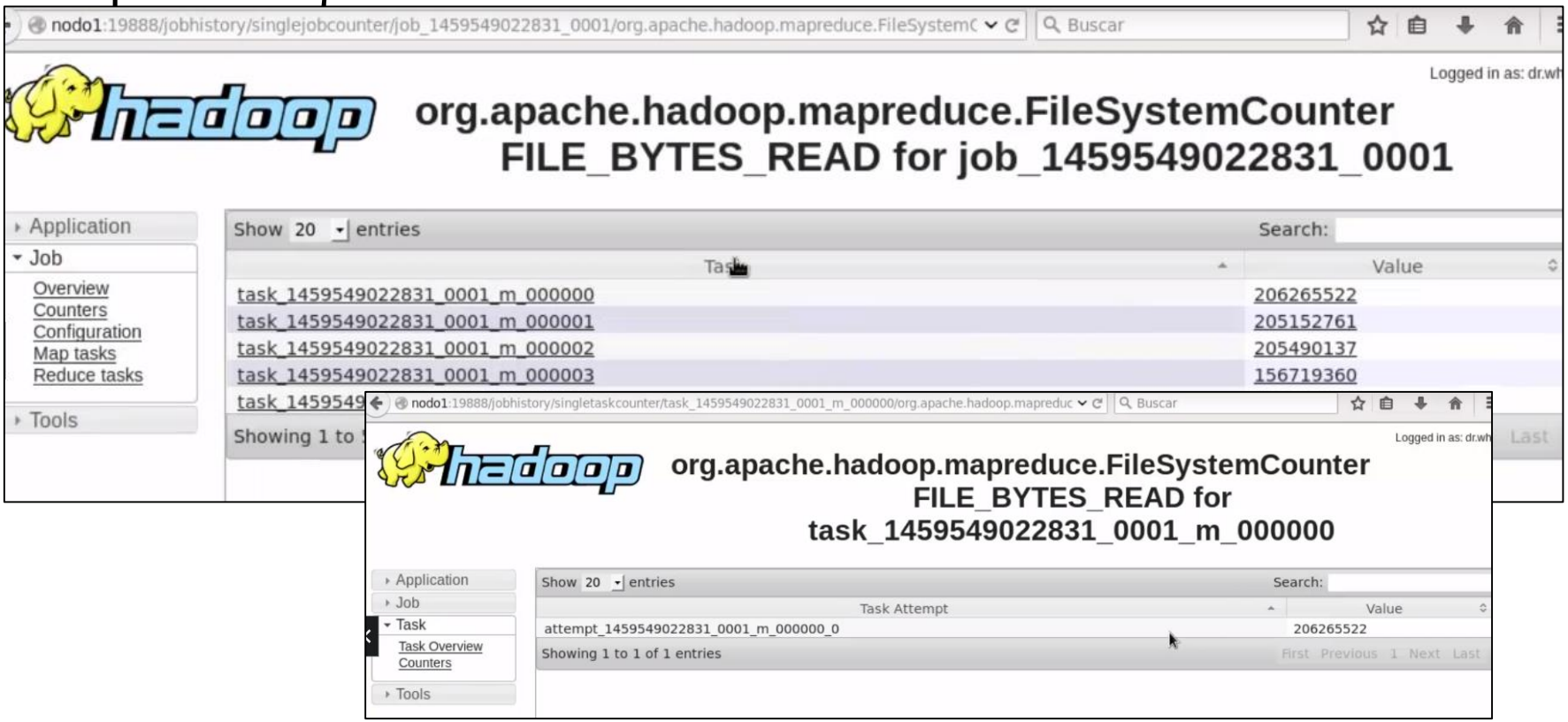
Paso 22. En counters puedo ver todos los procesos y los contadores que ha tenido que realizar para esos procesos

Counter Group	Counters			
	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	773.627.780	773.627.690	1.547.255.470
	FILE: Number of bytes written	1.547.724.114	773.744.796	2.321.468.910
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	504.954.220	0	504.954.220
	HDFS: Number of bytes written	0	57.779.668	57.779.668
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	12	3	15
	HDFS: Number of write operations	0	2	2
Job Counters	Name	Map	Reduce	Total
	Data-local map tasks	0	0	4
	Launched map tasks	0	0	4
	Launched reduce tasks	0	0	1
	Total megabyte-milliseconds taken by all map tasks	0	0	299.696.128
	Total megabyte-milliseconds taken by all reduce tasks	0	0	35.111.936
	Total time spent by all map tasks (ms)	0	0	292.672
	Total time spent by all maps in occupied slots (ms)	0	0	292.672
	Total time spent by all reduce tasks (ms)	0	0	34.289

1. EJEMPLO MAPREDUCE EN CLUSTER

Proceso Finished

Paso 23. En file_byres_read podemos incluso meternos cuantos ficheros ha leído. Si pincho es me dice cada tarea que es lo que ha hechos Haciendo click puedo llega hasta la parte más profunda del aplicativo que se ha lanzado.



2. PRACTICA CON PROGRAMA JAVA

Paso 1. Usaremos un fichero de ejemplo de patentes en Estados Unidos, cite75_99.txt, que subiremos a la carpeta prácticas de HDFS

hdfs dfs -put cite75_99.txt /practicas

```
hadoop@nodo1:~/Downloads$ hdfs dfs -mkdir /practicas
hadoop@nodo1:~/Downloads$ hdfs dfs -put cite75_99.txt /practicas
hadoop@nodo1:~/Downloads$ hdfs dfs -ls /practicas
Found 1 items
-rw-r--r--   2 hadoop supergroup  264075431 2023-03-12 08:39 /practicas/cite75_99.txt
hadoop@nodo1:~/Downloads$
```

Paso 2. Descargamos del Moodle un programa MyJob.java, el cual agrupa las patentes por el código principal

Paso 3. Exportamos la librería para localizarla en el CLASSPATH

export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar

```
hadoop@nodo1:~/Downloads$ export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
hadoop@nodo1:~/Downloads$ echo $HADOOP_CLASSPATH
/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar
hadoop@nodo1:~/Downloads$
```

2. PRACTICA CON PROGRAMA JAVA

Paso 4. Compilamos

hadoop com.sun.tools.javac.Main MyJob.java

```
hadoop@nodo1:~/Downloads$ hadoop com.sun.tools.javac.Main MyJob.java
/opt/hadoop/libexec/hadoop-functions.sh: line 2366: HADOOP_COM.SUN.TOOLS.JAVAC.MAIN_USER: invalid variable name
/opt/hadoop/libexec/hadoop-functions.sh: line 2461: HADOOP_COM.SUN.TOOLS.JAVAC.MAIN_OPTS: invalid variable name
Note: MyJob.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
hadoop@nodo1:~/Downloads$
```

Paso 5. Activamos el history para ver los mappers y los reduce

mapred --daemon start historyserver

Paso 6. Creamos el JAR correspondiente

jar cvf MyJob.jar My*

```
hadoop@nodo1:~/Downloads$ jar cvf MyJob.jar My*
manifiesto agregado
agregando: MyJob$MapClass.class(entrada = 1490) (salida = 582)(desinflado 60%)
agregando: MyJob$Reduce.class(entrada = 1815) (salida = 786)(desinflado 56%)
agregando: MyJob.class(entrada = 2028) (salida = 958)(desinflado 52%)
agregando: MyJob.java(entrada = 2293) (salida = 728)(desinflado 68%)
hadoop@nodo1:~/Downloads$
```

2. PRACTICA CON PROGRAMA JAVA

Paso 7. Ejecutamos

hadoop jar MyJob.jar MyJob /practicas/cite75_99.txt /resultado7

```
hadoop@nodo1:~/Downloads$ hadoop jar MyJob.jar MyJob /practicas/cite75_99.txt /resultado7
2023-03-12 09:05:00,138 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-03-12 09:05:00,341 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/st
2023-03-12 09:05:00,463 INFO input.FileInputFormat: Total input files to process : 1
2023-03-12 09:05:00,516 INFO mapreduce.JobSubmitter: number of splits:2
2023-03-12 09:05:00,600 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1678604403322_0002
2023-03-12 09:05:00,601 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-12 09:05:00,688 INFO conf.Configuration: resource-types.xml not found
2023-03-12 09:05:00,688 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-03-12 09:05:00,721 INFO impl.YarnClientImpl: Submitted application application_1678604403322_0002
2023-03-12 09:05:00,746 INFO mapreduce.Job: The url to track the job: http://nodo1:8088/proxy/application_1678604
2023-03-12 09:05:00,747 INFO mapreduce.Job: Running job: job_1678604403322_0002
2023-03-12 09:05:04,797 INFO mapreduce.Job: Job job_1678604403322_0002 running in uber mode : false
2023-03-12 09:05:04,798 INFO mapreduce.Job:  map 0% reduce 0%
2023-03-12 09:05:19,892 INFO mapreduce.Job:  map 72% reduce 0%
2023-03-12 09:05:21,907 INFO mapreduce.Job:  map 100% reduce 0%
```

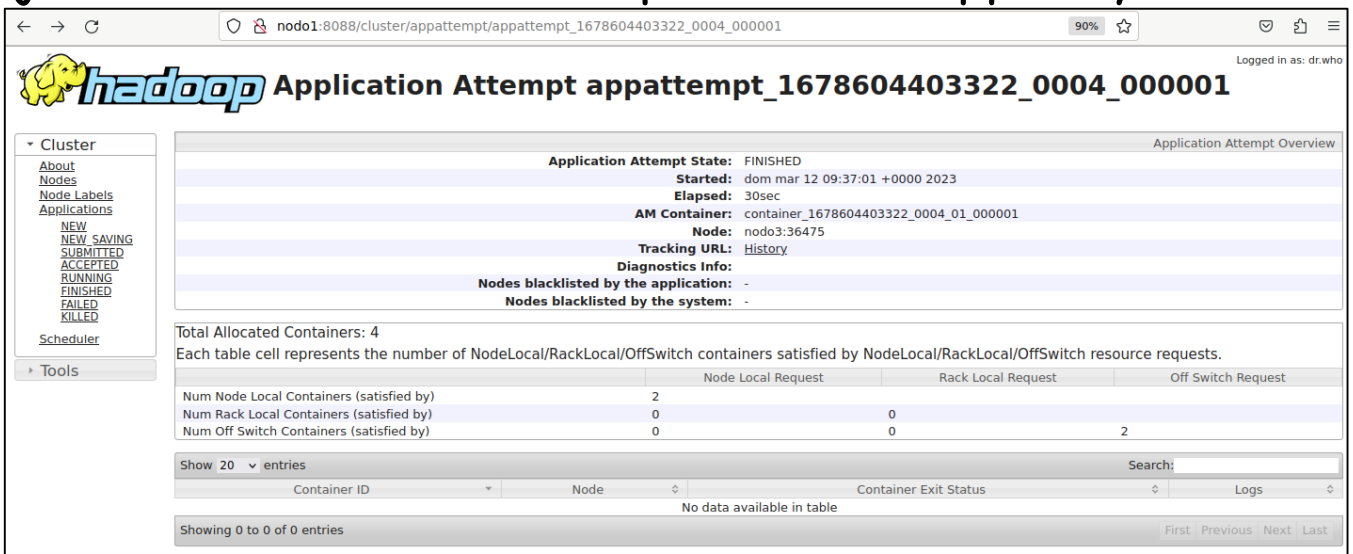
2. PRACTICA CON PROGRAMA JAVA

Paso 8. Comprobamos el resultado dejado en el directorio "resultado7"

hdfs dfs -cat /resultado7/part-r-00000

```
hadoop@nodo1:~/Downloads$ hdfs dfs -ls /resultado7
Found 2 items
-rw-r--r--   2 hadoop supergroup          0 2023-03-12 09:05 /resultado7/_SUCCESS
-rw-r--r--   2 hadoop supergroup  158078539 2023-03-12 09:05 /resultado7/part-r-00000
hadoop@nodo1:~/Downloads$
```

Paso 9. Podemos ver en la página de Administración de YARN donde se ha ejecutado cada uno de los procesos, mappers y reducers



3. PRACTICA CON STREAMING

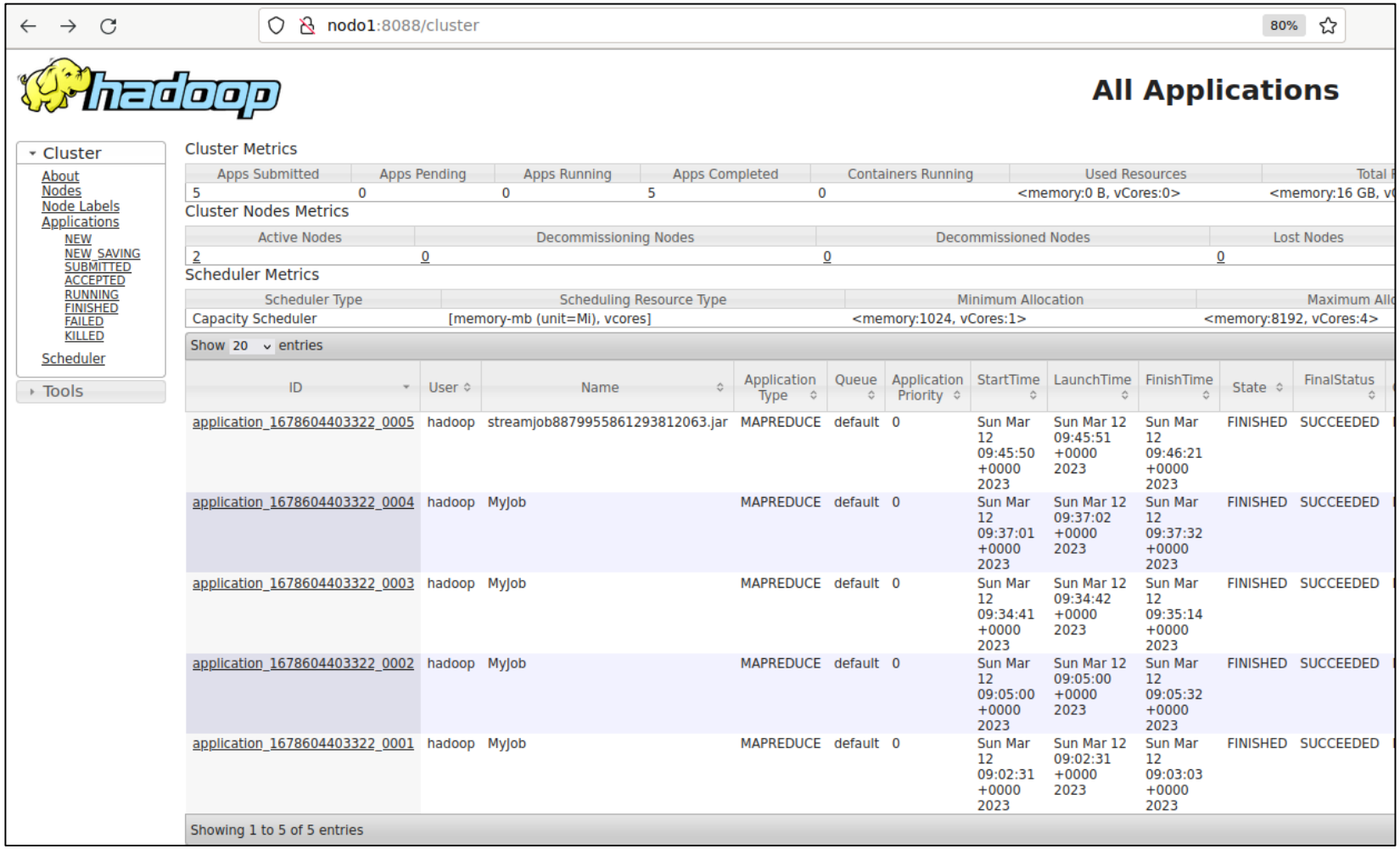
Paso 1. Lanzaremos un proceso en streaming con comandos Shell de Linux. Vamos a usar comandos de la Shell de Linux para hacer de Mapper y Reducer, simulando el programa java anterior. Lo hacemos con la librería de Streaming. Le pasamos el fichero y en el mapper extraemos solo los datos del campo 2 con el comando "cut" y eliminamos duplicados con el reducer y el comando "uniq".

hadoop jar /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar -input /practicas/cite75_99.txt -output /resultado8 -mapper 'cut -f 2 -d ,' -reducer 'uniq'

```
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ hadoop jar hadoop-streaming-3.2.4.jar -input /practicas/cite75_99.txt
-output /resultado8 -mapper 'cut -f 2 -d,' -reducer 'uniq'
packageJobJar: [/tmp/hadoop-unjar1499805906784375966/] [] /tmp/streamjob8879955861293812063.jar tmpDir=null
2023-03-12 09:45:50,390 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-03-12 09:45:50,475 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-03-12 09:45:50,604 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging
/hadoop/.staging/job_1678604403322_0005
2023-03-12 09:45:50,719 INFO mapred.FileInputFormat: Total input files to process : 1
2023-03-12 09:45:50,727 INFO net.NetworkTopology: Adding a new node: /default-rack/192.168.0.102:9866
2023-03-12 09:45:50,727 INFO net.NetworkTopology: Adding a new node: /default-rack/192.168.0.103:9866
2023-03-12 09:45:50,761 INFO mapreduce.JobSubmitter: number of splits:2
2023-03-12 09:45:50,832 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1678604403322_0005
2023-03-12 09:45:50,833 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-12 09:45:50,917 INFO conf.Configuration: resource-types.xml not found
2023-03-12 09:45:50,917 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-03-12 09:45:50,947 INFO impl.YarnClientImpl: Submitted application application_1678604403322_0005
2023-03-12 09:45:50,976 INFO mapreduce.Job: The url to track the job: http://nodo1:8088/proxy/application_1678604403322
_0005/
2023-03-12 09:45:50,977 INFO mapreduce.Job: Running job: job_1678604403322_0005
2023-03-12 09:45:55,020 INFO mapreduce.Job: Job job_1678604403322_0005 running in uber mode : false
2023-03-12 09:45:55,021 INFO mapreduce.Job:  map 0% reduce 0%
2023-03-12 09:46:10,109 INFO mapreduce.Job:  map 86% reduce 0%
2023-03-12 09:46:11,119 INFO mapreduce.Job:  map 100% reduce 0%
```

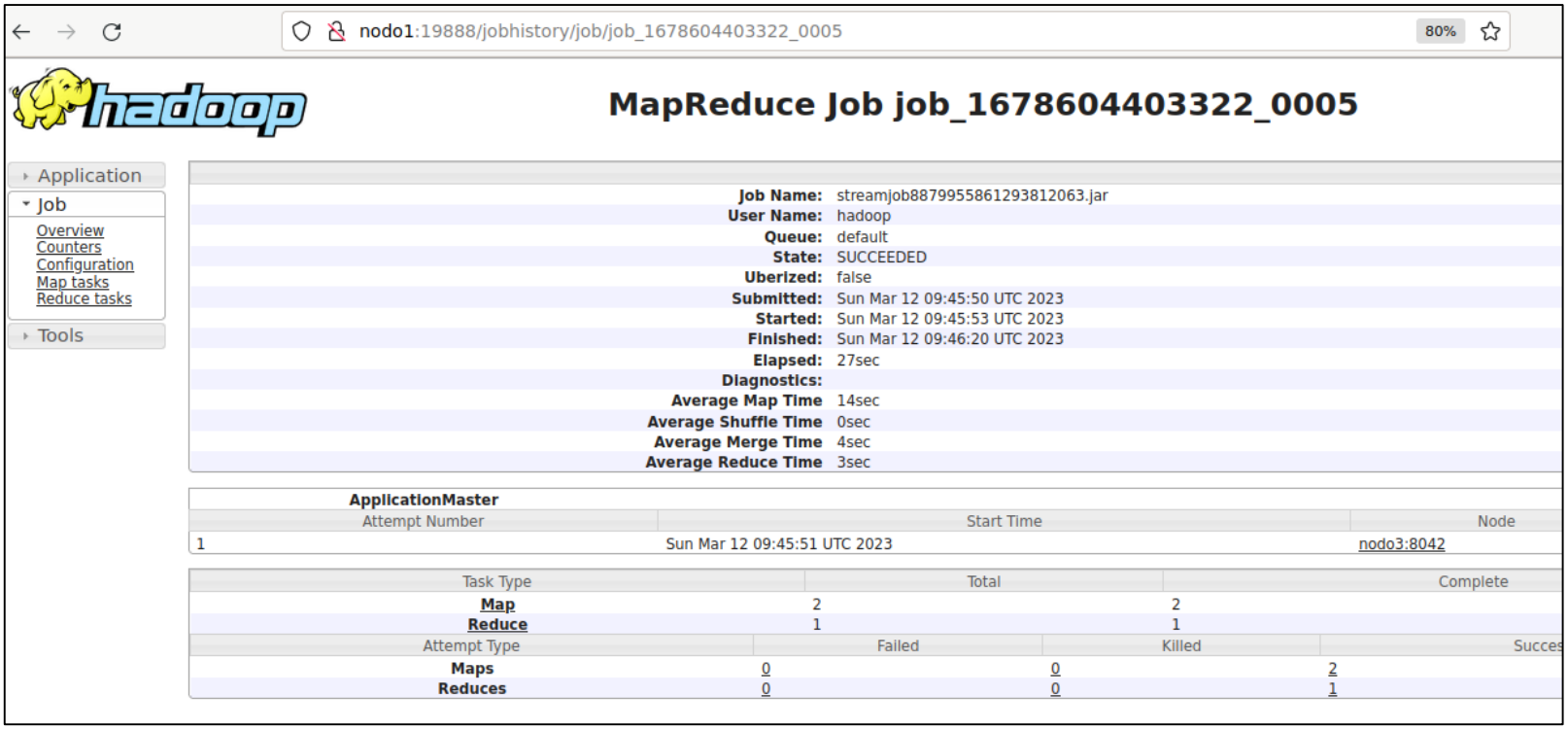
3. PRACTICA CON STREAMING

Paso 2. Podemos verlo en la WEB de administración



3. PRACTICA CON STREAMING

Paso 3. Si seleccionamos History, podemos ver el número de mappers y reducers



3. PRACTICA CON STREAMING

Paso 4. Si solo queremos contar ocurrencias, no necesitamos reducers, sólo hay mappers. Ejecutamos el mapper con el comando "wc -l"

hadoop jar /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar -D mapred.reduce.tasks=0 -input /practicas/cite75_99.txt -output /resultado9 -mapper 'wc -l'

```
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ hadoop jar hadoop-streaming-3.2.4.jar -D mapred.reduce.tasks=0
-input /practicas/cite75_99.txt -output /resultado9 -mapper 'wc -l'
packageJarJar: [/tmp/hadoop-unjar1684889347788076126/] [] /tmp/streamjob3661013380378792368.jar tmpDir=null
2023-03-12 10:02:41,041 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-03-12 10:02:41,131 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-03-12 10:02:41,248 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/
staging/hadoop/.staging/job_1678604403322_0006
2023-03-12 10:02:41,379 INFO mapred.FileInputFormat: Total input files to process : 1
2023-03-12 10:02:41,386 INFO net.NetworkTopology: Adding a new node: /default-rack/192.168.0.102:9866
2023-03-12 10:02:41,386 INFO net.NetworkTopology: Adding a new node: /default-rack/192.168.0.103:9866
2023-03-12 10:02:41,424 INFO mapreduce.JobSubmitter: number of splits:2
2023-03-12 10:02:41,441 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduc
e.job.reduces
2023-03-12 10:02:41,898 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1678604403322_0006
2023-03-12 10:02:41,899 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-12 10:02:41,980 INFO conf.Configuration: resource-types.xml not found
2023-03-12 10:02:41,980 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-03-12 10:02:42,014 INFO impl.YarnClientImpl: Submitted application application_1678604403322_0006
2023-03-12 10:02:42,036 INFO mapreduce.Job: The url to track the job: http://nodo1:8088/proxy/application_167860
4403322_0006/
2023-03-12 10:02:42,037 INFO mapreduce.Job: Running job: job_1678604403322_0006
2023-03-12 10:02:46,088 INFO mapreduce.Job: Job job_1678604403322_0006 running in uber mode : false
2023-03-12 10:02:46,088 INFO mapreduce.Job:  map 0% reduce 0%
2023-03-12 10:02:52,135 INFO mapreduce.Job:  map 100% reduce 0%
2023-03-12 10:02:52,143 INFO mapreduce.Job: Job job_1678604403322_0006 completed successfully
```

3. PRACTICA CON STREAMING

Paso 5. Podemos ver el resultado en el directorio /resultado9. Como le hemos quitado el reducer y tenemos dos bloques en el fichero (cada bloque es de 128Mb y nuestro fichero ocupa 200MB), nos aparecen dos ficheros con el número de líneas en cada bloque

hdfs dfs -ls /resultado9

```
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ hdfs dfs -ls /resultado9
Found 3 items
-rw-r--r--   2 hadoop supergroup          0 2023-03-12 10:02 /resultado9/_SUCCESS
-rw-r--r--   2 hadoop supergroup          9 2023-03-12 10:02 /resultado9/part-00000
-rw-r--r--   2 hadoop supergroup          9 2023-03-12 10:02 /resultado9/part-00001
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ hdfs dfs -cat /resultado9/part-00000
8258241
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ hdfs dfs -cat /resultado9/part-00001
8264198
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$
```

4. COMANDOS YARN GESTION CLUSTER

- El comando **yarn** permite gestionar, visualizar y comprobar las aplicaciones de tipo infraestructura yarn que se están ejecutando en el cluster.
- Permite la gestión de las aplicaciones de un clúster desde la consola del propio Linux
- Representa un sustitutivo de la gestión que realizamos via web
- Es útil en situaciones que no disponemos de un entorno gráfico para gestionar el cluster mediante un navegador web.
- El comando **yarn** tiene una serie de opciones, donde podemos:
  - arrancar el resourcemanager
  - arrancar el nodemanager
  - o las 3 opciones con las que podemos comprobar el resultado de nuestro entorno: **yarn application**, **yarn container** y **yarn node**

4. COMANDOS YARN GESTION CLUSTER

Comando yarn node

- Permite ver la información de los nodos del cluster.
- **yarn node –list** → Lista los nodos de nuestro clúster. Vemos node-ID (útil para el comando Yarn) y el nº de contenedores en uso

```
hadoop@nodo1:~$ yarn node -list
2023-03-12 07:00:17,117 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
     nodo2:37063                RUNNING        nodo2:8042                                  0
hadoop@nodo1:~$
```

- **yarn node –list –showDetails** → Lista los nodos con mayor detalle: recursos de memoria asignados, vcores, contenedores usados, etc

```
hadoop@nodo1:~$ yarn node -list -showDetails
2023-03-12 07:40:03,328 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-03-12 07:40:03,497 INFO conf.Configuration: resource-types.xml not found
2023-03-12 07:40:03,497 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
Detailed Node Information :
        Configured Resources : <memory:8192, vCores:8>
        Allocated Resources : <memory:0, vCores:0>
        Resource Utilization by Node : PMem:1204 MB, VMem:1204 MB, VCores:0.00996016
        Resource Utilization by Containers : PMem:0 MB, VMem:0 MB, VCores:0.0
        Node-Labels :
     nodo2:37063                RUNNING        nodo2:8042                                  0
Detailed Node Information :
        Configured Resources : <memory:8192, vCores:8>
        Allocated Resources : <memory:0, vCores:0>
        Resource Utilization by Node : PMem:1193 MB, VMem:1193 MB, VCores:0.049983338
        Resource Utilization by Containers : PMem:0 MB, VMem:0 MB, VCores:0.0
        Node-Labels :
hadoop@nodo1:~$
```

4. COMANDOS YARN GESTION CLUSTER

Comando yarn application

- Da información sobre las aplicaciones que tenemos ejecutando dentro del clúster, con una serie de opciones.
- **yarn application –list** → Lista las aplicaciones en ejecución.

```
hadoop@nodo1:~$ yarn application -list
2023-03-12 08:07:37,944 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total number of applications (application-types: [], states: [SUBMITTED, ACCEPTED, RUNNING] and tags: []):0
                Application-Id      Application-Name        Application-Type          User           Queue
      Progress                       Tracking-URL
hadoop@nodo1:~$
```

- **yarn application –status <applicationID>** → Vemos los detalles de la aplicación lanzada: estado, URL tracker, recursos asociados, etc
- **yarn application -kill <applicationID>** → Permite parar una aplicación que todavía esta en ejecución. Indicada para aplicaciones que no funcionan bien, o que tardan mucho tiempo.
- **yarn applicationattempt –list <applicationID>** → Lista intentos de lanzar los containers (mappers o reducers que pueden fallar)

```
hadoop@nodo1:~$ yarn applicationattempt -list application_1678604403322_0007

2023-03-12 11:36:42,728 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total number of application attempts :1
         ApplicationAttempt-Id               State                     AM-Container-Id
                 Tracking-URL
appattempt_1678604403322_0007_000001               FINISHED    container_1678604403322_0007_01_000001
 http://nodo1:8088/proxy/application_1678604403322_0007/
hadoop@nodo1:~$
```

4. COMANDOS YARN GESTION CLUSTER

Comando yarn container

yarn container → Permite ver información sobre los distintos contenedores que han ejecutado las aplicaciones

yarn container –list <appattemptID> → Da información de los contenedores que están asociados: container ID, información del nodo donde se ejecutan, la url de log para saber qué es lo que ha pasado. Si ya ha finalizado no va a encontrar nada.

4. COMANDOS YARN GESTION CLUSTER

Ejemplo con comandos yarn

Lanzamos una aplicación, con **yarn application –list** cogemos su applicationID, y lo utilizaremos con **yarn application -status**

```
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ hadoop jar hadoop-streaming-3.2.4.jar -input /practicas/cite75_
99.txt -output /resultado10 -mapper 'cut -f 2 -d,' -reducer 'uniq'
```

```
hadoop@nodo1:~$ yarn application -list
2023-03-12 10:34:35,639 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total number of applications (application-types: [], states: [SUBMITTED, ACCEPTED, RUNNING] and tags: []):1
                Application-Id      Application-Name        Application-Type          User           Queue
           State              Final-State            Progress                        Tracking-URL
application_1678604403322_0007  streamjob5943815195833610861.jar                   MAPREDUCE        hadoop
  default                  RUNNING               UNDEFINED                   0%                  http://nodo2:36
661
hadoop@nodo1:~$ yarn application -status application_1678604403322_0007
2023-03-12 10:34:58,074 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-03-12 10:34:58,264 INFO conf.Configuration: resource-types.xml not found
2023-03-12 10:34:58,264 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
Application Report :
        Application-Id : application_1678604403322_0007
        Application-Name : streamjob5943815195833610861.jar
        Application-Type : MAPREDUCE
        User : hadoop
        Queue : default
        Application Priority : 0
        Start-Time : 1678617272933
        Finish-Time : 0
        Progress : 50%
        State : RUNNING
        Final-State : UNDEFINED
        Tracking-URL : http://nodo2:36661
        RPC Port : 33483
        AM Host : nodo2
        Aggregate Resource Allocation : 87997 MB-seconds, 60 vcore-seconds
        Aggregate Resource Preempted : 0 MB-seconds, 0 vcore-seconds
        Log Aggregation Status : DISABLED
        Diagnostics :
        Unmanaged Application : false
        Application Node Label Expression : <Not set>
        AM container Node Label Expression : <DEFAULT_PARTITION>
        TimeoutType : LIFETIME  ExpiryTime : UNLIMITED  RemainingTime : -1seconds
```

4. COMANDOS YARN GESTION CLUSTER

Ejemplo con comandos yarn

Con el comando **yarn node –list** vemos los contenedores que está utilizando esta aplicación. Se esta ejecutando en el nodo 2 donde ha lanzado 3 contenedores (2 mappers y un reduce)

```
hadoop@nodo1:~$ yarn node -list
2023-03-12 11:03:45,650 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
     nodo2:37063                RUNNING        nodo2:8042                                  3
hadoop@nodo1:~$ yarn node -list
2023-03-12 11:03:48,246 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
     nodo2:37063                RUNNING        nodo2:8042                                  1
hadoop@nodo1:~$ yarn node -list
2023-03-12 11:04:03,165 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
     nodo2:37063                RUNNING        nodo2:8042                                  1
hadoop@nodo1:~$ yarn node -list
2023-03-12 11:04:05,300 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
     nodo2:37063                RUNNING        nodo2:8042                                  0
hadoop@nodo1:~$
```

5. PRACTICA COMANDOS YARN

Paso 1. Para la realización de la practica con el comando Yarn es necesario mantener dos terminales abiertos

Paso 2. En el primero lanzamos la siguiente aplicación mapreduce:

hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.4.jar wordcount /practicas/cite75_99.txt /salida10

```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hadoop jar hadoop-mapreduce-examples-3.2.4.jar
 wordcount -D mapred.job.queuename=warehouse /practicas/cite75_99.txt /salida10
2023-03-12 14:00:10,509 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.1
01:8032
2023-03-12 14:00:10,699 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path:
/tmp/hadoop-yarn/staging/hadoop/.staging/job_1678604403322_0012
2023-03-12 14:00:10,818 INFO input.FileInputFormat: Total input files to process : 1
2023-03-12 14:00:10,868 INFO mapreduce.JobSubmitter: number of splits:2
2023-03-12 14:00:10,939 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_16786044033
22_0012
2023-03-12 14:00:10,940 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-12 14:00:11,022 INFO conf.Configuration: resource-types.xml not found
2023-03-12 14:00:11,022 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-03-12 14:00:11,053 INFO impl.YarnClientImpl: Submitted application application_16786044033
22_0012
2023-03-12 14:00:11,071 INFO mapreduce.Job: The url to track the job: http://nodo1:8088/proxy/a
pplication_1678604403322_0012/
2023-03-12 14:00:11,071 INFO mapreduce.Job: Running job: job_1678604403322_0012
2023-03-12 14:00:15,119 INFO mapreduce.Job: Job job_1678604403322_0012 running in uber mode : f
alse
```

Institut TIC de Barcelona

5. PRACTICA COMANDOS YARN

Paso 3. En el segundo terminal comprobamos que se ha lanzado

yarn application -list

```
hadoop@nodo1:~$ yarn application -list
2023-03-12 14:15:24,205 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total number of applications (application-types: [], states: [SUBMITTED, ACCEPTED, RUNNING] and tags: []):1
                Application-Id      Application-Name        Application-Type          User           Queue
     Progress                         Tracking-URL
application_1678604403322_0017            word count                MAPREDUCE        hadoop         default
            5%                    http://nodo2:36401
```

Paso 4. Comprobamos en que nodos se está ejecutando

yarn node -list

```
hadoop@nodo1:~$ yarn node -list
2023-03-12 14:15:32,445 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  2
     nodo2:37063                RUNNING        nodo2:8042                                  1
hadoop@nodo1:~$ yarn node -list
2023-03-12 14:15:43,366 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  1
     nodo2:37063                RUNNING        nodo2:8042                                  1
hadoop@nodo1:~$ yarn node -list
2023-03-12 14:15:50,186 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
     nodo2:37063                RUNNING        nodo2:8042                                  1
hadoop@nodo1:~$ yarn node -list
2023-03-12 14:15:57,663 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
     nodo3:36983                RUNNING        nodo3:8042                                  0
     nodo2:37063                RUNNING        nodo2:8042                                  0
```

5. PRACTICA COMANDOS YARN

Paso 5. Comprobamos el estado completo de la aplicación

yarn application -status <application_ID>

Paso 6. Comprobamos los Application Attempts:

yarn applicationattempt -list <application_ID>

Paso 7. Vemos los contenedores asociados

yarn container -list <appattempt_ID>

Paso 8. Eliminamos la aplicación

yarn application -kill <application_ID>

Paso 9. Comprobamos que se ha eliminado

yarn application -list

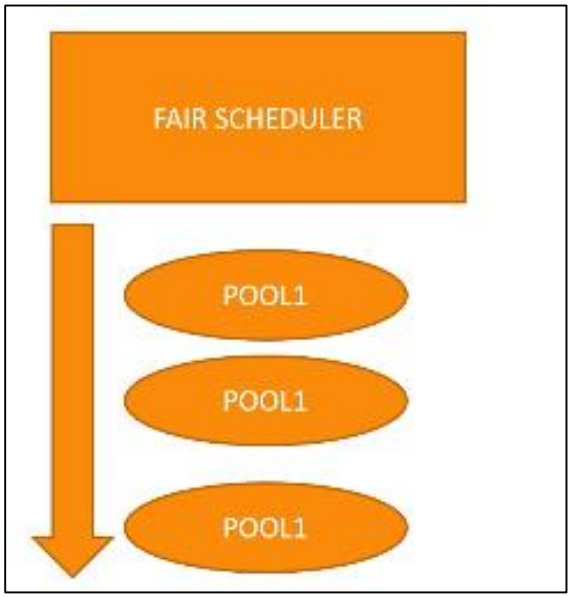
6. PLANIFICADORES EN HADOOP

- Hemos visto los procesos que se ejecutan en el Yarn cuando se ejecuta una aplicación: AppMaster, container, mappers y reducers
- El resorcemanager del maestro disponía de un recurso llamado el scheduler o planificador, componente básico dentro de Hadoop
- Determina los recursos que tienen cada uno de los componentes que van a formar parte del clúster.
- Tenemos dos tipos de planificadores:
  - Fair Scheduler → predefinido en la versión 1 de Hadoop
  - Capacity Scheduler → se utiliza en la versión 2 de Hadoop

6. PLANIFICADORES EN HADOOP

Fair Scheduler (v1 hadoop)

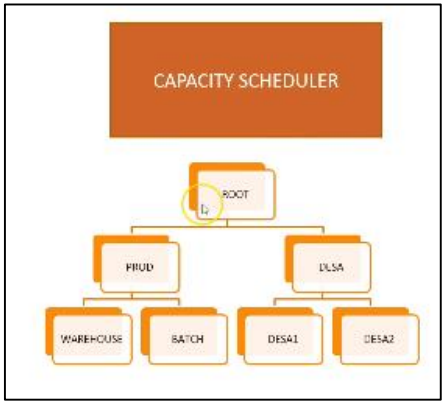
- Tenía un pull de recursos dependiendo del número de nodos, de la memoria, de la CPU, etc
- A todos los procesos que se lanzaban dentro del Fair Scheduler le asignaban recursos de manera homogénea dentro del Cluster
- Era un planificador muy homogéneo, sin darle demasiada prioridad a los recursos



6. PLANIFICADORES EN HADOOP

Capacity Scheduler (v2 hadoop)

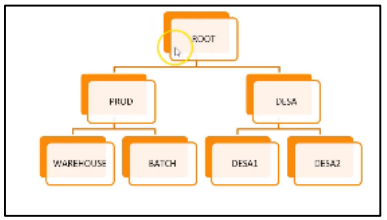
- Usa una estructura de jerarquías basada en colas, empezando por la cola primaria o root.
- Por defecto colgando de root tenemos una cola llamada default que está asociada al 100% de los recursos
- Cuando no esta configurado el Capacity Scheduler (por defecto) se comporta de una manera relativamente parecida al Fair Scheduler donde los recursos compiten por el 100% de los mismos
- El Capacity Scheduler tiene una arquitectura jerárquica que permite dividir mis recursos en distintas colas.



6. PLANIFICADORES EN HADOOP

Capacity Scheduler (v2 hadoop)

- El Capacity Scheduler se podría realizar la configuración anterior: Dos colas principales en el cluster, Producción y Desarrollo, donde dentro de cada una de ellos hay distintas ramas:
  - Producción usará el 70% de los recursos cluster
  - Desarrollo sólo el 30%
  - Dentro de producción, la cola warehouse ocupa el 80% de los recursos del padre y Batch el 20%
  - Los hijos dentro del Capacity heredan los recursos que se le han asignado al padre. Por ejemplo el recurso warehouse se llevaría el 70%*80% = 56% de los recursos del cluster
  - Si a los hijos Desa1 y Desa2 de Desa no se les indica nada, usan el 100% de los recursos asignados al padre, es decir utilizan el 100% del 30% que es el que se le ha dado a Desarrollo



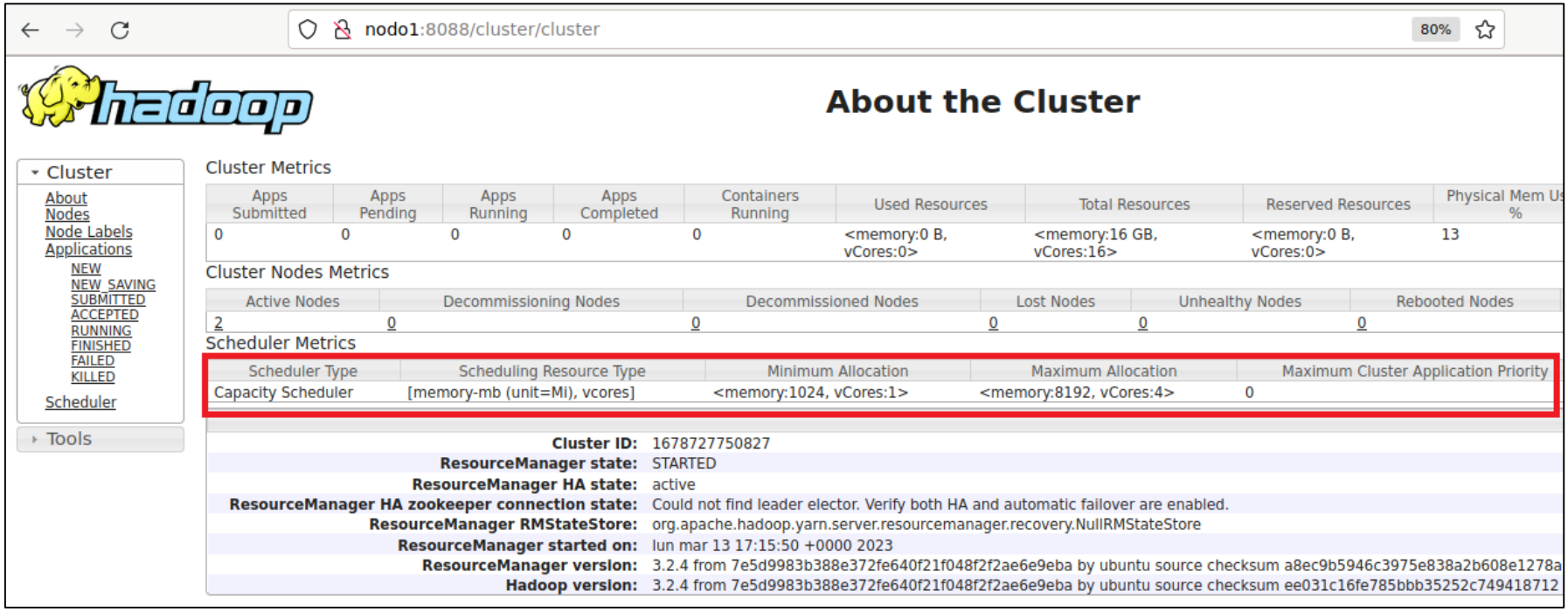
7. FUNCIONAMIENTO YARN SCHEDULER

IDEA

- La configuración del Capacity Scheduler del Yarn tiene sentido cuando en nuestro cluster lanzamos procesos muy heterogéneos:
  - Distintos en cuanto a funcionalidad
  - Distintos en cuanto a criticidad o urgencia
- Podemos hacer que determinados recursos se ejecuten más rápido porque les damos más recursos/capacidad en el cluster a través de colas con mas capacidad
- En cambio a otros procesos de menor tipo le podemos decir que utilicen menos recursos, asignándoles colas con menor capacidad

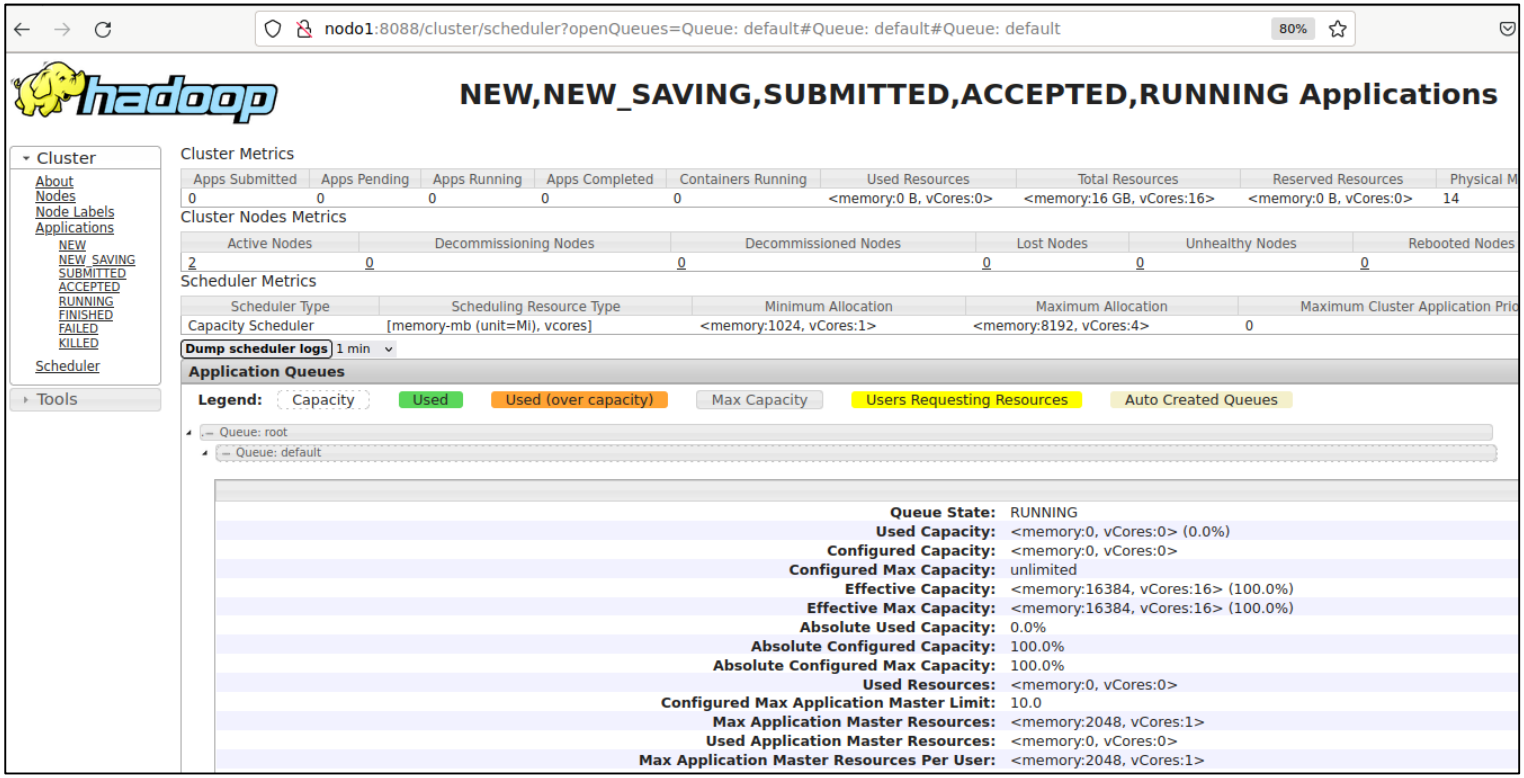
7. FUNCIONAMIENTO YARN SCHEDULER

Paso 1. En nodo1:8088 en la pestaña About the cluster, vemos que el Scheduler que estamos utilizando es el Capacity Scheduler, junto con una serie de propiedades generales como el mínimum Allocation o el máximo Allocation y la prioridad



7. FUNCIONAMIENTO YARN SCHEDULER

Paso 2. En el Menu Scheduler podemos ver las colas por defecto del Capacity Scheduler: La cola **root** con la que trabaja habitualmente y si no hemos cambiado nada, la cola **default**. Vemos que está configurada para el 100% de capacidad de recursos porque es la única existente. Hereda las características del padre que viene siempre con ese valor.

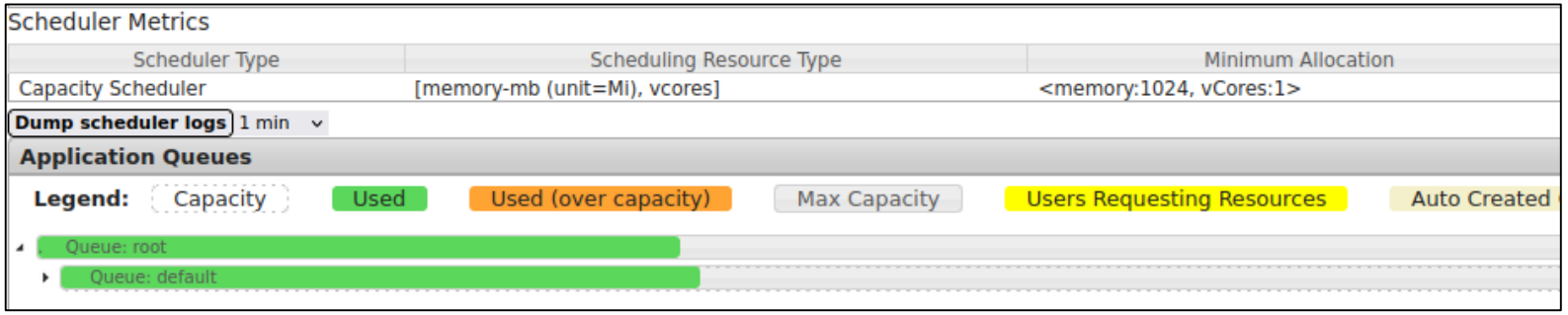


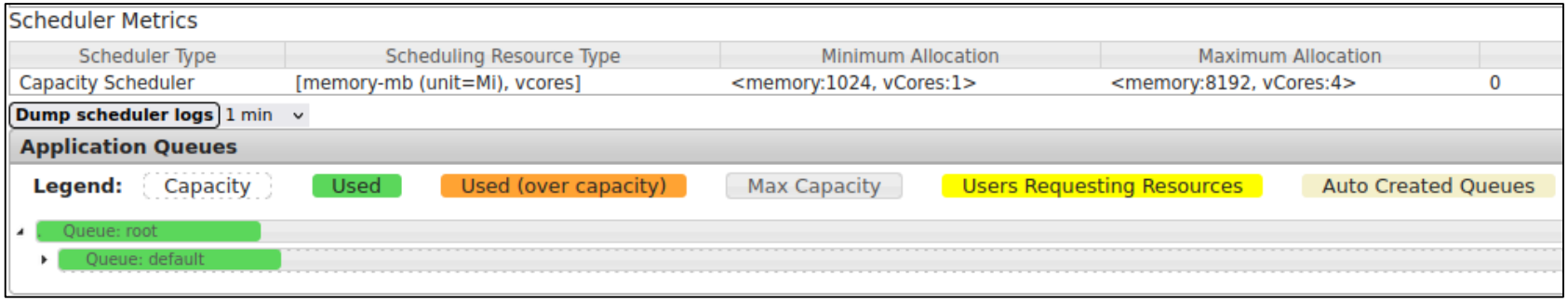
7. FUNCIONAMIENTO YARN SCHEDULER

Paso 3. Para ver el uso de nuestras colas de trabajo, lanzamos un job y veremos su uso y si está al límite de su capacidad.

```
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ hadoop jar hadoop-streaming-3.2.4.jar -input
/practicas/cite75_99.txt -output /resultado20 -mapper 'cut -f 2 -d,' -reducer 'uniq'
```

Si vamos refrescando la web, nos muestra una vista aceptable de la evolución de la capacidad de los recursos que se esta utilizando en cada una de las colas.





7. FUNCIONAMIENTO YARN SCHEDULER

Paso 4. Para ver información sobre el uso de nuestras colas de trabajo, desde la línea de comandos, podemos usar **mapred queue**, que muestra lo mismo que nos aparece a nivel de la visión gráfica

- **mapred queue –list** → indica la lista de las colas que tenemos y la información que se está ejecutando sobre ellas

```
1 hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ mapred queue -list
2023-03-14 07:50:06,954 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.
101:8032
======================
Queue Name : default
Queue State : running
Scheduling Info : Capacity: 100.0, MaximumCapacity: 100.0, CurrentCapacity: 0.0
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$
```

- **mapred queue –info default** → A nivel de línea de comandos va a salir información de la cola default similar a la pagina web

```
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$ mapred queue -info default
2023-03-14 07:58:12,857 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.
101:8032
======================
Queue Name : default
Queue State : running
Scheduling Info : Capacity: 100.0, MaximumCapacity: 100.0, CurrentCapacity: 0.0
hadoop@nodo1:/opt/hadoop/share/hadoop/tools/lib$
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 5. La configuración de las las colas de trabajo se realiza mediante el fichero /opt/hadoop/etc/hadoop/capacity-scheduler.xml. En este fichero se configura el comportamiento de nuestras colas y schedules

```
  GNU nano 6.2                /opt/hadoop/etc/hadoop/capacity-scheduler.xml
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>

  <property>
    <name>yarn.scheduler.capacity.maximum-applications</name>
    <value>10000</value>
    <description>
      Maximum number of applications that can be pending and running.
    </description>
  </property>
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 6. La primera property que tenemos que buscar y estudiar es **yarn.scheduler.capacity.root.queues** donde se indican las colas que tenemos por debajo de root. Por defecto solo está la cola default.

```
GNU nano 6.2                /opt/hadoop/etc/hadoop/capacity-scheduler.xml

<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>
```

Paso 7. Si quisiéramos tener dos colas más, una para aplicaciones de producción y otra para aplicaciones de desarrollo y otra para aplicaciones que no están etiquetadas (default) añadiríamos:

```
GNU nano 6.2                /opt/hadoop/etc/hadoop/capacity-scheduler.xml *

<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default,prod,desa</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 8. A continuación indicaremos la capacidad para cada cola. Tenemos que copiar dos veces mas la property de la capacidad de la cola default (que es del 100%) para las colas prod y desa.

Por ejemplo, podemos realizar la siguiente configuración de la capacidad de las colas: default→30%, prod→ 50%, desa→20%

Open

capacity-scheduler.xml
/opt/hadoop/etc/hadoop

```
54  <property>
55    <name>yarn.scheduler.capacity.root.default.capacity</name>
56    <value>30</value>
57    <description>Default queue target capacity.</description>
58  </property>
59
60  <property>
61    <name>yarn.scheduler.capacity.root.prod.capacity</name>
62    <value>50</value>
63    <description>Default queue target capacity.</description>
64  </property>
65
66  <property>
67    <name>yarn.scheduler.capacity.root.desa.capacity</name>
68    <value>20</value>
69    <description>Default queue target capacity.</description>
70  </property>
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 9. También se le pueden indicar muchas propiedades adicionales a las colas: la capacidad máxima, estado al arrancar (running o stopped), los permisos, distintas opciones de prioridad, numero de cores, etc. Pero lo importante es lo realizado en el punto anterior.

Open | capacity-scheduler.xml /opt/hadoop/etc/hadoop | Save

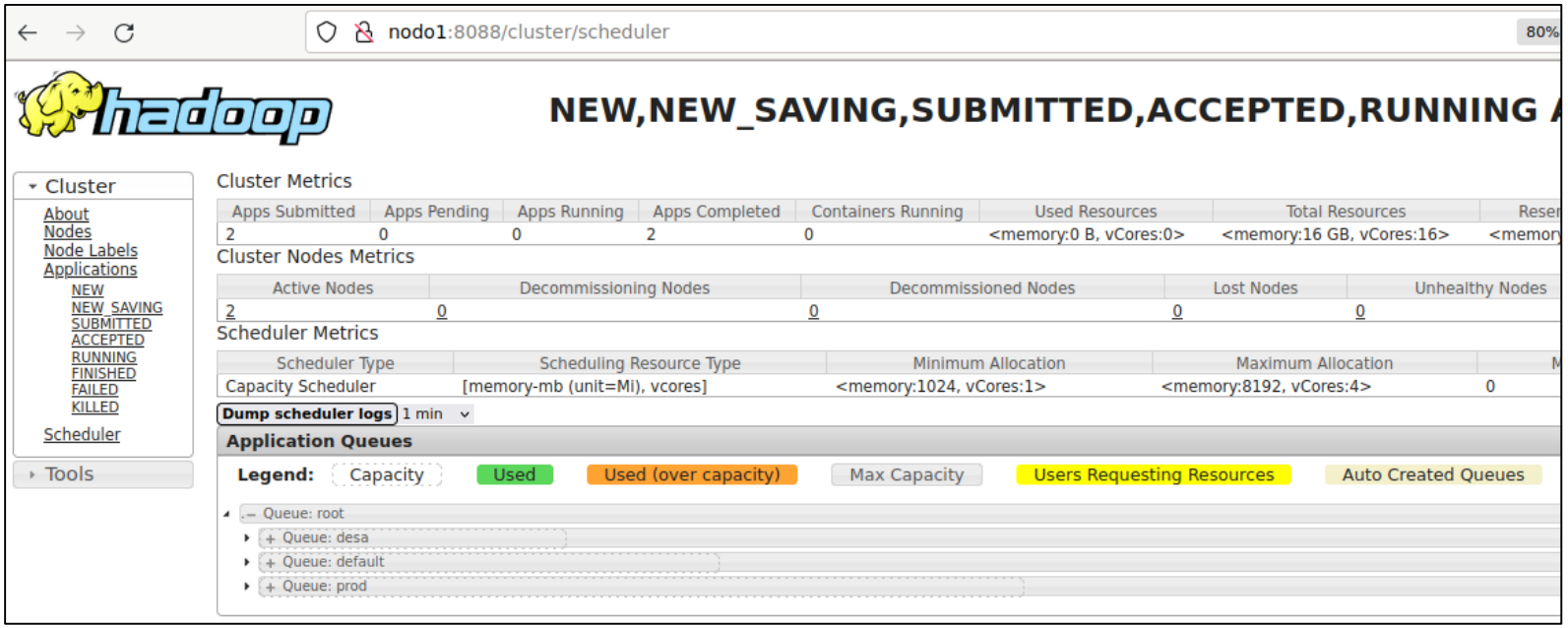
```
  <property>
    <name>yarn.scheduler.capacity.root.default.maximum-capacity</name>
    <value>100</value>
    <description>
      The maximum capacity of the default queue.
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.default.state</name>
    <value>RUNNING</value>
    <description>
      The state of the default queue. State can be one of RUNNING or STOPPED.
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.default.acl_submit_applications</name>
    <value>*</value>
    <description>
      The ACL of who can submit jobs to the default queue.
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.default.acl_administer_queue</name>
    <value>*</value>
    <description>
      The ACL of who can administer jobs on the default queue.
    </description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.default.acl_application_max_priority</name>
    <value>*</value>
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 10. **yarn rmadmin -refreshQueues** → Reconfigura las colas, activa la configuración anterior realizada en el capacity-scheduler.xml

```
255 hadoop@nodo1:~$ yarn rmadmin -refreshQueues
2023-03-14 09:44:10,406 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.
101:8033
hadoop@nodo1:~$
```

Paso 11. Si refrescamos el menú Scheduler de la web de gestión, nos muestra las colas configuradas: default, prod y desa



7. FUNCIONAMIENTO YARN SCHEDULER

Paso 12. Podemos tener jerarquías de jerarquías. Crearemos dos colas que cuelguen de Prod: waherhouse y batch.

Open

capacity-scheduler.xml
/opt/hadoop/etc/hadoop

```
46  <property>
47    <name>yarn.scheduler.capacity.root.queues</name>
48    <value>default,prod,desa</value>
49    <description>
50      The queues at the this level (root is the root queue).
51    </description>
52  </property>
53
54  <property>
55    <name>yarn.scheduler.capacity.root.prod.queues</name>
56    <value>warehouse,batch</value>
57    <description> The queues at the this level prod</description>
58  </property>
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 13. Ahora tenemos que dar la capacidad de las colas prod.warehouse (40%) y prod.batch (60%)

Open

capacity-scheduler.xml
/opt/hadoop/etc/hadoop

```
61  <property>
62    <name>yarn.scheduler.capacity.root.default.capacity</name>
63    <value>30</value>
64    <description>Default queue target capacity.</description>
65  </property>
66
67  <property>
68    <name>yarn.scheduler.capacity.root.prod.capacity</name>
69    <value>50</value>
70    <description>Prod queue target capacity.</description>
71  </property>
72
73  <property>
74    <name>yarn.scheduler.capacity.root.prod.warehouse.capacity</name>
75    <value>40</value>
76    <description>Prod queue target capacity.</description>
77  </property>
78
79  <property>
80    <name>yarn.scheduler.capacity.root.prod.batch.capacity</name>
81    <value>60</value>
82    <description>Prod queue target capacity.</description>
83  </property>
84
```

7. FUNCIONAMIENTO YARN SCHEDULER

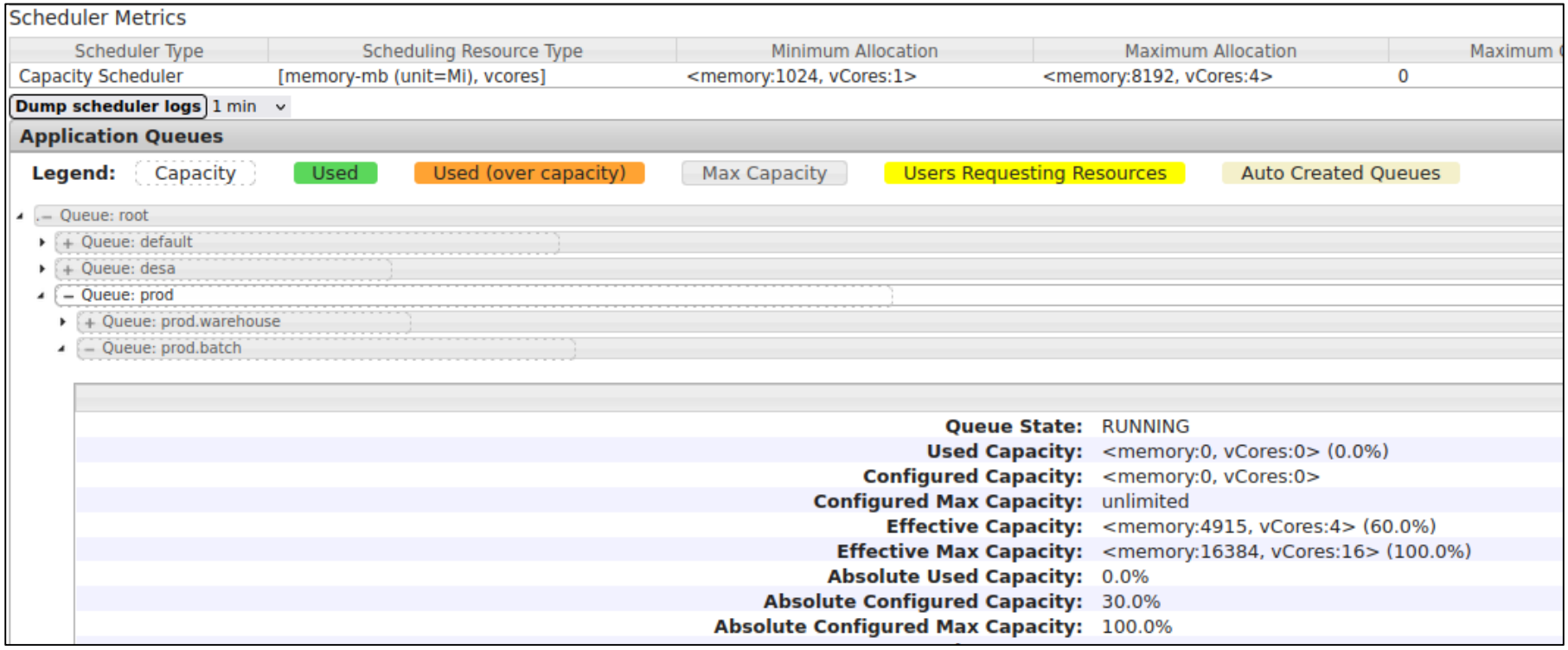
Paso 14. Si refrescamos de nuevo, vemos que no nos deja, ya que estamos intentando asignar una cola fija a la cola root.prod que ya esta activa y no lo permite. Hay que parar el cluster e iniciarlo

```
255 hadoop@nodo1:~$ yarn rmadmin -refreshQueues
2023-03-14 10:22:39,841 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8033
refreshQueues: java.io.IOException: Failed to re-init queues : Can not convert the leaf queue: root.prod to parent queue since it is not y
 Current State : RUNNING
        at org.apache.hadoop.yarn.ipc.RPCUtil.getRemoteException(RPCUtil.java:38)
        at org.apache.hadoop.yarn.server.resourcemanager.AdminService.logAndWrapException(AdminService.java:936)
        at org.apache.hadoop.yarn.server.resourcemanager.AdminService.refreshQueues(AdminService.java:414)
        at org.apache.hadoop.yarn.server.api.impl.pb.service.ResourceManagerAdministrationProtocolPBServiceImpl.refreshQueues(ResourceMana
tocolPBServiceImpl.java:120)
        at org.apache.hadoop.yarn.proto.ResourceManagerAdministrationProtocol$ResourceManagerAdministrationProtocolService$2.callBlockingM
rAdministrationProtocol.java:287)
        at org.apache.hadoop.ipc.ProtobufRpcEngine$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine.java:549)
        at org.apache.hadoop.ipc.ProtobufRpcEngine$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine.java:518)
        at org.apache.hadoop.ipc.RPC$Server.call(RPC.java:1086)
        at org.apache.hadoop.ipc.Server$RpcCall.run(Server.java:1035)
        at org.apache.hadoop.ipc.Server$RpcCall.run(Server.java:963)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.ipc.Server$Handler.run(Server.java:2960)
Caused by: java.io.IOException: Failed to re-init queues : Can not convert the leaf queue: root.prod to parent queue since it is not yet i
rent State : RUNNING
        at org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler.reinitialize(CapacityScheduler.java:479)
        at org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler.reinitialize(CapacityScheduler.java:507)
        at org.apache.hadoop.yarn.server.resourcemanager.AdminService.refreshQueues(AdminService.java:438)
        at org.apache.hadoop.yarn.server.resourcemanager.AdminService.refreshQueues(AdminService.java:409)
        ... 11 more
Caused by: java.io.IOException: Can not convert the leaf queue: root.prod to parent queue since it is not yet in stopped state. Current St
        at org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacitySchedulerConfigValidator.validateQueueHierarchy(Capaci
idator.java:189)
        at org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacitySchedulerQueueManager.reinitializeQueues(CapacitySched
a:180)
        at org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler.reinitializeQueues(CapacityScheduler.java:77
        at org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler.reinitialize(CapacityScheduler.java:474)
        ... 14 more
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 15. Reiniciamos el clúster y volvemos a refrescar la pagina web

¿Porque la capacidad máxima de la cola batch es del 30%?



Institut TIC de Barcelona

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 16. Vamos a hacer un ejemplo de como lanzar trabajos a una determinada cola de nuestro Scheduler de planificación. Una de las opciones mas directas consiste en añadir la opción –D al proceso:

-Dmapred.job.queue.name='nombre_cola'

Lanzaremos dos procesos contadores de palabras, uno sobre la cola warehouse y el otro sobre la cola desa

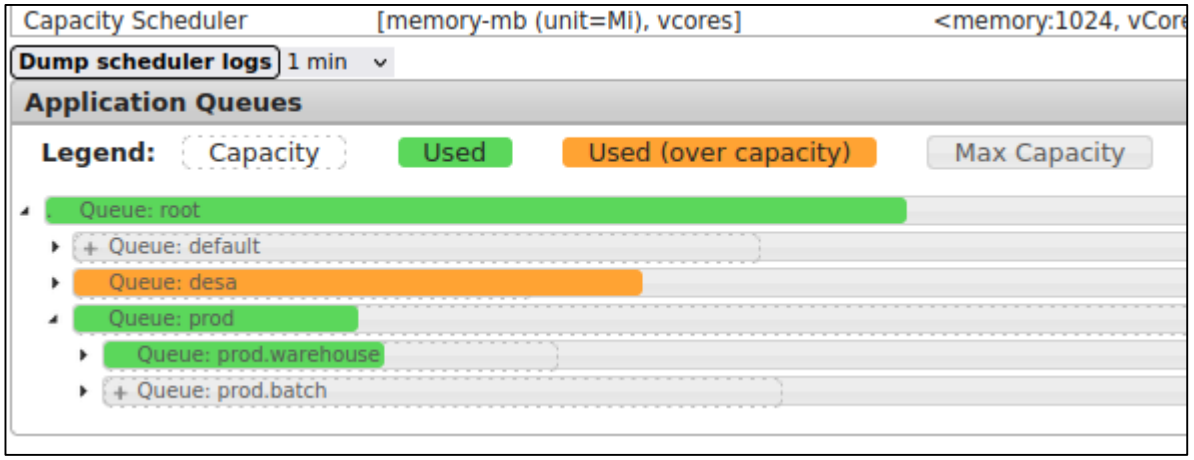
hadoop@nodo1 (192.168.0.101) - byobu × hadoop@nodo1: /opt/hadoop/share/hadoop/mapr... ×

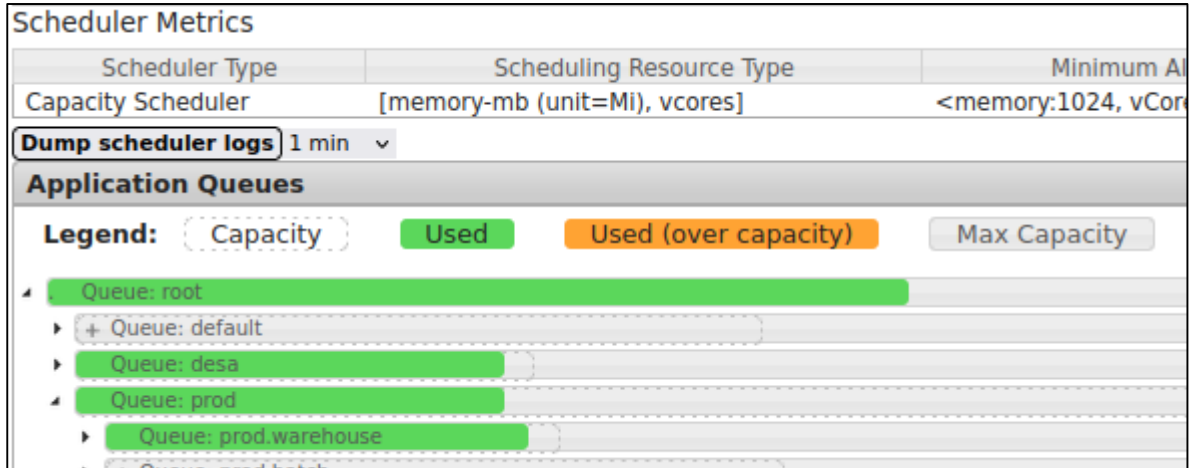
```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hadoop jar hadoop-mapreduce-examples-3.2.4.jar
wordcount -Dmapred.job.queue.name=warehouse /practicas/cite75_99.txt /salida30
2023-03-14 11:30:04,435 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.10
1:8032
2023-03-14 11:30:04,656 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /
tmp/hadoop-yarn/staging/hadoop/.staging/job_1678789833016_0007
2023-03-14 11:30:04,786 INFO input.FileInputFormat: Total input files to process : 1
2023-03-14 11:30:04,831 INFO mapreduce.JobSubmitter: number of splits:2
```

```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hadoop jar hadoop-mapreduce-examples-3.2.4.jar
wordcount -Dmapred.job.queue.name=desa /practicas/cite75_99.txt /salida31
2023-03-14 11:31:13,584 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.10
1:8032
2023-03-14 11:31:13,788 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /
tmp/hadoop-yarn/staging/hadoop/.staging/job_1678789833016_0008
2023-03-14 11:31:13,917 INFO input.FileInputFormat: Total input files to process : 1
2023-03-14 11:31:13,964 INFO mapreduce.JobSubmitter: number of splits:2
```

7. FUNCIONAMIENTO YARN SCHEDULER

Paso 17. Si hacemos un refresh podemos observar en cuanto las dos se pongan en marcha, la evolcuion de los recursos la cola desa y por otro la cola warehouse, en diferentes momentos





8. PRACTICA YARN SCHEDULER

Paso 1. Vamos a configurar tres colas para nuestro planificador de capacidad

NOMBRE	CAPACIDAD
default	10%
rrhh	50%
marketing	20%
ventas	20%

Paso 2. Abrimos el fichero /opt/hadoop/etc/hadoop/capacity-scheduler.xml y creamos las colas en las siguientes property:

```
<property>
        <name>yarn.scheduler.capacity.root.queues</name>
        <value>default,rrhh,marketing,ventas</value>
        <description>The queues at the level root</description>
</property>
```

8. PRACTICA YARN SCHEDULER

Paso 3. Ahora creamos una propiedad con la capacidad para cada una de las queues que hemos creado.

```
<property>
    <name>yarn.scheduler.capacity.root.default.capacity</name>
    <value>10</value>
    <description>Default queue target capacity</description>
</property>

<property>
    <name>yarn.scheduler.capacity.root.rrhh.capacity</name>
    <value>50</value>
    <description>Default queue target capacity</description>
</property>
```

8. PRACTICA YARN SCHEDULER

```
<property>
      <name>yarn.scheduler.capacity.root.marketing.capacity</name>
      <value>20</value>
      <description>Default queue target capacity</description>
</property>

<property>
      <name>yarn.scheduler.capacity.root.ventas.capacity</name>
      <value>20</value>
      <description>Default queue target capacity</description>
</property>
```

8. PRACTICA YARN SCHEDULER

Paso 4. Refrescamos las colas

yarn rmadmin –refreshQueues

Paso 5. Comprobamos con el comando mapred que están activas

mapred queue -list

```
18/01/07 20:43:18 INFO client.RMProxy: Connecting to ResourceManager at
nodo1/192.168.56.101:8032
 ======================
Queue Name : marketing
Queue State : running
Scheduling Info : Capacity: 20.0, MaximumCapacity: 100.0, CurrentCapacity: 0.0
 ======================
Queue Name : default
Queue State : running
Scheduling Info : Capacity: 10.0, MaximumCapacity: 100.0, CurrentCapacity: 0.0
======================
Queue Name : rrhh
Queue State : running
Scheduling Info : Capacity: 50.0, MaximumCapacity: 100.0, CurrentCapacity: 0.0
======================
Queue Name : ventas
Queue State : running
Scheduling Info : Capacity: 20.0, MaximumCapacity: 100.0, CurrentCapacity: 0.0
```

8. PRACTICA YARN SCHEDULER

Paso 6. Comprobamos con la pagina web que están activas