



ARQUITECTURA DE SISTEMES BIG DATA PEL RECORD LINKAGE DE XARXES SOCIALS

TREBALL FINAL DE GRAU – Informe de progrés II



2018-2019

ALEJANDRO GARCIA CARBALLO

1423957

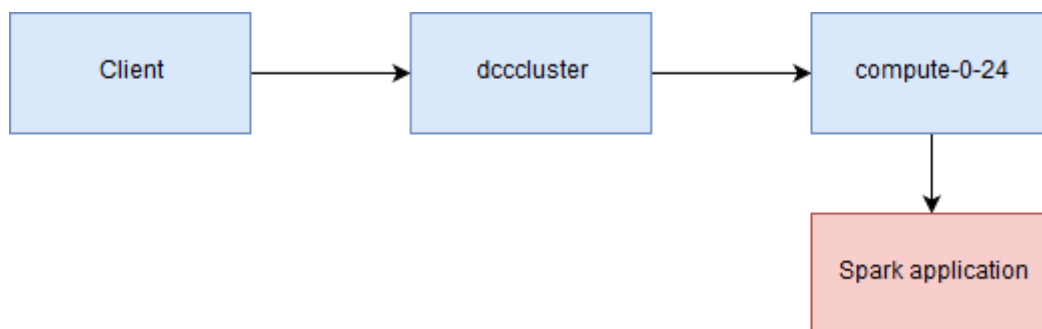
Índex:

1. Connexió client amb clúster.....	3
1.1. Objectiu.....	3
1.2. Creació script de prova	3
1.3. Redireccions de ports.....	4
1.4. Errors.....	5
1.5. Creació d'un contenidor client	7
2. Testing	10
2.1. Data tagging validation	10
2.2. Testing de rendiment	12
3. Seguiment de la planificació	14
4. Metodologia seguida durant el projecte.....	15
5. Resultats obtinguts	16
6. Conclusions	17
7. Bibliografia	18

1 Connexió client amb clúster:

1.1 Objectiu

Per a poder realitzar una interfície de dades el primer pas és aconseguir que el client sigui capaç d'executar el seu codi en mode clúster, és a dir, ha de poder connectar-se al clúster i mitjançant la configuració especificada en els fitxers de configuració llançar una aplicació Spark [6] amb YARN.



Il·lustració 1- Objectiu a aconseguir

1.2 Creació script de prova

Per realitzar proves de connexió i comprovar si el client estava connectant-se de manera satisfactòria amb el clúster, s'ha creat un script que realitza un recompte de lletres.

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
from operator import add

conf = SparkConf().setAppName("CharCount").setMaster("yarn")
sc = SparkContext(conf=conf)
data = sc.parallelize(list("Hello World"))
counts = data.map(lambda x: (x, 1)).reduceByKey(add).sortBy(lambda x: x[1], ascending=False).collect()
for (word, count) in counts:
    print("{}: {}".format(word, count))
sc.stop()
```

Il·lustració 2 - Script de prova per comprovar connexió

El output d'aquesta funció mostra per a cadascuna de les lletres el nombre de vegades que apareix:

```
2019-05-20 15:42:05 INFO DAGScheduler:54 - Job 2 finished: collect at /test.py:8, took 1.699409 s
l: 3
o: 2
H: 1
: 1
e: 1
d: 1
r: 1
W: 1
```

Il·lustració 3 - Output resultant de l'execució del script

1.3 Redireccions de ports

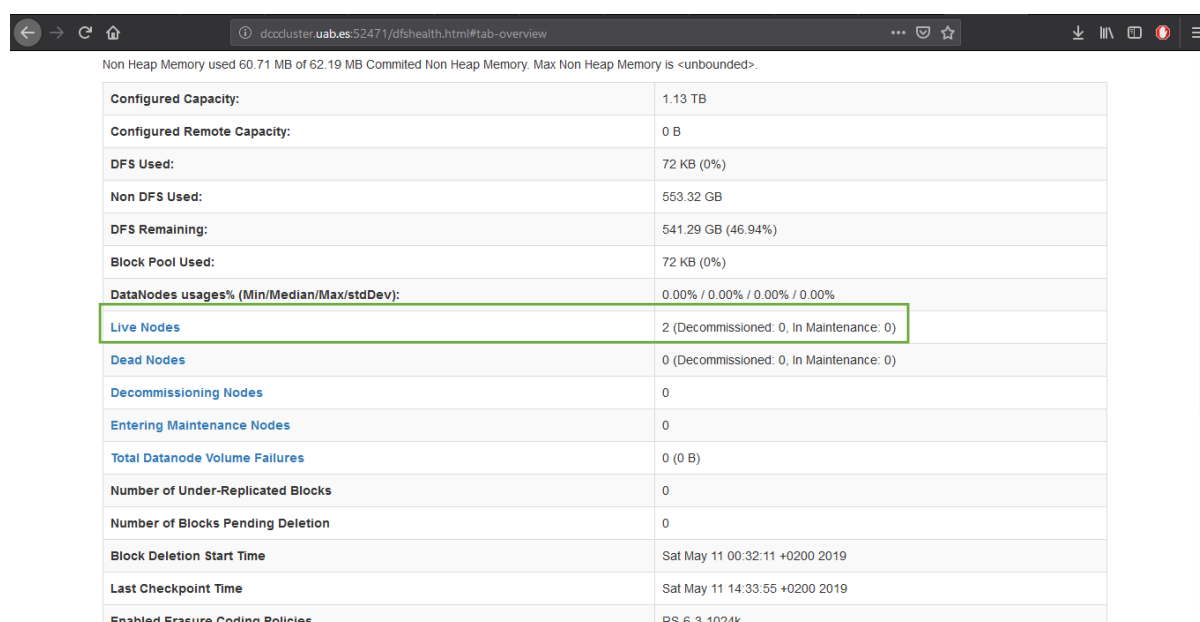
Per aconseguir realitzar aquesta execució s'han de realitzar un conjunt de redireccions en els ports [1] [3]. El rang de ports que poden ser utilitzats treballant en el node compute-0-24 de dcccluster és des de el port 52470 fins al 52480. La redirecció de ports s'ha realitzat a través del script encarregat d'executar les comandes docker per aixecar els containers especificant una redirecció a través del paràmetre -p de docker run. Per exemple:

```
docker run -dti --net hadoop --net-alias node2 -h node2 -p 52478:9866 --name node2 datanode-tfg
```

La redirecció de ports que s'ha realitzat és la següent:

Nom configuració	Port original	Redirecció del port
fs.default.name	9000	52475
yarn.resourcemanager.address	8032	52476
yarn.resourcemanager.webapp.address	8088	52472
yarn.nodemanager.webapp.address	8042	52479
dfs.datanode.address (node1)	9866	52479
dfs.datanode.address (node2)	9866	52478
spark.ui.port	4044	52473

Per comprovar que la redirecció de ports està funcionant correctament es pot accedir a les UI del namenode o yarn:



The screenshot shows the Hadoop Namenode Web UI. At the top, it displays memory usage: 'Non Heap Memory used 60.71 MB of 62.19 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.' Below this is a table of cluster metrics. The 'Live Nodes' row is highlighted with a green border and shows '2 (Decommissioned: 0, In Maintenance: 0)'. Other metrics include Configured Capacity (1.13 TB), Configured Remote Capacity (0 B), DFS Used (72 KB (0%)), Non DFS Used (553.32 GB), DFS Remaining (541.29 GB (46.94%)), Block Pool Used (72 KB (0%)), DataNodes usages% (0.00% / 0.00% / 0.00% / 0.00%), Dead Nodes (0), Decommissioning Nodes (0), Entering Maintenance Nodes (0), Total Datanode Volume Failures (0 (0 B)), Number of Under-Replicated Blocks (0), Number of Blocks Pending Deletion (0), Block Deletion Start Time (Sat May 11 00:32:11 +0200 2019), Last Checkpoint Time (Sat May 11 14:33:55 +0200 2019), and Enabled Erasure Coding Policies (RS-6-3-1024k).

Configured Capacity:	1.13 TB
Configured Remote Capacity:	0 B
DFS Used:	72 KB (0%)
Non DFS Used:	553.32 GB
DFS Remaining:	541.29 GB (46.94%)
Block Pool Used:	72 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	2 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	Sat May 11 00:32:11 +0200 2019
Last Checkpoint Time	Sat May 11 14:33:55 +0200 2019
Enabled Erasure Coding Policies	RS-6-3-1024k

Il·lustració 4 - Resum de la informació del namenode

Com es pot observar en el resum d'informació del namenode, els dos nodes estan funcionant (Live Nodes) i la redirecció de ports està realitzada d'acord als ports indicats en el resum dels datanodes:

Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
✓ node1:9866 (192.168.128.2:9866)	http://node1:9864	0s	2m	576.62 GB <div><div></div></div>	0	24 KB (0%)	3.2.0
✓ node2:9866 (192.168.128.3:9866)	http://node2:9864	0s	3m	576.62 GB <div><div></div></div>	0	24 KB (0%)	3.2.0

Showing 1 to 2 of 2 entries

Previous 1 Next

Il·lustració 5 - Datanodes i els ports que utilitzen

1.4 Errors

El següent pas era comprovar el funcionament de l'arquitectura a l'hora d'executar un script en python desde fora de la virtual LAN. S'han realitzat proves executant un script de mostra que està inclòs per defecte en spark (anomenat pi.py). Primerament les execucions es van realitzar amb spark-submit ja que l'objectiu principal era testear si l'execució en mode clúster funcionava. Per realitzar l'execució és necessari indicar a Spark on són els fitxers de configuració per tal de poder realitzar l'execució amb la configuració establerta. Això es pot aconseguir creant dues variables d'entorn anomenades HADOOP_CONF_DIR i YARN_CONF_DIR que apunten al directori que conté els fitxers de configuració (hdfs-site.xml, core-site.xml ...). Executant la següent comanda de spark submit s'inicia l'execució:

```
./spark-submit --master yarn --deploy-mode cluster ../examples/src/main/python/pi.py
```

El primer error que va surgir va ser el següent:

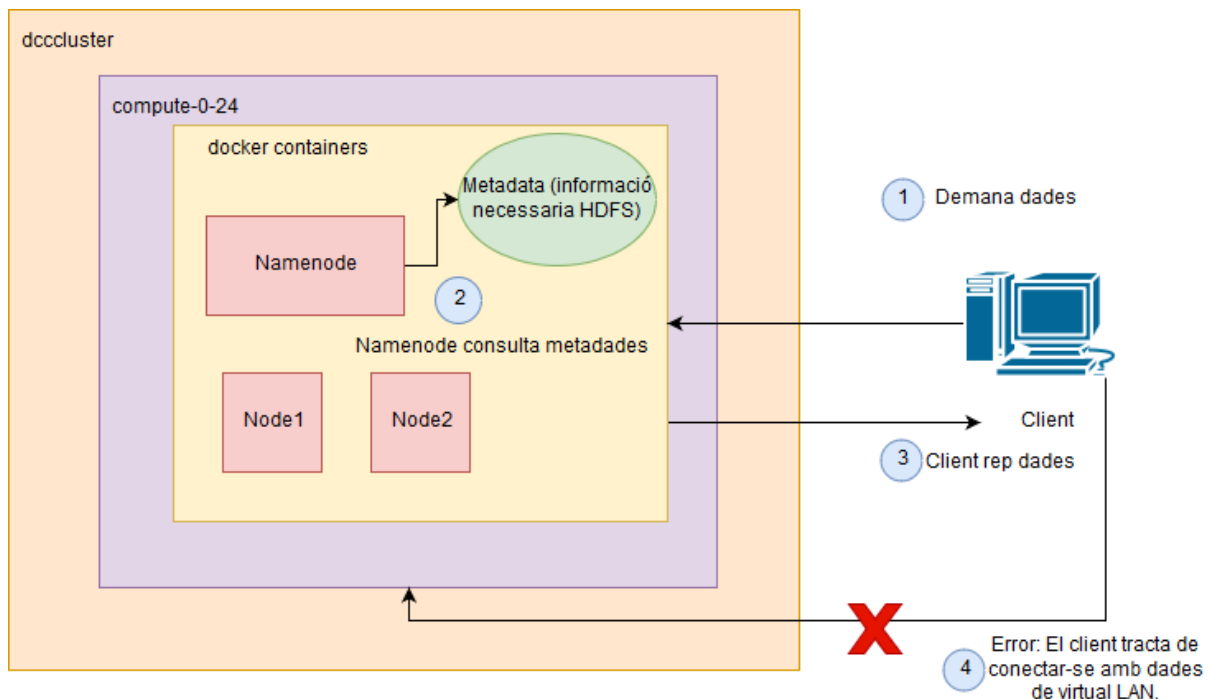
DFSClient: Exception in createBlockOutputStream

java.net.ConnectException: Connection timed out: no further information

Exception in thread "main" org.apache.hadoop.ipc.RemoteException(java.io.IOException):
File

/user/alexg/.sparkStaging/application_1557225967013_0013/___spark_libs___5441966897994847380.zip could only be written to 0 of the 1 minReplication nodes. There are 2 datanode(s) running and 2 node(s) are excluded in this operation.

Aquest error indica que no s'han pogut realitzar les rèpliques a l'hora de realitzar el staging de spark (procés en el qual l'aplicació es puja al HDFS). El motiu d'aquest error és molt ampli ja que no hi ha una causa única per la qual pugui aparèixer, per tant, trobar la seva solució pot arribar a ser un procés complicat. Inicialment pensava que es podia tractar d'un problema de permisos en els directoris del HDFS, així que els vaig modificar, però l'error persistia. També vaig realitzar un expose d'alguns ports que no estaven exposats fins al moment per assegurar que no s'estigués evitant la connexió a causa d'això. Realitzant una amplia cerca [4] [10] vaig trobar un article [2] que comentava que el possible error es que el client està intentant connectar-se als datanodes del clúster a través de la IP privada d'aquests:



Il·lustració 6 - Descripció del problema a l'hora d'executar codi en mode clúster

Com podem veure en la il·lustració 6, a través de les redireccions establertes anteriorment per poder fer possible la comunicació, el client demana al namenode la informació pertanyent als datanodes. El namenode no es comunica directament amb els datanodes, sinó que retorna les dades que emmagatzema en les metadades al client per a que aquest s'encarregui de realitzar la comunicació. Per tant, quan el client rep les dades, està rebent les dades pertanyents a la virtual LAN i al tractar de comunicar-se amb aquestes dades sorgeix un error.

Realitzant una cerca, vaig trobar una possible solució [5] que consistia en afegir un paràmetre en la configuració del fitxer `hdfs-site`:

```
<property>
  <name>dfs.client.use.datanode.hostname</name>
  <value>true</value>
</property>
```

Il·lustració 7 - Propietat del `hdfs-site` per utilitzar el `hostname` dels datanodes des de el client

Per defecte, els clients HDFS es connecten als datanodes utilitzant l'adreça IP proporcionada pel namenode. Depenent de la configuració de la xarxa, els clients no poden aconseguir aquesta adreça IP. La solució és deixar que els clients realitzin la seva pròpia resolució DNS del `hostname` del datanode [5]. També es va provar d'afegir la següent propietat:

```

<property>
  <name>dfs.datanode.use.datanode.hostname</name>
  <value>true</value>
  <description>Whether datanodes should use datanode hostnames when
    connecting to other datanodes for data transfer.
</description>
</property>

```

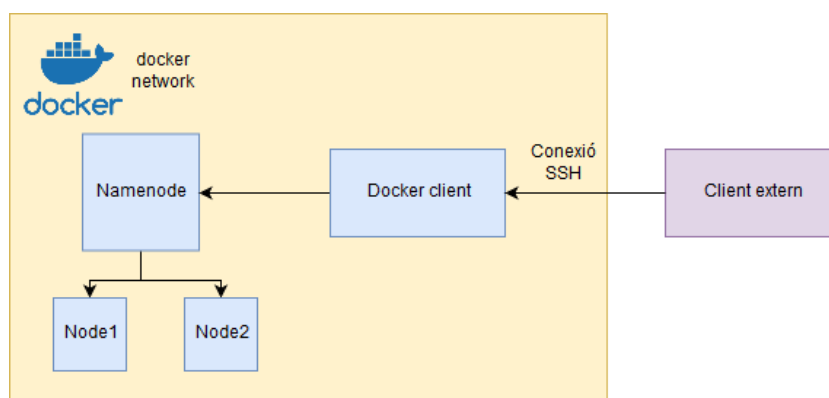
Il·lustració 8 - Propietat del hdfs-site per a que els datanodes es comuniquin entre ells a través del hostname

Rarament pot succeir que l'adreça IP resolta pel namenode per a un datanode sigui inaccessible des d'altres datanodes. La solució consisteix a obligar als datanodes a realitzar la seva pròpia resolució DNS per a connexions entre dades. El paràmetre mostrat a la il·lustració 8 habilita aquest comportament.

A continuació, es van afegir al fitxer de hosts els hostnames dels datanodes per tal de que puguin ser reconeguts. Tot i realitzar aquest conjunt de passos, l'error seguia apareixent, per tant, es van haver de buscar altres solucions.

1.5 Creació d'un contenidor client

La solució a aquest problema, tenint en compte que el projecte havia de continuar i una gran quantitat del temps d'aquesta fita s'ha dedicat a entendre per què la connexió estava fallant, ha sigut aixecar un container que actuï com a client dins de la virtual LAN de Docker i d'aquesta manera poder realitzar les execucions.



Il·lustració 9 - Solució proposada per a solucionar el problema

Aquest container té la configuració necessària de SSH i les redireccions necessàries per a poder ser accedit desde qualsevol client i executar els scripts directament desde el client de Docker. Per aconseguir-ho, s'han realitzat algunes modificacions en el Dockerfile de Spark, per a generar el que actuarà com a container Client. En el Dockerfile s'han afegit les següents línies per instal·lar i configurar el servidor SSH [7]:

```
#creating SSH server
RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/ssh
RUN echo 'root:testtest' | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/ssh_config

# SSH Login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/ssh

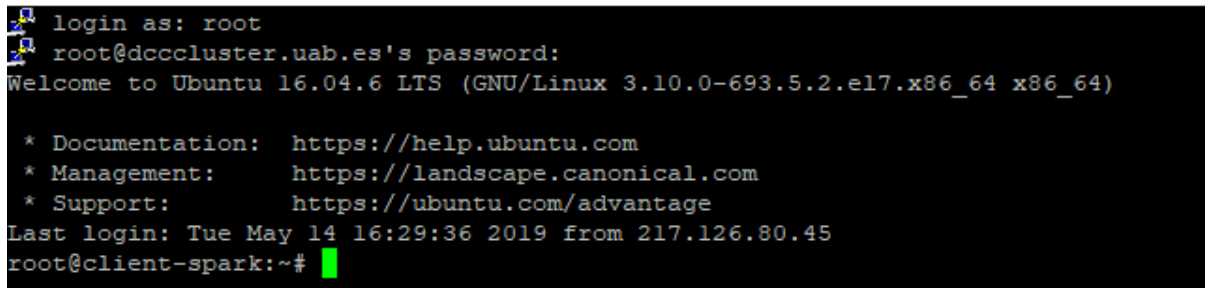
ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

#copy spark config
ADD configs/spark-defaults.conf $SPARK_HOME/conf/

# expose various ports
EXPOSE 4044 22
```

Il·lustració 10 - Línies afegides al Dockerfile del client per a que pugui ser accessible amb SSH

Aquesta imatge és molt similar a la de Spark, ja que conté Spark per a poder realitzar spark-submit, però conté les línies afegides que es poden observar en la il·lustració 10, per tal d'instal·lar un servidor SSH al contenidor i poder accedir-hi. Executant aquesta imatge es crea un contenidor amb Spark al qual podem accedir amb l'usuari de SSH 'root' i la password 'testtest':

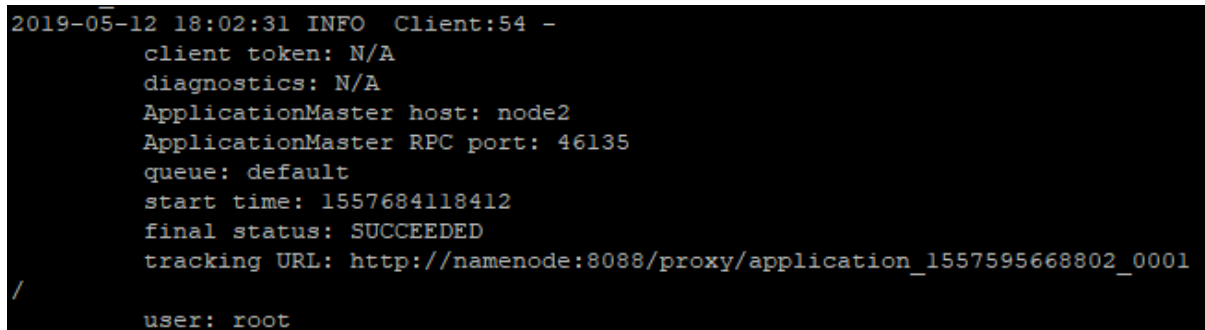


```
login as: root
root@dcccluster.uab.es's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 3.10.0-693.5.2.el7.x86_64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
Last login: Tue May 14 16:29:36 2019 from 217.126.80.45
root@client-spark:~#
```

Il·lustració 11 - Connexió SSH amb el contenidor del client

D'aquesta manera, al executar el spark-submit obtenim una execució exitosa:

```
2019-05-12 18:02:31 INFO Client:54 -
  client token: N/A
  diagnostics: N/A
  ApplicationMaster host: node2
  ApplicationMaster RPC port: 46135
  queue: default
  start time: 1557684118412
  final status: SUCCEEDED
  tracking URL: http://namenode:8088/proxy/application_1557595668802_0001
/
  user: root
```

Il·lustració 12 - Execució exitosa de script en el client

Si volem veure més detalls de l'execució podem accedir a la UI de YARN:


Show 20 entries														
ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores		
application_1557595668802_0001	root	pi.py	SPARK	default	0	Sun May 12 20:01:58 +0200 2019	Sun May 12 20:01:59 +0200 2019	Sun May 12 20:02:31 +0200 2019	FINISHED	SUCCEEDED	N/A	N/A	N	

Il·lustració 13 - Detalls de l'execució en la UI de YARN

On si seleccionem el nom de l'aplicació se'ns desplega una pantalla amb tots els detalls com ara el temps transcorregut, la data i hora d'inici d'execució, l'usuari que ha fet l'execució, etc. També és molt important tenir accés als logs de l'aplicació, ja que per depurar o veure en detall els resultats serà necessari. En aquest cas, l'execució s'ha realitzat en el node2 i una URL d'exemple per accedir al logs d'aquesta aplicació és:

http://dcccluster.uab.cat:52479/node/containerlogs/container_1557595668802_0002_01_00_0001/root/stdout/?start=-4096

Com es pot observar en la URL, s'han canviat els ports per defecte. Inicialment ens redirigia al port 8042, però amb la redirecció construïda amb Docker, haurem d'accedir al port 52479.



Logs for container_1557595668802_000

ResourceManager

RM Home

NodeManager

Tools

Showing 4096 bytes. Click [here](#) for full log

o missing parents

```

2019-05-12 18:12:31 INFO MemoryStore:54 - Block broadcast_0 stored as values in memory (estimated size 6.5 KB, free 366.3 MB)
2019-05-12 18:12:32 INFO BlockManagerInfo:54 - Added broadcast_0_piece0 in memory on node1:41733 (size: 4.3 KB, free: 366.3 MB)
2019-05-12 18:12:32 INFO SparkContext:54 - Created broadcast_0 from broadcast at DAGScheduler.scala:1161
2019-05-12 18:12:32 INFO DAGScheduler:54 - Submitting 2 missing tasks from ResultStage 0 (PythonRDD[1] at reduce at pi.py:44) (first 15 tasks are for partition
2019-05-12 18:12:32 INFO YarnClusterScheduler:54 - Adding task set 0.0 with 2 tasks
2019-05-12 18:12:32 INFO TaskSetManager:54 - Starting task 0.0 in stage 0.0 (TID 0, node2, executor 1, partition 0, PROCESS_LOCAL, 7841 bytes)
2019-05-12 18:12:32 INFO TaskSetManager:54 - Starting task 1.0 in stage 0.0 (TID 1, node1, executor 2, partition 1, PROCESS_LOCAL, 7841 bytes)
2019-05-12 18:12:32 INFO BlockManagerInfo:54 - Added broadcast_0_piece0 in memory on node1:34306 (size: 4.3 KB, free: 11.0 GB)
2019-05-12 18:12:32 INFO BlockManagerInfo:54 - Added broadcast_0_piece0 in memory on node2:35924 (size: 4.3 KB, free: 11.0 GB)
2019-05-12 18:12:34 INFO TaskSetManager:54 - Finished task 1.0 in stage 0.0 (TID 1) in 2079 ms on node1 (executor 2) (1/2)
2019-05-12 18:12:34 INFO TaskSetManager:54 - Finished task 0.0 in stage 0.0 (TID 0) in 2120 ms on node2 (executor 1) (2/2)
2019-05-12 18:12:34 INFO YarnClusterScheduler:54 - Removed TaskSet 0.0, whose tasks have all completed, from pool
2019-05-12 18:12:34 INFO PythonAccumulatorV2:54 - Connected to AccumulatorServer at host: 127.0.0.1 port: 57994
2019-05-12 18:12:34 INFO DAGScheduler:54 - ResultStage 0 (reduce at pi.py:44) finished in 2.364 s
2019-05-12 18:12:34 INFO DAGScheduler:54 - Job 0 finished: reduce at pi.py:44, took 2.448754 s
2019-05-12 18:12:34 INFO AbstractConnector:318 - Stopped Spark8355253c8[HTTP/1.1,[http/1.1]][0.0.0.0:0]
2019-05-12 18:12:34 INFO SparkUI:54 - Stopped Spark web UI at http://node1:42916
2019-05-12 18:12:34 INFO YarnAllocator:54 - Driver requested a total number of 0 executor(s).
2019-05-12 18:12:34 INFO YarnClusterSchedulerBackend:54 - Shutting down all executors
2019-05-12 18:12:34 INFO YarnSchedulerBackend$YarnDriverEndpoint:54 - Asking each executor to shut down
2019-05-12 18:12:34 INFO SchedulerExtensionServices:54 - Stopping SchedulerExtensionServices
(serviceOption=None,
services=List(),
started=false)
2019-05-12 18:12:34 INFO MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpoint stopped!

```

Il·lustració 14 - Logs de l'execució de l'aplicació

En la il·lustració 14 es pot observar un exemple de log d'aplicació, en aquest cas de l'aplicació encarregada de calcular el nombre pi. Podem observar com mostra el resultat de la seva execució, ja que al executar el script en mode clúster, els outputs es produeixen en els nodes en els que s'executen i no pas en el node client que llença l'aplicació i d'aquesta manera els outputs son visibles en els logs.

2 Testing

2.1 Data staging validation

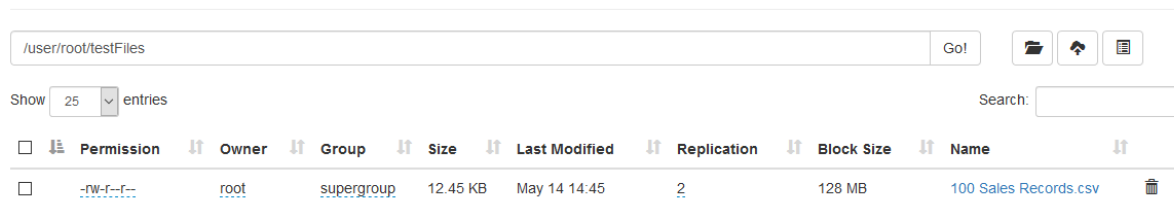
El data staging validation [8] [9] és una fase en la qual es validen els següents punts:

- Les diverses dades han de ser pujades al sistema de manera que totes les dades arribin de la font al destí correctament, evitant les duplicacions i respectant el nombre de rèpliques establertes en el replication factor.
- Comparar les dades pujades al HDFS amb les originals per tal d'assegurar que les dues coincideixen.
- Verificar que les dades correctes es situen en la localització del HDFS correcte.

```
root@namenode:/TestingDatasets# hdfs dfs -put 100%20Sales%20Records.csv testFiles
```

Al executar aquesta comanda s'indica que es vol pujar el fitxer .csv al directori testFiles. Com a resultat obtenim el següent:

Browse Directory



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	12.45 KB	May 14 14:45	2	128 MB	100 Sales Records.csv

Il·lustració 15 – Directori de fitxers HDFS

El fitxer es puja en el directori correcte. Si es selecciona el nom del fitxer, podem veure detalls com ara si s'ha realitzat una rèplica en els diferents nodes:



File information - 100 Sales Records.csv

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073741825
Block Pool ID: BP-1731354512-192.168.128.4-1557839554919
Generation Stamp: 1001
Size: 12744
Availability:

- node2
- node1

Il·lustració 16 - Disponibilitat del fitxer en els nodes

També podem comprovar que tal i com es va comentar a l'inici, el tamany de bloc és de 128 MB per a tots els fitxers:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	119.01 MB	May 15 23:09	2	128 MB	1000000SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	11.91 MB	May 15 23:09	2	128 MB	100000SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	1.19 MB	May 15 23:09	2	128 MB	10000SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	122.08 KB	May 15 23:09	2	128 MB	1000SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	12.45 KB	May 14 18:49	2	128 MB	100SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	178.51 MB	May 15 23:09	2	128 MB	1500000SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	59.51 MB	May 15 23:09	2	128 MB	500000SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	5.95 MB	May 15 23:09	2	128 MB	50000SalesRecords.csv	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	608.53 KB	May 15 23:09	2	128 MB	5000SalesRecords.csv	

Il·lustració 17 - Tamany de bloc

Com que la majoria dels fitxers no sobrepassen els 128 MB, les dades es guarden en un únic bloc, però en canvi, si el fitxer sobrepassa els 128 MB com és en el cas del fitxer amb 1500000 mostres, podem comprovar que es creen dos blocs:

File information - 1500000SalesRecords.csv

Download
Head the file (first 32K)
Tail the file (last 32K)

Block information --

Block 0
Block 0
Block 1

Block ID: 10731418
Block Pool ID: BP-660753791-192.168.128.4-1557849963263
Generation Stamp: 1035
Size: 13421728
Availability:

- node2
- node1

Il·lustració 18 - Creació de més d'un bloc quan es sobrepassen els 128 MB

Com es pot observar en la il·lustració 18 s'han realitzat les rèpliques correctament. El següent pas serà descarregar el fitxer i comparar-lo amb l'original per comprovar que no hi ha cap error en la descàrrega i que els fitxers no es vegin alterats:

```
C:\Users\alexg\Documents\UAB\Cuarto\TFG\TestingDatasets>fc 100SalesRecords.csv 100SalesRecordsHDFS.csv > output.txt
```

Fent ús de l'eina fc de Windows que s'encarrega de comparar dos fitxers i guardant el resultat en un .txt anomenat output es podrà comprovar si existeix alguna diferència entre els dos fitxers. Al obrir el fitxer dels resultats tenim el següent contingut:

```
Comparando archivos 100SalesRecords.csv y 100SALESRECORDSHDFS.CSV  
FC: no se han encontrado diferencias
```

Per tant, podem concloure que tant la pujada com la descàrrega de contingut al HDFS s'està duent a terme correctament.

2.2 Testing de rendiment

Hadoop està dissenyat per processar un gran volum de dades. Per tant, les proves arquitectòniques són crucials per garantir l'èxit d'un projecte Big Data. Un sistema mal dissenyat o inadequat pot provocar una degradació del rendiment i el sistema podria no complir el requisit. En un entorn Hadoop s'han de realitzar els serveis de proves de rendiment i de fallada.

Per realitzar les proves s'utilitzarà un conjunt de datasets [12] de diferents mides sobre registres de vendes. Els fitxers contenen desde 100 registres fins a un màxim de 1.500.000 registres i tenen un tamany de de 13 KB fins a 182 MB. Les proves s'han realitzat amb els diferents tamanyos de fitxers per poder comprovar com afecta el tamany dels fitxers i el nombre de registres i poder tractar de trobar a partir de quin límit es comença a notar la diferència en el temps d'execució.

Per realitzar les proves de temps de finalització de treballs s'executarà el següent script:

```
from pyspark.sql import SparkSession  
import time  
  
if __name__ == "__main__":  
    start = time.time()  
    spark = SparkSession.builder.appName("Testing Data Ingestion").getOrCreate()  
    df = spark.read.format("csv").option("header", "true").option("inferSchema", "true") \\\n        .load("hdfs:///user/root/testFiles/100SalesRecords.csv")  
    print(type(df))  
    df.printSchema()  
    print(df.rdd.map(lambda x: (1,x[1])).reduceByKey(lambda x,y: x + y).collect()[0][1])  
    print("Tiempo total:")  
    end = time.time()  
    print(end-start)
```

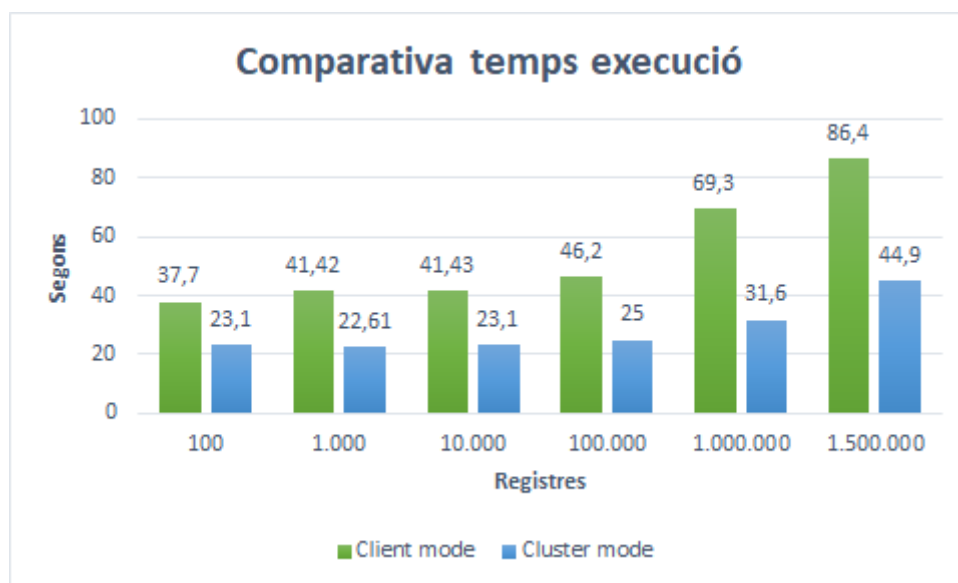
Il·lustració 19 - Script executat per a les proves de temps

El script mostrat en la il·lustració 17 s'encarrega de llegir els fitxers .csv i realitzar un sumatori de tots els elements d'una de les columnes amb un RDD (Resilient Distributed Dataset). Per realitzar la prova s'han utilitzat les funcions map i reduce per tal de tractar de paral·lelitzar el codi.

La següent taula mostra els diferents resultats obtinguts a l'hora d'executar el script, fent una comparativa del temps obtingut en mode client i el mode clúster.

	Temps	
Registres en el fitxer	Client mode	Clúster mode
100	37,7 sec	23,1 sec
1.000	41,4 sec	22,6 sec
10.000	41,4 sec	23,1 sec
100.000	46,2 sec	25,0 sec
1.000.000	69,3 sec	31,6 sec
1.500.000	86,4 sec	44,9 sec

D'aquesta taula podem extreure la següents gràfica comparativa:



II-lustració 20 - Gràfica comparativa de temps d'execució

Observant la gràfica podem ressaltar primerament que en tots els cassos l'execució en mode clúster obté millors resultats. Quan el número de registres no és molt gran, tot i obtenir uns millors resultats en mode clúster, la diferència no es tan gran. A mida que el nombre de registres augmenta, la reducció de temps en mode clúster en comparació amb l'execució en mode client es fa més notable, arribant a reduir el temps d'execució a pràcticament la meitat en el cas del fitxer que conté 1.500.000 registres. Per tant, el potencial d'aquest sistema és més rellevant quan tractem amb quantitats de dades grans que no pas en quantitats petites, on els resultats obtinguts no són tan bons en comparació amb els resultats obtinguts a l'hora d'utilitzar un conjunt de dades grans.

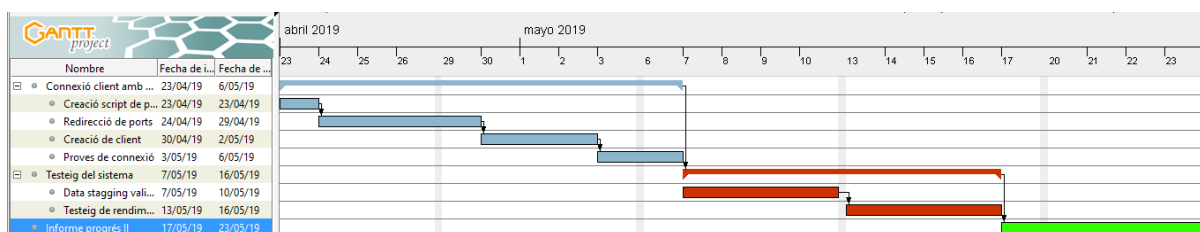
3 Seguiment de la planificació

Inicialment la planificació prevista per a realitzar en aquesta fita era la següent:



II·lustració 21 - Planificació inicial

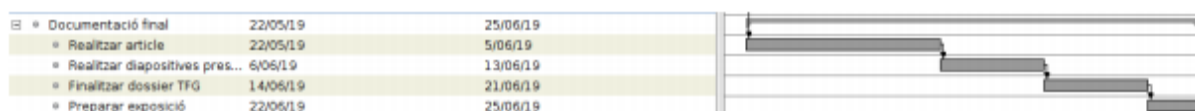
En aquesta planificació no hi era contemplada una fase que ha ocupat gran part del temps que és la fase de connexió del client amb el clúster.



II·lustració 22 - Re-planificació fita progrés II

Per tant, la planificació que he dut a terme finalment ha sigut la mostrada en la il·lustració 22, on m'he centrat en tancar el mòdul de la creació de l'arquitectura i el seu testeig i deixar les tasques de les interfícies de dades per a una possible futura implementació. En un inici tenia contemplada la idea de realitzar la primera interfície de dades, on s'havien de retornar les dades del HDFS sense harmonitzar, però finalment hi ha hagut un endarreriment en les tasques a causa d'un error a l'hora d'executar scripts en el clúster desde el client, degut també a que no tenia contemplada que aquesta fase pogués donar problemes. Aquest error ha provocat que les demés fases no es poguessin anar avançant i d'aquesta manera endarrerir el conjunt del projecte.

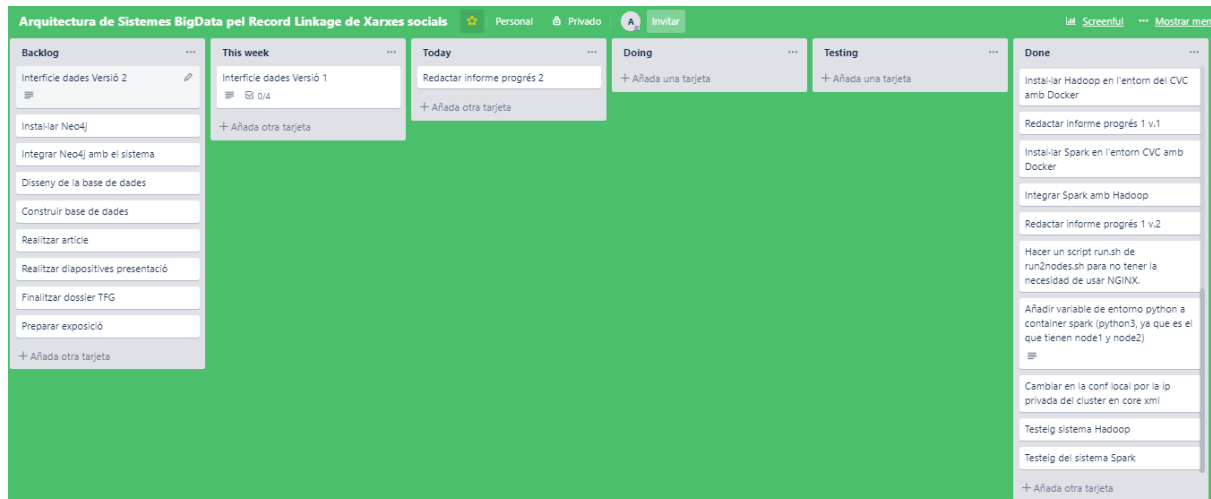
Com que aquesta era l'última fase de progrés, les següents tasques a realitzar seran aquelles que tenen relació amb la finalització de la documentació final.



II·lustració 23 - Planificació informe final

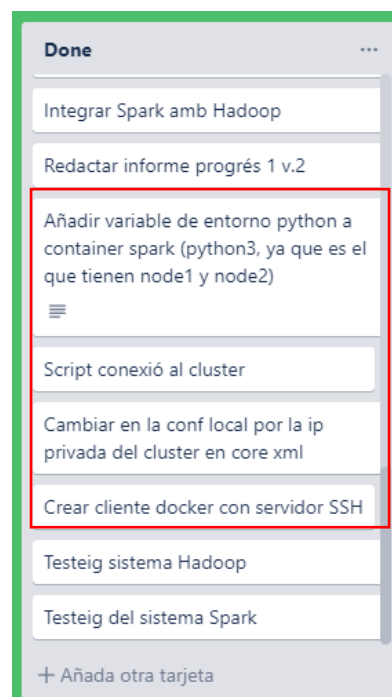
4 Metodologia seguida durant el projecte

Durant el projecte s'ha seguit la metodologia proposada a l'inici del projecte, una metodologia Kanban amb una visualització del flux de treball i unes fases per poder limitar el treball, on no calgués una gran quantitat de gestió ja que és un treball individual i, per tant, no tenia sentit realitzar reunions amb un equip de treball o plantejar algunes de les pràctiques comuns en altres metodologies.



Il·lustració 24 - Kanban desk amb l'eina de Trello

Gràcies a aquest tipus de metodologia també era fàcil adaptar-se a canvis o imprevistos, com ara el que ha sorgit en aquesta fita, on no es va contemplar com a necessari tenir en compte les tasques referents a la connexió de client i clúster. Només ha fet falta afegir al backlog les tasques que eren necessàries i limitar-les en el temps.



Il·lustració 25 – Algunes de les tasques afegides per a la connexió amb el clúster

5 Resultats obtinguts

Durant la realització d'aquest projecte s'ha aconseguit dur a terme l'arquitectura proposada en el disseny i els diferents diagrames, complint amb les configuracions que s'havien establert que havien de complir els sistemes Hadoop i Spark.

El sistema Hadoop està funcionant en el node compute-0-24 de dcccluster en una versió de Hadoop molt recent (3.2.0), adaptat a les necessitats que suposava la realització d'aquest projecte i sense funcionalitats o mòduls extra que no eren necessaris i que podien fer més pesat el conjunt de mòduls de l'arquitectura. Aquest sistema compleix les indicacions referents al tamany de blocs, replication factor, nombre de nodes i tot el conjunt de característiques mostrats en els diagrames. S'ha aconseguit fer funcionar YARN per a la negociació de recursos i poder separar MapReduce del HDFS per a utilitzar Spark com a tecnologia de processament de dades. Cadascun dels mòduls, tant el namenode com YARN, tenen la seva UI (User Interface) accessible en el navegador gràcies a les redireccions i configuració de la connexió establerta. En aquestes UI es poden veure detalls de cadascun dels mòduls d'una manera més accessible que a través de consola, detalls com ara els recursos utilitzats a l'hora d'executar les aplicacions de Spark, els logs per poder veure detalls de les execucions de les aplicacions Spark, o bé com estan distribuïts els fitxers a través del HDFS i poder veure si realment s'estan replicant els blocs i a la vegada, s'estan respectant les limitacions de tamany de bloc.

Referent a la part de Spark, s'ha aconseguit crear un mòdul funcional i integrat amb YARN, i també la realització d'un client que sigui capaç de connectar-se amb el clúster, de manera que aquest client sigui accessible a través d'una configuració d'un servidor SSH.

També s'ha realitzat un testeig tant de funcionament, com de validació de l'arquitectura i comprovació de que les dades eren correctes i es distribuïen de manera correcta per el sistema de Hadoop, així com un testeig de temps d'execució i comparació entre l'execució d'una aplicació Spark en mode client i en mode clúster.

6 Conclusions

La realització d'aquest projecte principalment tenia com a objectiu millorar la qualitat del projecte global de recerca XARXES, augmentant la seva eficiència i facilitant l'escalabilitat. Per aconseguir aquests objectius es plantejava com a solució la implementació d'un sistema big data.

Aconseguir part dels objectius ha comportat realitzar una gran recerca sobre les tecnologies utilitzades per a la implementació del sistema (Hadoop i Spark) i la utilització d'altres eines com ara Docker les quals inicialment tenia un coneixement molt bàsic, i gràcies a la realització d'aquest projecte he pogut aprofundir-hi més, posar-les en pràctica i aprendre a utilitzar-les.

Una part dels objectius principal s'han pogut dur a terme, aconseguint una arquitectura big data funcionant en el clúster de dcccluster. Encara que també una part planificada inicialment en el projecte referent a la realització d'una interfície de dades que facilités la utilització d'aquest sistema a través d'una llibreria no s'ha pogut dur a terme a causa dels imprevistos comentats en les seccions de planificació, reduint d'aquesta manera el temps que s'hauria d'haver utilitzat en aquest apartat. Tot i així, s'ha aconseguit tancar el mòdul pertanyent a la realització de l'arquitectura amb un sistema funcional.

La utilització d'una metodologia i la realització d'una planificació en un projecte d'una llargada considerable han sigut molt útils per poder anar contemplant quin seria l'abast del projecte, poder fer una re-planificació i establir uns objectius en un límit de temps. Gràcies a això i als conceptes que he pogut anar aprenent al llarg d'aquests anys de carrera, conjuntament amb tot allò que he après a l'hora de realitzar aquest projecte, ha estat possible realitzar aquest projecte.

7 Bibliografia

- [1] S. Lippens, «Hadoop 3 default ports», 16 Novembre 2018. [En línia]. Disponible a: <https://www.stefaanlippens.net/hadoop-3-default-ports.html>. [Últim accés: 10 Maig 2019].
- [2] Hadoop Team, «One Of Several Explanations To “could only be replicated to 0 nodes” Error», 17 Febrer 2014. [En línia]. Disponible a: <http://www.hadoopinrealworld.com/could-only-be-replicated-to-0-nodes/>. [Últim accés: 5 Maig 2019].
- [3] R. Tang, «Default Ports Used by Hadoop Services (HDFS, MapReduce, YARN)», 2017. [En línia]. Disponible a: <https://kontext.tech/docs/DataAndBusinessIntelligence/p/default-ports-used-by-hadoop-services-hdfs-mapreduce-yarn>. [Últim accés: 6 Maig 2019].
- [4] F. C. Ospina, «submit local spark job to emr», 8 Febrer 2019. [En línia]. Disponible a: <https://stackoverflow.com/questions/54598950/submit-local-spark-job-to-emr>. [Últim accés: 10 Maig 2019].
- [5] Apache Hadoop, «HDFS Support for Multihomed Networks», 17 Març 2017. [En línia]. Disponible a: <https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-hdfs/HdfsMultihoming.html>. [Últim accés: 11 Maig 2019].
- [6] Apache Spark, «Apache Spark™ is a unified analytics engine for large-scale data processing.», [En línia]. Disponible a: <https://spark.apache.org/>. [Últim accés: 2 Maig 2019].
- [7] Docker Team, «Dockerize an SSH service», [En línia]. Disponible a: https://docs.docker.com/engine/examples/running_ssh_service/. [Últim accés: 11 Maig 2019].
- [8] Guru99, «Big Data Testing Tutorial: What is, Strategy, How to test Hadoop», [En línia]. Disponible a: <https://www.guru99.com/big-data-testing-functional-performance.html>. [Últim accés: 13 Maig 2019].
- [9] Intellipaat, «Big Data Testing Tutorial | Big Data Hadoop Testing YouTube Video | Intellipaat», 15 Septembre 2017. [En línia]. Disponible a: https://www.youtube.com/watch?v=2FutPlcU_zA. [Últim accés: 13 Maig 2019].
- [10] Tariq, «Upload data to HDFS running in Amazon EC2 from local non-Hadoop Machine», 18 Abril 2013. [En línia]. Disponible a: <https://stackoverflow.com/questions/16073831/upload-data-to-hdfs-running-in-amazon-ec2-from-local-non-hadoop-machine>. [Últim accés: 5 Maig 2019].

[11] L. Gu and H. Li, "Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark," *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, Zhangjiajie, 2013, pp. 721-727.

doi: 10.1109/HPCC.and.EUC.2013.106

keywords: {cache storage;distributed processing;iterative methods;performance evaluation;iterative operation;Hadoop framework;Spark framework;data-intensive applications;data reloading;cache mechanism;distributed machines;memory consumption;Sparks;Generators;Computers;Distributed databases;Iterative methods;Central Processing Unit;Twitter;Hadoop;Spark;System Performance;Iterative Operation},

URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6831988&isnumber=68255>
16

[12] Eforexcel, «Downloads 18 - Sample CSV Files / Data Sets for Testing (till 1.5 Million Records) - Sales», 26 Agost 2017. [En línia]. Disponible a:

<http://eforexcel.com/wp/downloads-18-sample-csv-files-data-sets-for-testing-sales/>. [Últim accés: 14 Maig 2019]