Arquitectura de sistemes Big Data pel Record Linkage de xarxes socials

Alejandro García Carballo

Resum—Els documents demogràfics permeten estudiar un gran conjunt de comportaments en diferents àmbits com ara l'evolució social i econòmica del passat. Gràcies a les tècniques informàtiques de visió per computador es pretenen construir xarxes socials històriques aprofitant les dades demogràfiques de les que es disposen. Aquest treball es centra en la implementació d'una arquitectura Big Data on es puguin emmagatzemar les diferents dades demogràfiques, garantint un sistema escalable per tal de poder anar ampliant amb tants nodes com sigui necessari tant l'espai com la potència. Per aconseguir-ho, s'han escogit algunes de les tecnologies Big Data més utilitzades actualment i s'han tractat dades reals de diverses poblacions catalanes.

Paraules clau—Big Data, Arquitectura de Software, Hadoop, Spark, Dades demogràfiques, Sistema escalable, Record Linkage.

Abstract— Demographic documents allow us to study a large group of behaviors in different areas such as the social and economic evolution of the past. Thanks to computer-based computer vision techniques, it is intended to build historical social networks taking advantage of demographic data available. This work focuses on the implementation of a Big Data architecture where the different demographic data can be stored, guaranteeing a scalable system in order to be able to expand with as many nodes as it is necessary both space and power. To achieve this, some of the most widely used Big Data technologies have been chosen and real data from different Catalan populations have been dealt with.

Index Terms— Big Data, Software architecture, Hadoop, Spark, Demographic data, Scalable system, Record Linkage.

1 Introducció

Aquest treball està emmarcat dins del projecte de recerca XARXES [1], un projecte que s'està duent a terme en col·laboració amb el Centre de Visió per Computació (CVC) i el Centre d'Estudis Demogràfics (CED) de la UAB.

Les dades demogràfiques a part de ser útils per a entendre el comportament demogràfic tenen moltes altres utilitats com ara entendre l'evolució social o econòmica d'èpoques passades. Gràcies a l'avanç de les tècniques informàtiques de visió per computador es pretenen construir xarxes socials històriques aprofitant les dades demogràfiques de les que es disposen. Actualment es tenen dades de les poblacions catalanes de Sant Feliu de Llobregat, Begues, Castellví de Rosanes, Collbató, Corbera de Llobregat, El Papiol, Molins de Rei, Sant Vicenç de Castellet, Santa Coloma de Cervelló i Torrelles de Llobregat, però en un futur el nombre de poblacions pot augmentar i el sistema ha de ser capaç de suportar el possible creixement de dades. Les dades de les que es disposen actualment són alguns fitxers excels amb unes 25.000 entrades aproximadament i una altra part de la informació en MySQL, que conté dades dels habitants registrats en els padrons i en altres fonts de població de localitats veïnes.

Aquest projecte està pensat per a que sigui útil per a diferents usuaris, tant per a aquells usuaris que es dediquen a l'estudi de dades demogràfiques, històriques o qualsevol àmbit que pugui ser d'interès, com per a usuaris no experts, als quals se'ls pot oferir un servei didàctic més atractiu per tal d'apropar el patrimoni històric demogràfic de manera digital i facilitar també així el consum d'aquest coneixement.

Dins d'aquest gran projecte la meva feina s'ha centrat en la creació d'una arquitectura Big Data a través d'una arquitectura distribuïda. S'ha col·laborat amb Joana Maria Pujadas (del Centre d'Estudis Demogràfics) i amb un estudiant de màster encarregat del processament de les dades el qual podra utilizar el sistema Big Data implementat.

2 Estat de l'art

Per dur a terme aquest projecte s'han utilitzat algunes de les tecnologies més utilitzades actualment dins del àmbit del Big Data. En la fase inicial del projecte s'ha dedicat una part del temps a la investigació i a aprendre sobre aquest tipus de tecnologies i comparar diferents opcions possibles. Com a base de l'arquitectura s'ha utilitzat un sistema Hadoop, com a capa de programació distribuïda Spark i finalment com a sistema de base de dades estructurada en graf s'ha pensat utilitzar Neo4j.

Hadoop [2] és un framework d'Apache que permet el processament distribuït de grans conjunts de dades a través de clústers. Està dissenyat per a poder ser escalat des de servidors individuals fins a milers de màquines, cadascuna de les quals ofereix computació i almacenatge

locals. La llibreria està dissenyada per proporcionar un servei amb gran disponibilitat en els clústers, cadascun dels quals pot ser propens a fallades. Explicaré breument els components pels quals està format:

- HDFS (Hadoop Distributed File System): Emmagatzema grans quantitats de dades a través de múltiples màquines, replicant les dades a través de diferents hosts per aconseguir tolerància a falles. Les dades es divideixen en blocs de mida fixa que acostumen a ser 64 MiB, però el tamany d'aquests blocs es pot configurar. Com el mateix nom indica, es tracta d'un sistema de fitxers al igual que el sistema de fitxers d'un sistema operatiu, però a diferència d'aquests, aquest sistema de fitxers està pensat per a poder ser distribuït.
- YARN (Yet another Resource Negotiator): És una tecnologia de gestió de clústers que va sorgir en la segona versió de Hadoop. Amb YARN s'aconsegueix desacoblar la gestió de recursos als clústers del sistema de processament de dades com pot ser l'original de Hadoop MapReduce.
- MapReduce: És un mòdul de Hadoop basat en un model de programació per processar grans datasets en paral·lel i de manera distribuïda. Està basat en dos passos principals que són Map i Reduce.

Gràcies al desacoblament que s'aconsegueix amb YARN, s'ha utilitzat un altre sistema de processament de dades diferent de MapReduce anomenat Spark, ja que pot aconseguir executar un programa 100 vegades més ràpid en memòria o 10 vegades més ràpid en disc que utilitzant MapReduce, a part d'altres avantatges com ara la possibilitat d'utilitzar diversos llenguatges de programació i emmagatzemar les dades en memòria.

Com a base de dades estructurada en grafs s'ha pensat utilitzar Neo4j, ja que permet una integració simple entre la base de dades i Spark.

3 OBJECTIUS

La feina realitzada en aquest projecte ha consistit en aconseguir una arquitectura que permeti una escalabilitat horitzontal per tal de poder anar ampliant amb tants nodes com sigui necessari tant l'espai com la potència de còmput. A diferència d'una escalabilitat vertical on el creixement consisteix en la millora del hardware i per tant, pot arribar al seu límit amb facilitat, amb un sistema amb escalabilitat horitzontal es resol aquest problema. Gràcies a aquest tipus d'arquitectura es podran anar afegint les dades de més poblacions sense haver-nos de preocupar per arribar a un màxim de capacitat i aconseguir una gestió eficaç de les dades.

Tenint un context per entendre on entra aquest treball dins del projecte global de recerca XARXES, podem establir que els objectius d'aquest TFG son els següents:

- Millorar la qualitat del projecte global de recerca XARXES a través d'un sistema amb arquitectura Big Data.
- Augmentar l'eficiència del projecte XARXES.
- Facilitar l'escalabilitat de les dades.

Per tant, el projecte quedarà finalitzat quan quedi implementada l'arquitectura Big Data amb cadascun dels components connectats, funcionant entre ells i garantint una qualitat en el sistema final.

4 PLANIFICACIÓ

En aquest apartat explicaré les diferents fases per a realitzar la planificació del projecte.

4.1 Fases i tasques del projecte

Per aconseguir dur a terme els objectius i finalitzar el projecte correctament he realitzat un conjunt de tasques. Cada tasca pertany a una fase del projecte, el qual he dividit en sis fases generals:

- Primera fase: En aquesta fase s'ha realitzat una planificació general del projecte, una recerca inicial per informar-me de tot el necessari per a poder fer tant la planificació com poder escollir correctament les tecnologies que utilitzaré, i he escollit quina serà la metodologia que utilitzaré durant el projecte per dur-lo a terme. Finalment, com en totes les fases, he documentat tota la feina i he tractat de plasmar-la en la redacció de l'informe.
- Segona fase: El sistema Hadoop és la base que utilitzen la resta de components, per tant, aquest és el primer component que he implementat. He realitzat una instal·lació en les màquines de dccluster, s'ha configurat el HDFS i el YARN i s'ha testejat que la implantació s'ha fet correctament amb diferents tipus de tests. Tot el sistema implementat està en contenidors de docker.
- Tercera fase: Una vegada estava implementat el sistema Hadoop es va realitzar la implementació del sistema Spark, on s'ha hagut d'instal·lar i integrar-ho amb el Hadoop, testejant que la integració era correcte.
- Quarta fase: En aquesta fase he tractat de fer un mòdul en Python per retornar les dades que estan distribuïdes per el HDFS. La primera versió d'aquest mòdul es centra només en retornar les dades que demana l'usuari, però si hi havia temps suficient, es tractaria de fer una harmonització de les dades. Harmonitzar les dades consisteix en donar una versió actual i sense faltes ortogràfiques de les dades. Per exemple, alguns noms al llarg del temps poden evolucionar i canviar la manera en que s'escriuen o bé, han sigut anotats en les dades amb faltes ortogràfiques. A través d'un procés d'harmonització, es pot aconseguir el nom correcte. S'han de reservar els dos noms, tant l'harmonitzat com el que no ho està, ja que el sistema de visió per computador ha de detectar el que està escrit sense discriminar si es

correcte ortogràficament. Aquesta versió harmonitzada no es va implementar degut a la falta de temps, realitzant així la primera versió de la interfície.

- Cinquena fase: Aquesta fase tracta tot el relacionat amb les bases de dades, on s'ha d'integrar el sistema de bases de dades Neo4j, un sistema basat en graf, amb el sistema que vagi construint. S'ha de dissenyar la base de dades i construir-la, tenint en compte que les dades estan en formats diversos com pot ser fitxers excel o bases de dades MySQL. Aquesta fase finalment no es va realitzar, degut a la falta de temps.
- Sisena fase: Finalment, una última fase per tal d'acabar el dossier del TFG entregable, fer l'article i les diapositives de la presentació i preparar-me tot el necessari per a fer l'exposició.

5 METODOLOGIA

Aquest projecte és una feina individual, per tant, no és recomanable utilitzar metodologies com ara SCRUM, que està orientada a treballar amb equips. Fent una recerca dels diferents tipus de metodologies àgils que existeixen, vaig pensar que una de les més fàcils d'adaptar a una persona podria ser la metodologia XP (Extreme Programming), però algunes de les seves característiques implica treballar en equip o disposar de diferents rols i d'altres crec que no són molt útils per al tipus de treball que havia de realitzar. Al ser un treball individual tampoc havia de fer un excés de gestió, per tant, podia utilitzar alguna metodologia més simple que em permeti sobretot tenir clar quins son els objectius a realitzar en tot moment i tenir una manera visual de poder veure cadascuna de les tasques, limitant el treball a realitzar.

La metodologia Kanban [4] pot ser útil per a realitzar el TFG, ja que entre els seus principis tenim que:

1. Hem de visualitzar el flux del treball: un dels principis més bàsics de Kanban és la visualització del treball de manera senzilla.
2. Limitar el treball en procés: limitar la quantitat de treball ens permet fer una millor gestió de les tasques en un cert temps.
3. Centrar-se en el flux: una vegada tenim els dos primers principis en funcionament, hem de centrar la nostra atenció en qualsevol interrupció que es produeixi en el flux de treball. Això representa una oportunitat per una visualització addicional i millora d'un procés.
4. Millora continua: hem d'analitzar constantment possibles millores. Les condicions i demandes del client poden canviar al llarg del temps, així doncs, podem afegir millores al flux de treba-quan sigui necessari.

Fent una comparació amb les característiques d'una de les metodologies més famoses com és SCRUM, podem diferenciar els següents punts principals:

Kanban	Scrum
No hi ha rols precrits	Hi ha rols definits com ara el Scrum Master, el product owner o el membre de l'equip
Entrega continua	Sprint de temps parcial
Els canvis es poden produir en qualsevol moment	Els canvis no es poden produir a meitat d'un sprint.

Per a realitzar el Kanban, he utilitzat l'eina Trello [5], que permet visualitzar les tasques i gestionar-les a través de diferents columnes:

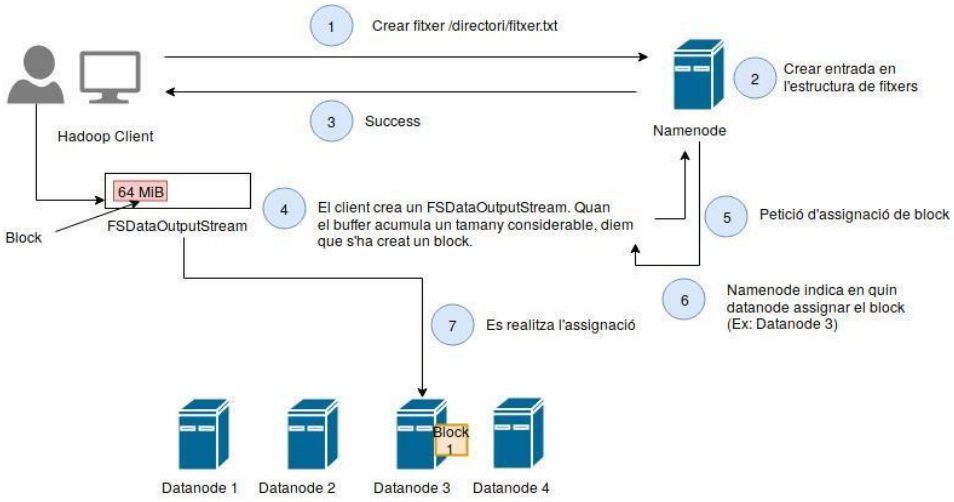
- Backlog: Conté les tasques que estan a la cua per a ser realitzades. D'aquest conjunt de tasques he anat agafant les que volia fer durant la setmana i les anava passant a la columna de This week.
- This week: En aquesta columna estan totes les tasques a realitzar durant la setmana. D'aquesta manera es fa una primera limitació de feina a realitzar.
- Today: Aquí he visualitzat quines eren les tasques que tenia previst realitzar durant el dia.
- Doing: Columna per visualitzar les tasques que estan en procés.
- Testing: Tota tasca que sigui testejable havia de passar per una fase de testeig.
- Done: Tasques que han sigut finalitzades. Això no implica que no es puguin realitzar millores en una tasca que s'ha contemplat com a Done.

6 DESENVOLUPAMENT

Abans de començar a especificar una arquitectura i d'explicar com he fet la implementació i configuració d'un node Hadoop, és important entendre cadascun dels mòduls que el composen. En aquest apartat s'explicarà quina és la funció de cadascun dels mòduls i com funcionen, i més endavant explicaré com configuraré cadascun d'aquests mòduls.

6.1 HDFS (HADOOP DISTRIBUTED FILE SYSTEM)

El HDFS [3] té una estructura de master/slave. El master del HDFS s'anomena namenode i s'encarrega de guardar l'estructura del sistema de fitxers i gestionar l'accés als fitxers demanats pel client. Acostuma a ser el node al que se li assigna més RAM, ja que ha de mantenir en memòria tota l'estructura de fitxers entre altres coses. Els nodes treballadors s'anomenen datanodes, normalment un per node del clúster, que gestiona l'emmagatzematge adjuntat als nodes on s'executen. Els usuaris poden emmagatzemar fitxers en el HDFS com si es tractés de qualsevol altre sistema de fitxers conegut com per exemple els de Linux, però a diferència d'aquest, el HDFS és un sistema distribuït. Per aconseguir això, el fitxer que es vol emmagatzemar en el HDFS es divideix en un o més blocs, que



Il·lustració 1 - Funcionament de l'arquitectura del HDFS

acostumen a ser de 64 MiB, i aquests blocs es guarden en el conjunt de datanodes de manera distribuïda. El namenode és l'encarregat de guardar en quin datanode es troben els blocs de dades, per tant, quan un client demana algun fitxer, el namenode buscarà quins son els datanodes que contenen els blocs necessaris, i els datanodes retornaran les dades demanades. Els datanodes també s'encarreguen de la creació, eliminació i replicació dels blocs a través de les instruccions indicades pel namenode.

El client fa una petició per crear un fitxer dins del sistema de fitxers de Hadoop al namenode. El namenode que és l'encarregat de gestionar i guardar l'estructura del sistema de fitxers, crea una entrada en el sistema de fitxers i si l'operació es duu a terme correctament, retorna un success al client. A partir d'aquí, el client crea un buffer que va acumulant dades fins arribar al tamany d'un bloc. El client pregunta al namenode quin és el datanode al que ha de assignar aquest bloc, el namenode indica a quin datanode assignarho i finalment es realitza l'assignació.

El HDFS és un sistema que garanteix alta disponibilitat i tolerància a fallades. Per aconseguir això Hadoop aporta una senzilla però efectiva solució. Quan s'emmagatzema un bloc en un Datanode, es realitza una rèplica en un altra datanode. El nombre de còpies que es realitzen es poden concretar a través del Replication Factor, que es pot configurar per tal d'indicar el nombre de còpies. Quan un bloc no està disponible en un datanode, les dades s'hauran de llegir des de un altra datanode que contingui una copia. S'ha d'escollir un nombre adequat de replication factor segons les nostres necessitats, ja que fer un procés de replicació de blocs es costós. L'espai que podem ocupar amb les rèpliques pot augmentar molt i arribar a ser un problema. Normalment s'utilitza un factor de replicació de 3, si el nombre és més gran a 3, la rèplica 4 i les següents a aquesta s'acostumen a assignar de manera aleatòria mantenint el nombre de rèpliques per sota del límit superior de la següent manera:

$$num.rèpliques = \frac{rèpliques - 1}{racks + 2}$$

6.2 YARN (Yet Another Resource Manager)

Yarn és un gestor de recursos que va sorgir a partir de la versió 2 de Hadoop. Inicialment, els components principals de Hadoop eren el HDFS i Mapreduce. Amb l'aparició de YARN, a la part més alta de l'arquitectura de Hadoop es pot utilitzar un mòdul de processament de dades distribuïdes diferent a Mapreduce, i a més, poder gestionar-ho.

La idea fonamental de YARN és dividir les funcionalitats de la gestió de recursos i la planificació / supervisió de tasques en dimonis separats. La idea és tenir un ResourceManager (RM) global i un ApplicationMaster (AM) per aplicació. Una aplicació és un treball únic o un DAG de treballs.

Està format pel ResourceManager i el NodeManager. El ResourceManager és l'encarregat de gestionar els recursos entre totes les aplicacions del sistema. El NodeManager és a totes les màquines i s'encarrega de monitoritzar la utilització de recursos en cadascuna d'aquestes, i a la vegada informar al ResourceManager d'això.

El ResourceManager està format per dos components principals:

- Scheduler: És l'encarregat d'assignar els recursos a les diferents aplicacions tenint en compte les limitacions de capacitat, les cues de treball, etc.
- ApplicationManager: S'encarrega d'acceptar peticions de treball i de negociar amb els contenidors on s'executen aquests treballs per tal d'executar una aplicació específica.

6.3 Spark

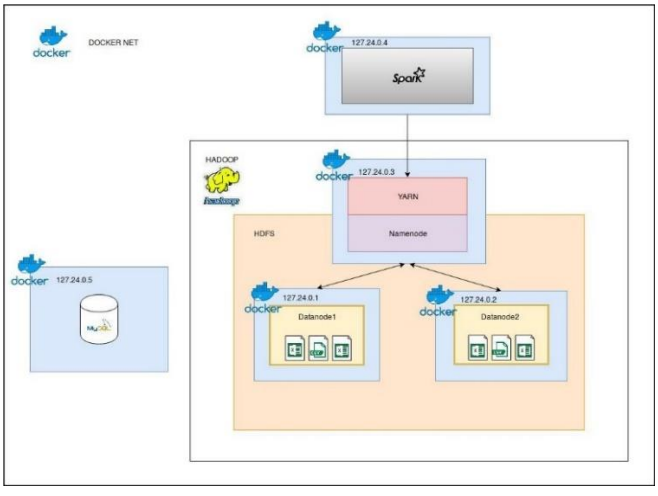
Spark [6] és un framework de codi obert de computació de dades distribuïdes en clústers. La característica principal de Spark és la computació en clústers en memòria, amb la qual s'aconsegueix la velocitat de processament d'una aplicació. Les característiques principals de Spark són les següents:

- Velocitat: Spark executa 100 vegades més ràpid que Hadoop Mapreduce en el processament de dades molt grans.
- Treballa en memòria: Una capa de programació simple proporciona la capacitat de poder treballar en memòria i a la vegada també permet persistència de dades en disc.
- Desplegament: Pot ser desplegat a través de diversos gestionadors de clusters com ara Hadoop YARN, Mesos o el propi de Spark.
- Temps real: Ofereix computació en temps real i baixa latència gràcies a es treballa en memòria.
- Diversitat de llenguatges disponible: Gràcies al conjunt de APIs de les que disposa es pot treballar en llenguatges d'alt nivell com ara Java, Scala, Python o R.

El node mestre és el que conté el driver program, l'encarregat de conduir tota l'aplicació. El codi escrit (o bé les ordres de la shell si utilitzem l'intèrpret) és el que es comporta com a driver program. Dins d'aquest driver program el primer que s'ha de fer sempre es crear el Spark Context, el qual és l'encarregat de establir connexió amb entorn d'execució de Spark i preparar els serveis interns d'aquests. Aquest Spark Context treballa amb un clúster manager per tal de gestionar diversos treballs. Un treball es divideix en múltiples tasques que son distribuïdes a través dels worker nodes. Els workers nodes són els nodes esclaus que s'encarreguen bàsicament d'executar les tasques.

6.4 Arquitectura del sistema implementat

L'arquitectura general del projecte està formada principalment per cinc containers Docker. Un container farà la funció de namenode i yarn manager, després tindrem dos datanodes, on s'emmagatzemen les dades csv i excels, un container que contindrà Spark i finalment un dedicat a MySQL. Els elements de Hadoop necessiten estar en una mateixa xarxa de Docker, per tal de poder comunicar-se, per això he creat una xarxa amb Docker a la que pertanyen tots els elements de l'arquitectura, de manera que no s'han de configurar els fitxers de /etc/hosts com



Il·lustració 2 - Arquitectura general del sistema implementat

s'hauria de fer en una instal·lació habitual sense Docker.

He decidit utilitzar dos datanodes ja que actualment la quantitat d'informació a emmagatzemar no és gran. Realment podría ser emmagatzemada en un únic datanode, però he decidit afegir un més per a poder fer un ús del replication factor. D'aquesta manera, tractaré de que sempre hi hagi una rèplica en els dos datanodes i si algun dels dos té algun inconvenient, es podran llegir les dades des de l'altre datanode i aconseguir així un sistema amb més disponibilitat.

6.5 Imatges utilitzades

Per implementar l'arquitectura hi havien principalment dues opcions: utilitzar imatges Docker ja creades (de Docker Hub, per exemple [7]) o bé crear les teves pròpies imatges i tot el necessari per a que funcionin. La idea inicial era utilitzar imatges ja creades i després fer el deployment d'aquestes, però durant la realització d'aquesta part vaig trobat molts problemes a l'hora d'utilitzar imatges ja creades que hem van fer perdre molt de temps. Per aquesta raó finalment vaig decidir crear les meves pròpies imatges per a quasi tots els mòduls.

Les diferents imatges que he utilitzat son les creades en els següents Dockerfiles, els quals he creat basant-me en alguns ja creats [8] [9] i modificant i afegint tot allò que veia necessari:

- hadoop-base-tfg: Aquesta imatge és la base de la resta i la que conté tots els elements bàsics necessaris per a poder tenir un correcte funcionament en tota la resta d'imatges. Aquesta imatge la he basat en ubuntu:16.04, ja que és un sistema al qual estic bastant acostumat i al ser Linux servia per a poder realitzar la feina. Utilitza l'última versió de Hadoop i Spark. Per a poder fer funcionar Hadoop és necessari tenir SSH ja que els diferents nodes es comuniquen a través de SSH, tenir el JDK de Java (per a la versió 3.2.0 de Hadoop podem utilitzar el JDK 8) i un editor per si és necessari modificar algun document dins el contenidor. Com vaig explicar al començament, Hadoop té una arquitectura master/slave (en aquesta última versió de Hadoop se'ls anomena workers en comptes de slaves). Per tant, en algun fitxer s'ha d'indicar quins són els nodes treballadors. Aquest fitxer s'anomena "workers", i conté les IP (o hostnames) dels nodes que son treballadors. En aquest cas, les IP associades tenen un alies dins de la xarxa creada per Docker i per tant he posat el hostname en comptes de la IP. Finalment el que faig és afegir el script start-hadoop.sh, que és l'encarregat de iniciar tots els serveis necessaris per a l'execució de Hadoop i de realitzar el format del namenode, que bàsicament el que fa és crear la metadata dels datanodes, que es guarda i es gestiona des de el namenode.
- namenode-tfg: Aquesta imatge té tot el necessari per a ser el namenode. El hostname del resource

manager de YARN també està en aquest mateix node. Creo el directori que utilitzarà el DFS per al namenode i faig un EXPOSE (fer accessibles aquests ports des de fora del container que es crea al executar la imatge) dels ports als quals haig d'accedir. Amb VOLUME faig que aquest directori sigui persistent, per tal de que quan l'execució del container s'aturi, no es perdin les dades. Finalment inicio el service de ssh i executo el script start-hadoop.sh comentat anteriorment.

- datanode-tfg: El datanode bàsicament haurà de fer les dades persistents i fer que es mantingui en execució per a que pugui ser utilitzar per el namenode.
- spark-tfg: Aquesta imatge té spark descarregat i conté tota la configuració necessària per a poder comunicar-se amb YARN. La versió descarregada de Spark es la més actual fins al mome
- mysql: Per al mòdul de MySQL, he utilitzat la imatge oficial [5]. Per a importar les dades en aquest container, he executat un script proporcionat per el tutor per a crear primer les taules.

6.6 Xarxa Docker

Per a gestionar l'execució dels containers primerament he desenvolupat un script en el qual he creat una xarxa de Docker anomenada "hadoop". En aquest script, es pot passar per paràmetre el nombre de datanodes que vols que s'inicialitzin, però actualment no s'haurien d'inicialitzar més de 2, ja que la configuració dels nodes "workers" està pensada per a dos datanodes. Els datanodes són executats en mode "detached", ja que aquests no han de realitzar cap execució sinó que han d'estar executant-se per a poder comunicar-se amb el namenode.

Quan s'executen les imatges, es concreta que tots els containers estiguin en la mateixa xarxa "hadoop", i afegeixo un hostname (node1, node2, namenode…) per a poder fer una configuració més general i no haver d'assignar IPs fixes als contenidors. Finalment, per a cadascun dels ports accessibles del namenode, realitzo un mapeig dels ports, per a que puguin ser accessibles des de fora.

6.7 Configuració Hadoop i Spark

Per a que l'arquitectura compleixi certs requisits, Hadoop i Spark poden ser configurats a través de diferents fitxers xml.

En la informació dels nodes dels clústers on treballaré s'especifiquen les següents característiques per a cadascun dels nodes:

Característiques
2 x 8 processor cores x64
64 GB de memòria
1 TB de disc dur
1 GB network connection

Els fitxers que he configurat són els següents:

- core-site: En aquest fitxer es configuren els paràmetres principals del HDFS. En la propietat fs.defaultFS s'indica quin és el sistema de fitxers per defecte. Gràcies a això es poden utilitzar les comandes de hdfs per a gestionar el contingut (pujar fitxers, llegir fitxers…) sense haver d'especificar tota la URL.
- hdfs-site: Conté tot el referent als diferents mòduls del HDFS. El dfs.replication fa referència al replication factor. En el dfs.namenode.name.dir i dfs.namenode .data.dir el que s'indica és quin és el path on s'emmagatzemen les dades tant en el namenode com el datanode respectivament.
- El paràmetre yarn.nodemanager.resource .memory-mb indica el total de memòria que pot ser assignada per als containers. He assignat un màxim de 54 GB disponible per als contenidors, ja que s'ha de tenir en compte que també s'ha de reservar memòria per al SO, i per als altres processos com ara el Namenode, Datanodes, etc. Per tant, dels 64 GB disponibles, he deixat 10 GB lliures i 54 GB utilitzables per als containers.
- spark-defaults.conf: Configuració referent a Spark. Per a escollir aquesta configuració he seguit un enfocament balancejat entre "Tiny executors" i "Fat" (un executor per core) [10]. Quan s'executa una aplicació Spark utilitzant un gestor de clústers com Yarn, hi haurà diversos daemons que s'executaran en segon pla com NameNode, NameNode secundari i DataNode. Per tant, mentre s'especifiquen els executors, hem d'assegurar-nos que deixem de banda bastants nuclis (~ 1 nucli per node) perquè aquests daemons funcionin sense problemes. El client HDFS té problemes amb un gran nombre de fils concurrents. Es va observar que HDFS aconseguia un rendiment complet d'escriptura amb ~ 5 tasques per executor. Per tant, és bo mantenir el nombre de nuclis per executor per sota d'aquest nombre.

Basant-me en aquestes recomanacions, he assignat 5 cores per executor per a tenir un bon rendiment del HDFS. He deixat un core lliure per als daemons de Hadoop i Yarn. Per tant, el número de cores disponible pasarà a ser 16 - 1 = 15. Per tant, el nombre total de cores en el clúster serà de:

$$total\ cores = \#cores\ disponibles \cdot total\ nodes\ en\ cluster = 15 \cdot 2 = 30$$

Com a número de executors disponibles tenim:

$$Num.exec\ disp = \frac{total\ cores}{num.cores\ per\ executor} = \frac{30}{5} = 6$$

Si tenim que num.executors disponibles és igual a 6,

cadascun dels nodes tindrà:

$$Exec.per\ node = \frac{num.exec.disponibles}{total\ nodes\ cluster} = \frac{6}{2} = 3$$

Amb això tenim que la memòria per cada executor serà de:

$$Memòria\ per\ executor = \frac{total\ memòria}{executor\ per\ node} = \frac{64\ GB}{3} = 21\ GB$$

6.8 Connexió client amb clúster

Per a poder realitzar les proves i la 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 amb YARN.

El primer pas realitzat per poder aconseguir la connexió del client amb el clúster ha sigut fer un conjunt de redireccions en els ports [11] de manera que, al estar treballant amb Docker, per a que els ports puguin ser escoltats desde fora de la VLAN, primerament he hagut d'exposar-los. Una vegada exposats, he realitzat un conjunt de redireccions de manera que quan s'accedeix a un port concret del node compute-0-24 de dccluster, aquest a la seva vegada accedeix al port que ens interessa dels diferents serveis que pot oferir Hadoop o Spark, els quals estan connectats a través d'una xarxa Docker

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. Primerament les execucions es van realitzar amb spark submit ja que l'objectiu principal era testejar si l'execució en mode clúster funcionava. A l'hora d'executar el script va aparèixer un error indicant que no s'havien 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 podia arribar a ser un procés complicat. L'explicació que tenia mes sentit degut a la descripció de l'error [12] i com estava estructurada l'arquitectura, consistia en que el client estava intentant connectar-se al clúster a través de les dades privades de la VLAN en la que està muntada l'arquitectura. El client demanava al namenode la informació pertanyent als datanodes. El namenode no es comunicava directament amb els datanodes, sinó que retornava les dades que emmagatzemava en les metadades al client per a que aquest s'encarregués de realitzar la comunicació. Per tant, quan el client rebia les dades, estava rebent les dades pertanyents a la virtual LAN i al tractar de comunicar-se amb aquestes dades sorgia un error.

La solució a aquest problema, tenint en compte que el projecte havia de continuar i una gran quantitat del temps d'aquesta fita s'havia 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.

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.

6.9 Interfície de dades

Les dades distribuïdes en el HDFS han de poder ser accedides fàcilment, sense que l'usuari que utilitzi les dades tingui la necessitat d'entendre on són aquestes dades ni com accedir-hi. Aquestes dades estan pensades per a ser utilitzades per un estudiant de màster, el qual està creant uns mòduls en Python que s'encarreguen del tractament de les dades per a un posterior anàlisis. La interfície de dades que he desenvolupat té com a objectiu accedir a les dades i retornar-les, de manera que el col·laborador encarregat de la part del tractament de les dades s'encarregui d'utilitzar-les. Per a facilitar la compatibilitat entre la seva part i la interfície creada, he decidit utilitzar com a llenguatge de programació Python i he retornat les dades en una estructura de dades compatible amb aquest.

Actualment disposem de les dades de tres localitzacions diferents: Sant Feliu, Castellví Rosanes i Santa Coloma. El codi està pensat per a que es pugui seleccionar qualsevol de les tres localitzacions i retornar totes les dades emmagatzemades i distribuïdes en el HDFS pertanyents a aquestes localitzacions. Revisant el codi proporcionat pel col·laborador encarregat del tractament de les dades, vaig observar que hi havia un conjunt de camps dins de les dades que li eren de més interès. Amb aquestes dades crea una estructura per a cadascuna de les localitzacions, les quals contenen les dades principals amb les que treballa.

Per tant, a part de retornar totes les dades de cadascuna de les localitzacions, també vaig crear unes funcions encarregades de retornar exclusivament les dades seleccionades pel col·laborador per a cadascuna de les localitzacions i d'aquesta manera facilitar l'accés a les dades.

6.10 Testing

Per entendre el conjunt de proves [14][15] que s'han realitzat hem d'entendre quins son els diferents passos que es realitzen per processar el Big Data en una arquitectura basada en Hadoop i Spark.

Podem separar les fases per les quals passa la informació en tres passos principals:

1. Carregar fitxers de dades en el HDFS: Dades de moltes fonts diverses son carregades en el HDFS i son dividides en múltiples fitxers. Aquestes dades es distribueixen en diferents nodes a partir dels quals podem accedir a les dades.
2. Realitzar el processament amb Spark: Una vegada les dades són carregades en el HDFS, es

realitzen operacions per al processament d'aquestes i poder obtenir els resultats desitjats. Podem utilitzar diferents frameworks de processament, en aquest cas, utilitzem Spark.

3. Extracció de dades i/o resultats: En aquest últim pas es tracta de extreure les dades de sortida resultants del processament i carregar-les en alguna estructura com ara un data warehouse per tal de generar reports analítics. En aquest projecte aquest últim pas no era un dels objectius, per tant, les dades resultants es guarden en logs propis del sistema Hadoop.

Il·lustració 3 - Àrees on enfocar-se per realitzar testing en Big Data

En aquest tipus de sistemes el fet de treballar amb una quantitat de dades molt gran i estar executant el codi en diferents nodes fa que sigui molt probable trobar problemes en cadascuna d'aquestes tres fases principals. És a causa d'això que l'enfocament del testing en Big Data és centra en cadascuna d'aquestes tres fases.

6.10.1 Testing funcional

Algunes de les proves que s'han realitzat es basen en tests funcionals, un tipus de proves on el sistema es testejat per tal de comprovar que el sistema compleix els requisits funcionals. Les proves de test funcionals es basen en observar quins son els outputs o resultats donada una informació d'entrada i comprovar que el sistema compleix correctament els requisits.

Dins de les proves funcionals he realitzat les següents validacions:

- Validació de data staging: Alguns dels problemes que podem trobar durant la primera fase del procés on movem les dades de l'origen al sistema de Hadoop són:
  - Que hi hagi una incorrecta captura de la informació.
  - Un incorrecte emmagatzematge de la informació o bé un emmagatzematge on les dades son incompletes.
  - Que hi hagi una replicació incorrecta.

  Les validacions que podem realitzar en aquesta fase poden ser comprovar que les dades son carregades correctament en el HDFS, respectant la destinació on em indicat que es guardin les dades, comparar que les dades originals son iguals que les que hem pujat al HDFS, i per últim validar que les dades són replicades correctament respectant el factor de replicació indicat, de manera que les dades també estiguin disponibles en diferents nodes i que es respecten els tamanys de blocs.
- Validació del processament Spark: Els possibles problemes que podem trobar a l'hora del processament de Spark poden ser problemes a l'hora d'executar el codi a causa d'alguna incompatibilitat o de codi no funcional, problemes a l'hora d'executar codi en mode clúster quan en client funciona correctament o un problema a l'hora de retornar el output. Per tant, s'ha validat que el procés es completa satisfactòriament, tant en mode client com en mode clúster, de manera que el output que obtenim sigui el desitjat.
- Validació extracció resultats: Com he comentat anteriorment, en aquest projecte no esta contemplada l'extracció dels resultats en cap tipus d'estructura, per tant, només s'ha comprovat que els logs amb els resultats son accessibles.

6.10.2 Testing no funcional

Les proves no funcionals tracten de validar els requisit que no son funcionals, aquells que tenen més a veure amb característiques com ara el rendiment del sistema o bé els recursos que aquest utilitza. En aquest cas, he realitzat una prova de rendiment comparant el temps d'execució d'un script en mode client i en mode clúster. Per realitzar les proves he utilitzat un conjunt de datasets [16] 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 tamanys 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.

7 RESULTATS OBTINGUTS

Els resultats mostrats en aquest apartat es basen en les diferents validacions que s'han proposat en l'apartat de testing, de manera que he comentat quins resultats s'han obtingut per a cadascuna de les proves i he interpretat els resultats.

En quant als requisits funcionals, seguint les validacions del data staging, el primer que s'ha comprovat és que les dades carregades en el HDFS es carreguessin de manera correcta i en el directori indicat. Al pujar les dades a través de la comanda "put" de Hadoop indicant el directori on volem que es carreguin, si anem a la UI del HDFS, podem comprovar que les dades s'han carregat correctament.

Per validar que aquestes dades estiguessin disponibles en diversos nodes, podem accedir a la informació addicional del fitxer carregat, on trobem un camp anomenat "availability" que ens mostra els nodes on estan disponibles aquestes dades. En aquest cas les dades son disponibles en els dos nodes creats, per tant, la distribució del fitxer és correcte. Les dades es distribueixen en tots els

nodes ja que la quantitat de dades a emmagatzemar no és molt gran, i per tant, només necessitem dos nodes. En el cas de que tinguéssim més informació a emmagatzemar i, per tant, més nodes, no seria una bona idea que els fitxers es distribueixin en cadascun dels nodes ja que això suposaria un cost addicional i un malbaratament d'espai innecessari.

La següent validació és comprovar que es respecten els tamanys de bloc. En la mateixa UI del HDFS podem trobar informació com ara el tamany de bloc i el nombre de blocs creats per a cadascun dels fitxers. Obtenim el tamany de bloc per defecte en Hadoop de 128 MB, que és el que vam tenir en compte des d'un inici del projecte.

Quan un dels blocs sobrepassa el tamany de 128 MB, si accedim a la informació addicional del fitxer, trobem que el nombre de blocs creats són els necessaris, respectant així el tamany de bloc indicat.

Tenint en compte que el tamany dels fitxers de les dades de les diferents localitzacions no són massa grans i que no sobrepassen el tamany de bloc, podem adonar-nos que realment el tamany de bloc utilitzat per a les dades que tenim es massa gran. Les dades que utilitzem sobre les poblacions més que ser grans fitxers, està pensat que siguin una gran diversitat de fitxers, però que aquests no siguin de un gran tamany, per tant, possiblement hauria sigut una bona idea reduir el tamany de bloc de Hadoop per tal d'adaptar-ho a les nostres dades. D'aquesta manera possiblement hauríem obtingut a la vegada un millor rendiment en el sistema i s'estalviaria espai.

En quant a la validació de comparació dels fitxers de dades d'entrada i els emmagatzemats en el HDFS, fent ús de l'eina fc de Windows que s'encarrega de comparar dos fitxers, he guardat el resultat en un .txt anomenat output per comprovar si existeix alguna diferència entre els dos fitxers. Com a resultat obtenim que no hi ha cap diferencia entre els dos fitxers, per tant, podem concloure que els dos fitxers contenen la mateixa informació.

La següent validació realitzada ha sigut la de garantir que Spark executa una aplicació de manera satisfactòria, tant en mode clúster com en mode client. Al executar una aplicació, es rep un final status SUCCEEDED indicant que l'aplicació s'ha executat correctament (sempre i quan no hi hagin errors en el codi).

```
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
```

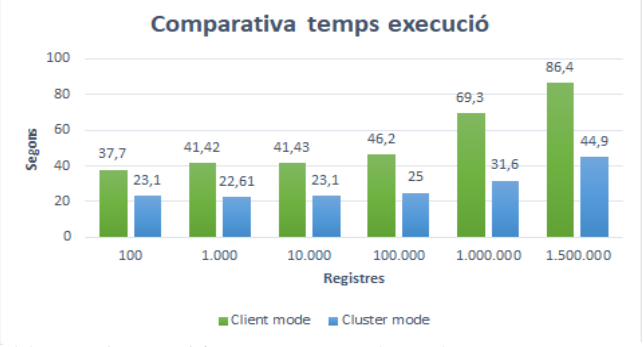
Il·lustració 4 - Execució satisfactòria d'una aplicació en mode clúster

Al executar el script en mode client obtenim l'output dels resultats directament en la consola de comandes on s'ha executat, ja que el programa s'executa en aquesta mateixa màquina, en canvi, al executar en mode clúster, el output s'obté en els logs dels nodes on s'ha executat, els quals també són accessibles a través de la UI del NameNode. Si fos necessari, es podria especificar més concretament on volem que s'emmagatzemin els outputs.

Pel que fa a la interfície de dades, a través de l'execució d'aquesta s'ha obtingut una estructura de dades de tipus Dataframe, amb els camps necessaris, o bé, el conjunt de totes les dades, que podrien ser utilitzades directament pel usuari.

Finalment, en la realització de les proves comentades en l'apartat de test no funcional, s'ha realitzat l'execució dels diferents fitxers tant en mode client com en mode clúster per tal de poder comparar els diferents resultats.

Tal i com podem veure a la Figura XX, quan executem el script que treballa amb 100 registres en mode client obtenim un temps d'execució de 37'7 segons, en comparació amb l'execució en clúster que obtenim 23'1 segons, pot semblar que la diferencia entre les dues execucions no es massa notòria. En canvi, a mida que el fitxer de dades va creixent i tenim més registres, la diferencia es va fent mes notable, obtenint pràcticament la meitat del temps d'execució en mode clúster en el script de que conté 1.500.000 registres. Això sembla indicar que el sistema esta dissenyat per a funcionar amb grans quantitats de dades i que per tant, la seva utilitat radica en utilitzar-ho per a aquests cassos.



Il·lustració 5 - Gràfica comparativa de rendiment entre una execució en mode client i en mode clúster

8 CONCLUSIÓ

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 un procés de recerca sobre les tecnologies utilitzades per a la implementació del sistema 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 gran part dels objectius principal s'han pogut dur a terme, aconseguint una arquitectura big data funcional en el clúster de dccluster. Tot i que el projecte es funcional com a tal, i els diferents components estan executant-se en el node-0-24 de dccluster, hi ha un conjunt de millores que es podrien realitzar abans de posar aquest sistema en producció, com ara modificar el tamany dels blocs, que podrien passar de 128 MB a 64 MB, reduint així l'espai ocupat innecessàriament, o bé, aconseguir que la connexió des d'un client pogués realitzarse sense la necessitat de connectar-se a una Shell dins de la VLAN, sinó aconseguir que els clients externs a la VLAN puguin connectar-se directament, potser tractant d'utilitzar els hostnames en comptes de les direccions IP en la comunicació amb els datanodes [17].

També amb la disposició de més temps, es podria finalitzar la part de la interfície referent a l'harmonització de les dades, per tal de retornar les dades ja harmonitzades i estalviar aquest procés a l'usuari.

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 fent 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.

AGRAÏMENTS

Primerament m'agradaria agraïr al meu tutor, Oriol Ramos, la dedicació a l'hora d'atendre els dubtes i tractar de millorar la meva feina. Quan he necessitat feedback de la meva feina, tot i quan la demanava amb no massa antelació, ha estat disposat a dedicar-hi temps i si feia falta, realitzar reunions per tal d'anar veient quins punts s'havien de millorar.

També m'agradaria agraïr al Centre d'Estudis Demogràfics per haver-nos cedit les dades necessàries de les diferentes poblacions.

BIBLIOGRAFIA

[1] Centre de Visió per Computació, "XARXES: Tecnologia i innovació ciutadana en la construcció de xarxes socials històriques per a la comprensió del llegat demogràfic". [En línia]. Disponible a: http://dag.cvc.uab.es/xarxes/info Consultat: 13 de febrer de 2019

[2] B.Bahari, "Hadoop Fundamental", Febrer 6, 2017. [En línia]. Disponible a: https://medium.com/sarccom/hadoop-fundamental-5179099f5b21 Consultat: 13 de febrer de 2019

[3] Apache, "HDFS Architecture", 13 de novembre, 2018. [En línia]. Disponible a: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html Consultat: 15 de febrer de 2019

[4] CollabNet, "What is Kanban? An introduction to Kanban methodology". [En línia]. Disponible a: https://resources.collab.net/agile-101/what-is-kanban [Consultat 6 de Març, 2019]

[5] "Trello, la manera gratuita, flexible y visual de organizarlo todo con cualquiera." (2019). Trello. [En línia] Disponible a: https://trello.com [Consultat 22 Feb. 2019]

[6] N. Vaidya, «Apache Spark Architecture – Spark Cluster Architecture Explained» 27 Febrer 2019. [En línia]. Disponible: https://www.edureka.co/blog/spark-architecture/. [Últim accés: 25 Març 2019].

[7] I. Ermilov, «Scalable Spark/HDFS Workbench using Docker» 23 Març 2016. [En línia]. Disponible: https://www.big-data-europe.eu/scalable-sparkhdfs-workbench-usingdocker/. [Últim accés: 20 Març 2019].

[8] Arvind, «Creating hadoop docker image» 27 Juny 2018. [En línia]. Disponible: http://bigdatums.net/2017/11/04/creating-hadoop-docker-image/. [Últim accés: 22 Març 2019].

[9] F. H. Linode, «How to Install and Set Up a 3-Node Hadoop Cluster» 1 Juny 2018. [En línia]. Disponible: https://www.linode.com/docs/databases/hadoop/how-to-install-andset-up-hadoop-cluster/. [Últim accés: 22 Març 2019].

[10] Spoddutur, «Distribution of Executors, Cores and Memory for a Spark Application running in Yarn» [En línia]. Disponible: https://spoddutur.github.io/sparknotes/distribution_of_executors_cores_and_memory_for_spark_application.html. [Últim

[11] 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].

[12] 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].

[13] 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].

[14] 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

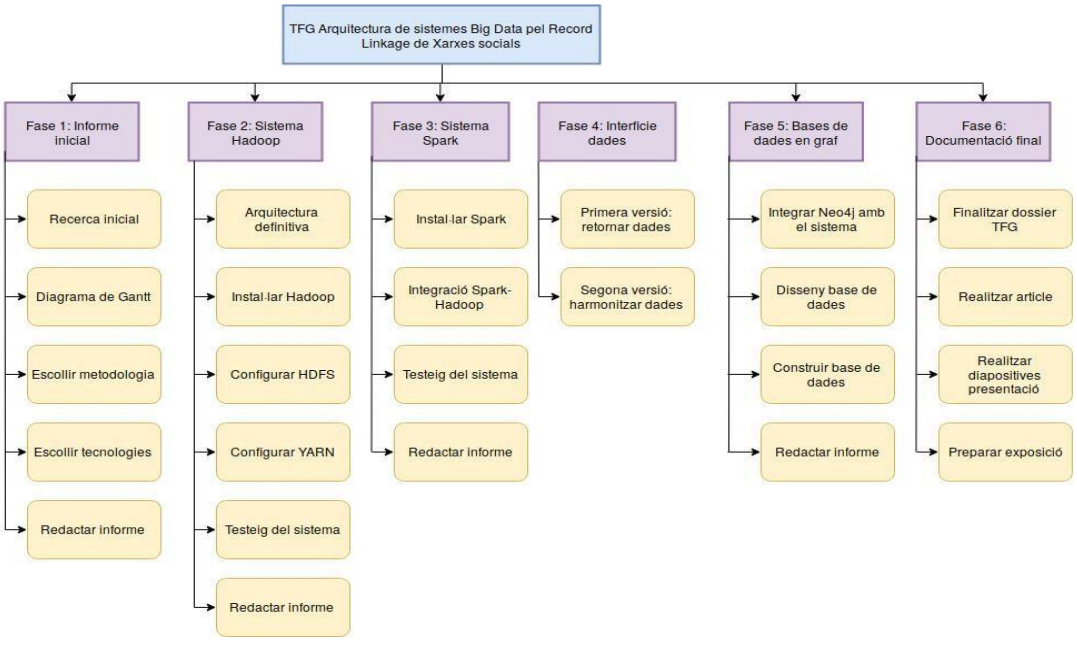
[15] Gudipati, M., Rao, S., Mohan, N. D., & Gajja, N. K. (2013). Big data: Testing approach to overcome quality challenges. Big Data: Challenges and Opportunities, 11(1), 65-72. URL: https://pdfs.semanticscholar.org/e0a1/e310956085428e04c93ec2e3b15dffecb112.pdf

[16] 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]

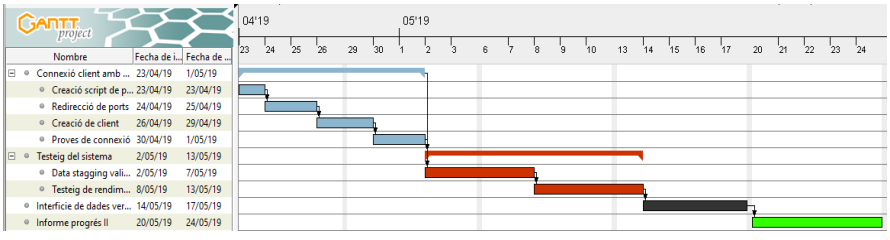
[17] Apache Hadoop, «HDFS Support for Multihomed Networks», 17 Març 2017. [En línia]. Disponible a: https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoophdfs/HdfsMultihoming.html. [Últim accés: 11 Maig 2019].

APÈNDIX

A1. FASES DEL PROJECTE I PLANIFICACIÓ

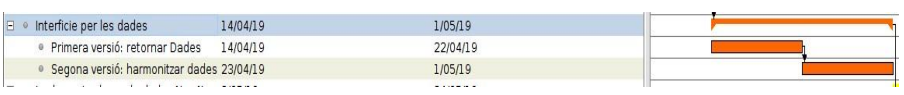


Il·lustració 6 - WBS inicial de les fases del projecte

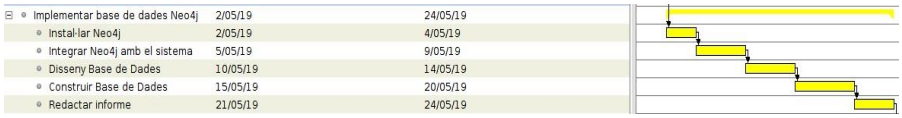


Il·lustració 7 - Diagrama de Gantt que inclou tasques de l'informe inicial

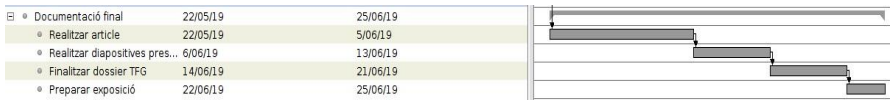
Il·lustració 8 - Diagrama de Gantt inicial amb les tasques a realitzar per a la primera entrega de l'informe de progrés



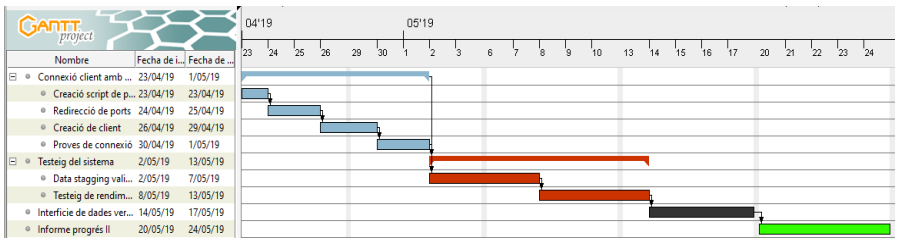
Il·lustració 9 - Diagrama de Gantt incial amb les tasques a realitzar per a la fase de interfície de dades



Il·lustració 9 - Diagrama de Gantt inicial de les tasques de bases de dades estructurada en graf

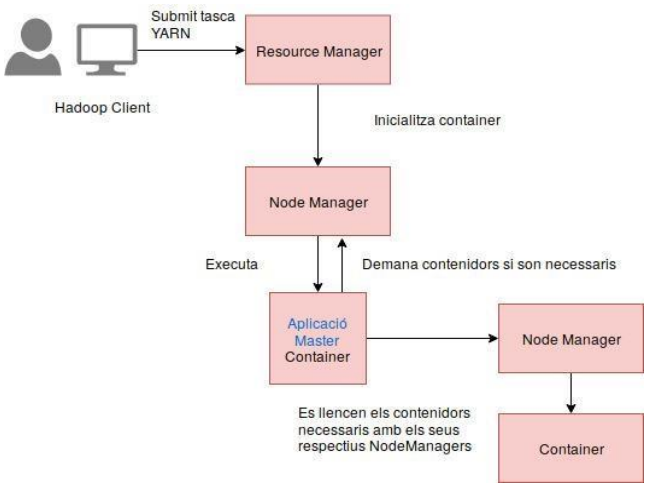


Il·lustració 10 - Diagrama de Gantt inicial amb les tasques per finalitzar el projecte

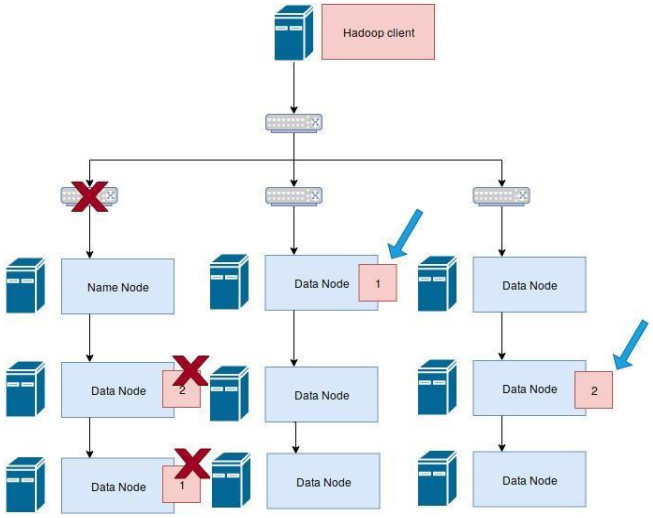


Il·lustració 11 - Replanificació per a la fita de progrés II

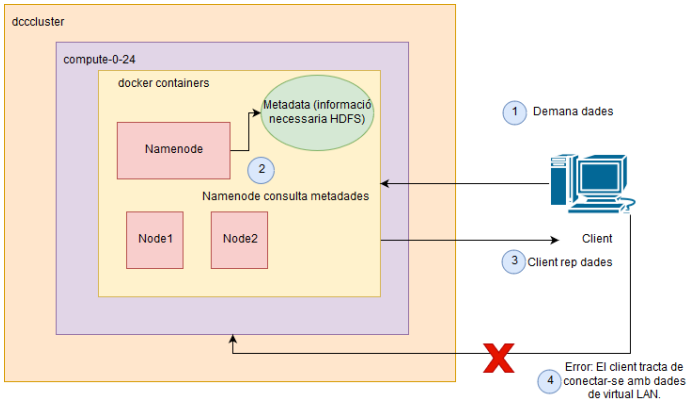
2. DIAGRAMES ARQUITECTURA



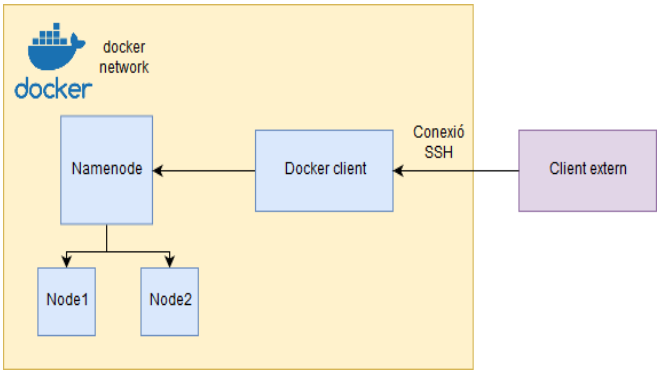
Il·lustració 12 – Funcionament YARN



Il·lustració 13 – Funcionament rack awareness



Il·lustració 14 - Descripció del problema a l'hora d'executar codi en mode clúster



Il·lustració 15 - Solució proposada per a solucionar el problema

3. Exemples fitxers Dockerfile

```
FROM hadoop-base-tfg:latest

# set environment vars
ENV HADOOP_HOME /opt/hadoop
ENV SPARK_HOME /opt/spark
ENV SPARK_VERSION 2.4.0
ENV PATH "$PATH:$SPARK_HOME/bin"
ENV HADOOP_CONF_DIR $HADOOP_HOME/etc/hadoop
ENV LD_LIBRARY_PATH $HADOOP_HOME/lib/native:$LD_LIBRARY_PATH

#download and extract Spark
RUN \
  wget https://archive.apache.org/dist/spark/spark-$SPARK_VERSION/spark-$SPARK_VERSION-bin-hadoop2.7.tgz && \
  tar -xvf spark-$SPARK_VERSION-bin-hadoop2.7.tgz && \
  mv spark-$SPARK_VERSION-bin-hadoop2.7 $SPARK_HOME && \
  echo "export SPARK_HOME=$SPARK_HOME" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export HADOOP_CONF_DIR=$HADOOP_CONF_DIR" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  mv $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-defaults.conf

ENV PYSPARK_PYTHON /usr/bin/python3

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

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

CMD ["/usr/sbin/sshd", "-D"]

# start hadoop
# CMD bash start-hadoop.sh
```

Il·lustració 16 - Dockerfile imatge Spark client

```
FROM ubuntu:16.04

# set environment vars
ENV HADOOP_HOME /opt/hadoop
ENV HADOOP_VERSION 3.2.0
ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64
ENV HDFS_NAMENODE_USER root
ENV HDFS_DATANODE_USER root
ENV HDFS_SECONDARYNAMENODE_USER root
ENV YARN_RESOURCEMANAGER_USER root
ENV YARN_NODEMANAGER_USER root
# install packages
RUN \
  apt-get update && apt-get install -y \
  ssh \
  rsync \
  vim \
  openjdk-8-jdk


# download and extract hadoop, set JAVA_HOME in hadoop-env.sh, update path
RUN \
  wget http://apache.mirrors.tds.net/hadoop/common/hadoop-$HADOOP_VERSION/hadoop-$HADOOP_VERSION.tar.gz && \
  tar -xzf hadoop-$HADOOP_VERSION.tar.gz && \
  mv hadoop-$HADOOP_VERSION $HADOOP_HOME && \
  echo "export JAVA_HOME=$JAVA_HOME" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export HDFS_NAMENODE_USER=$HDFS_NAMENODE_USER" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export HDFS_DATANODE_USER=$HDFS_DATANODE_USER" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export HDFS_SECONDARYNAMENODE_USER=$HDFS_SECONDARYNAMENODE_USER" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export YARN_RESOURCEMANAGER_USER=$YARN_RESOURCEMANAGER_USER" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export HADOOP_ROOT_LOGGER=DEBUG,console" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "export YARN_NODEMANAGER_USER=$YARN_NODEMANAGER_USER" >> $HADOOP_HOME/etc/hadoop/hadoop-env.sh && \
  echo "PATH=$PATH:$HADOOP_HOME/bin" >> ~/.bashrc

# create ssh keys
RUN \
  ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa && \
  cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys && \
  chmod 0600 ~/.ssh/authorized_keys

# copy hadoop configs
ADD configs/*xml $HADOOP_HOME/etc/hadoop/
ADD configs/*properties $HADOOP_HOME/etc/hadoop/

COPY configs/* /tmp/

RUN mv /tmp/workers $HADOOP_HOME/etc/hadoop/workers
```

Il·lustració 17 - Dockerfile principal de Hadoop