Universitat de les Illes Balears

escola politècnica superior

Treball Final de Grau

ENGINYERIA INFORMÀTICA

Hadoop com a eina de Big Data. Aplicació sobre dades acadèmiques.

ANTONIO CRESPÍ TOBEÑA

Tutor
Gabriel Fontanet Nadal

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 19 de juny de 2017

SUMARI

ÍNDEX DE FIGURES

ACRÒNIMS

ACID Atomicitat, Consistència, aÏllament i Durabilitat

API *Application Programming Interface*

GFS *Google File System*

GIF2 Grau d'Infermeria (Pla 2016)

GIN2 Grau en Enginyeria Informàtica

HDFS *Hadoop Distributed File System*

IDE *Integrated Development Environment*

IoT *Internet of Things*

MPI *Message Passing Interface*

NDFS *Nutch Distributed File System*

NoSQL *Not only SQL*

PAU Proves d'accés a la Universitat

RAID *Redundant Array of Inexpensive Disks*

RDBMS *Relational Database Management System*

SETI *Search for Extra-Terrestrial Intelligence*

SQL *Structured Query Language*

YARN *Yet Another Resource Negotiator*

RESUM

Analitzar grans quantitats de dades s'ha convertit en una necessitat de les empreses de qualsevol sector. Per facilitar aquesta tasca en els darrers anys han sortit diverses eines que intenten automatitzar i simplificar part del procés. Aquest treball té com a objectiu comprendre l'estructura i el funcionament d'una de les eines, Hadoop, i comparar-la amb altres eines o tècniques.

Per il·lustrar l'estudi realitzat, a més d'alguns exemples petits i concrets pensats per temes puntuals, es presenta el cas d'un estudi sobre dades acadèmiques de la Universitat. En particular s'analitza la relació existent entre el fracàs dels estudiants a primer curs i la seva nota d'accés.

CAPÍTOL 1

INTRODUCCIÓ.

L'activitat d'Internet genera una gran quantitat de dades cada dia. Analitzar aquesta informació és vital per la majoria d'empreses modernes, ja sigui amb la intenció de millorar els seus serveis o per ajustar el seu comportament a les noves tendències. Però emprant tècniques tradicionals l'anàlisi seria tan llarg que una vegada acabat la informació seria obsoleta, així que és important emprar noves metodologies que permetin realitzar l'anàlisi en un temps raonable.

Per aplicar tècniques d'anàlisi modernes és vital tractar dos aspectes: l'emmagatzemament de la informació i el posterior processament.

Al capítol 2 es tracta el primer punt mitjançant una introducció a l'arquitectura de les aplicacions que treballen amb Big Data i a tècniques de modelatge dinàmic de la informació.

Al capítol 3 s'expliquen els principis de Hadoop, un dels *frameworks* més populars per l'anàlisi de la informació. També hi ha un petit exemple que il·lustrarà els conceptes teòrics i procediments explicats durant el capítol.

Finalment al capítol 4 hi ha el que dona nom al projecte: l'aplicació del que s'ha explicat de Hadoop damunt dades acadèmiques. En aquest capítol es detalla el procediment i el codi que dona resultat a la gràfica final, juntament amb les conclusions a els que s'arriben.

CAPÍTOL 2

BIG DATA

2.1 Què és Big Data? Aproximació al problema.

Durant l'era d'Internet la informació generada està creixent de forma exponencial, tant en volum com en tipus de dades diferents. Precisament a això fa referència el concepte de Big Data: dades generades de forma ràpida, des de múltiples fonts i en diferents formats.

No hi ha una mida a partir de la qual un conjunt de dades es considera "*Big*". És més bé un aspecte de metodologia: un conjunt de dades és "Big Data" quan per tractar-lo es requereixen noves tècniques i eines que permetin el processament en un temps raonable.

Degut a aquestes noves necessitats les bases de dades tradicionals es veuen obligades d'adaptar-se. Però així com les tecnologies "antigues" —principalment basades en *Structured Query Language* (SQL)— es van actualitzant sorgeixen alternatives com les bases de dades *Not only SQL* (NoSQL), que sumades a un sistema d'anàlisi com Hadoop són la principal opció de moltes companyies.

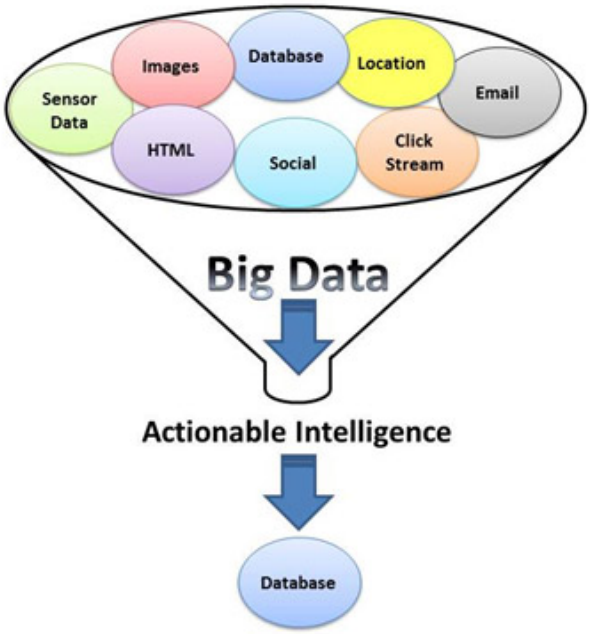

Figura 2.1: Esquema Big Data (Extreta de [1]).

2.2 És realmenet necessari emprar tècniques de Big Data? Evolució de la informació en els darrers anys.

Emprant un sistema d'emmagatzemament tradicional i adquirint memòria suficient podem arribar a guardar tota la informació que sigui necessària, però a l'hora de tractar-la sorgeix un nou problema: la velocitat de lectura dels nous discs no està creixent al mateix ritme que la seva capacitat. Però com de gran és la diferència?

Si agafam com exemple el Seagate ST-41600n[6] del 1990 que té una capacitat de 1.370 MB i una velocitat de lectura de 4,4MB/s, fent uns senzills càlculs podrem veure que es tardarien uns 5 minuts en llegir tot el disc. Uns 20 anys després els discs més comuns tenen 1TB de capacitat però la seva velocitat de lectura és de l'ordre de 100MB/s, així que llegir-ne un sencer suposa prop de dues hores i mitja.

En un intent de baixar aquest temps podem paral·lelitzar les lectures, llegint de diversos discs a l'hora. Si tinguéssim 100 discs, cadascun amb un 1% de les dades, i llegíssim de tots a l'hora tardaríem poc més de 2 minuts en acabar. Però emprar un disc del que només se n'ocupa un 1% resulta en molt d'espai desaprofitat.

Una millora d'aquest sistema seria posar a cada disc un conjunt de dades de 1 TB. Així podríem coordinar les lectures d'uns discs amb l'anàlisi de les dades llegides a altres i obtindríem un augment global de velocitat.

Però així com començam a tractar amb major quantitat de *hardware* se'ns presenten nous inconvenients. El més evident és la fallida d'un disc, provocant la pèrdua de

les dades que conté. Això es podria prevenir adoptant un sistema de tipus *Redundant Array of Inexpensive Disks* (RAID), on es dupliquen dades per tal de prevenir pèrdues.

A més, cal tenir en compte com s'actuaria si fos necessari combinar dades de discs diferents en cas de que es trobessin relacionades. El problema en sí és complex de resoldre, però molts dels *frameworks* de computació distribuïda (Hadoop entre ells) ho tracten de forma eficient i transparent al programador[4].

És evident que resulta necessari refinar el procés d'anàlisi a mesura que creix el nostre conjunt de dades, ja que si no ho feim tardarà un temps inassumible o els resultats no seran fiables. Per aquest motiu en la darrera dècada han sorgit entorns de desenvolupament que faciliten totes les tasques d'anàlisi i s'encarreguen d'aspectes com la prevenció de fallides o la integritat de les dades.

No obstant, abans de parlar de *frameworks* és necessari saber com s'estructuren i com s'emmagatzemen les dades en un entorn de Big Data.

2.3 Model de Big Data.

El model de Big Data proporciona una arquitectura que permet a més aplicacions fer ús de les dades.

El model està format per tres capes, tal i com es pot veure a la figura 2.2:

- La **capa física** és on tenim emmagatzemades totes les dades del nostre sistema: arxius d'audio, vídeo, *logs*, etc.
- La **capa del model de dades** comunica la capa de processament amb la física, permetent l'obtenció i emmagatzemament de la informació.
- La **capa de processament de dades** és on tractarem l'informació. És la capa on actuen els *frameworks* de Big Data.

Per tal de que qualsevol canvi tengui el mínim impacte possible una capa només interactua amb les adjacents. Per exemple si canviam l'estructura de la base de dades on tenim emmagatzemada la informació —canvi a la capa física— només haurem d'actualitzar el model.

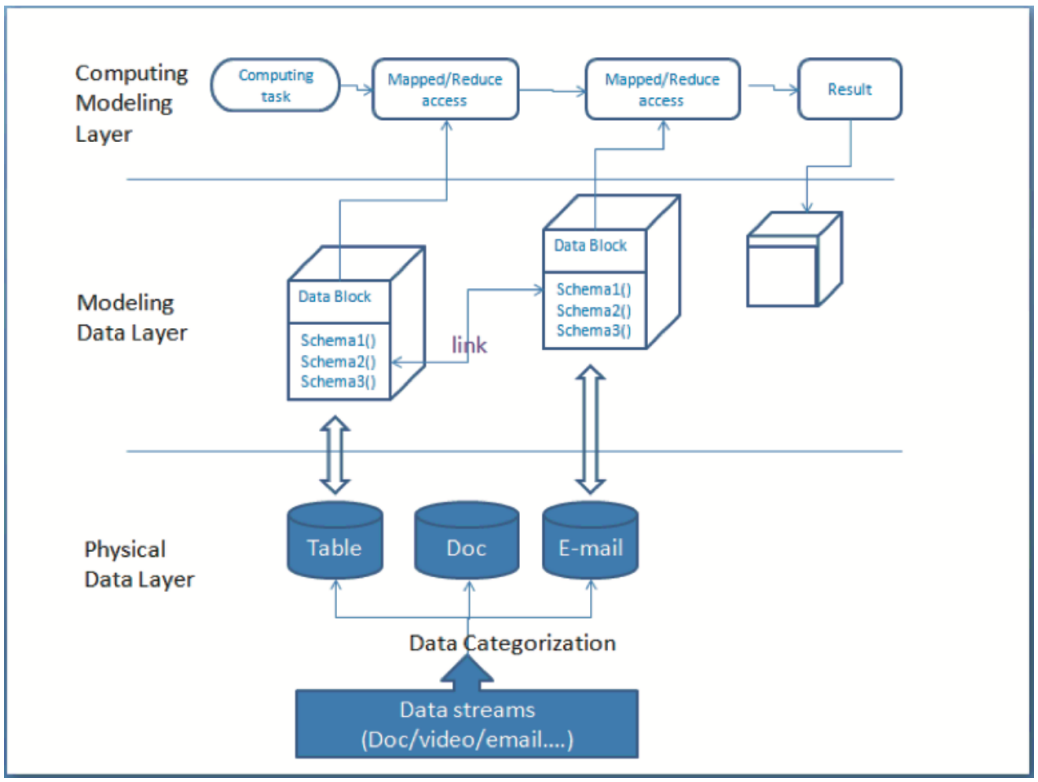

Figura 2.2: Arquitectura Big Data (Extreta de [2])

.

2.4 Arquitectura híbrida

Quan treballam amb Big Data podem tenir les dades dins una base de dades relacional —basada en SQL— o dins una base de dades no relacional —NoSQL—. Un sistema NoSQL ens proporciona una major capacitat d'emmagatzemament i facilitat d'escalament, mentre que la forma de fer consultes i operacions de tipus *join* a una base de dades SQL resulta molt pràctica i eficient.

És per aquest motiu que en els darrers anys empreses com Oracle o Microsoft —gegants del món de la informàtica que desenvolupen sistemes SQL— estan incloent característiques NoSQL als seus productes. De la mateixa manera, bases de dades NoSQL com Cassandra[7] o Hive[8] intenten incorporar un llenguatge per realitzar consultes similar a SQL.

Però així i tot hi segueix havent clares diferències entre els dos sistemes i a l'hora de programar una aplicació hem de sospesar totes les opcions i decidir quina ens convé més. En resum, les seves característiques són:

Sistema SQL:

- Permet relacionar taules de dades.
- Permet realitzar consultes complexes en un temps raonable.
- Assegura les característiques Atomicitat, Consistència, aÏllament i Durabilitat (ACID).

Sistema NoSQL:

- Disposa d'una gran capacitat d'escalament.
- Permet afegir dades en forma de columnes, no només en forma de files.
- Les dades es poden organitzar de forma jeràrquica o de graf.

2.5 La necessitat d'un model dinàmic.

Treballant amb Big Data sovint trobam que les dades a tractar són desestructurades o semi-estructurades —no sabem la quantitat de dades que hi ha, de quin tipus són o les relacions entre elles—. Però així i tot cal definir un model, ja que facilita l'anàlisi.

La tendència dels darrers anys és emprar tècniques de post-modelat, és a dir, crear el model una vegada les dades han estat recollides i emmagatzemades[2] (veure 2.3).

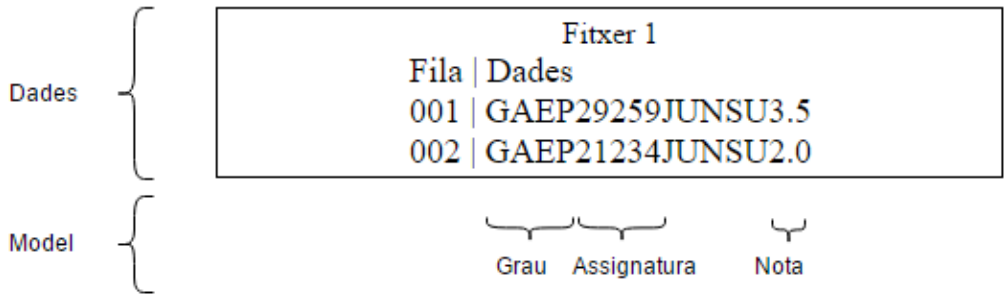

Figura 2.3: Actuació d'un model dinàmic (Elaboració pròpia).

Per crear un model d'aquest tipus el que hem de fer és dir a quina part de dades s'aplica i quins tipus de dades suporta. Per exemple, donada la cadena "`GAEP29259JUNSU3.5`", hauríem de dir al nostre model que els caràcters 0..3 —"GAEP"— són el codi del grau, els dígits 4..8 —"29249"— són el codi de l'assignatura, etc.

D'aquesta manera tendríem un fitxer amb dades emmagatzemades en forma de cadenes com l'anterior i si en temps d'execució es demana el codi del grau, el model retornarà els quatre primers caràcters de la línia que estem tractant.

CAPÍTOL 3

HADOOP I L'ANÀLISI DE DADES.

3.1 Què és Hadoop?

3.1.1 Història.

Hadoop[9] va ser creat per Doug Cuttling, desenvolupador de la llibreria de cerca de text Apache Lucene[10]. Té el seu origen en un motor de cerca anomenat Nutch[11] que formava part del projecte Lucene.

La construcció de Nutch va començar l'any 2002 però els seus desenvolupadors aviat van veure que amb l'arquitectura de la que disposaven era impossible que el motor de cerca cobris les milions de pàgines de la web. Basats en la publicació de Google de l'any 2003 on es descrivia el seu sistema d'arxius, *Google File System* (GFS),[12] van escriure'n una versió *open source* que van anomenar *Nutch Distributed File System* (NDFS) i va estar llesta l'any 2004.

També al 2004 Google va publicar l'article on introduïa MapReduce al món[13]. A principis de 2005, els desenvolupadors de Nutch havien afegit MapReduce al seu programa i a mitjans del mateix any la majoria dels seus algoritmes ja funcionaven damunt NDFS amb tècniques de MapReduce.

Aviat van veure que la versió de Nutch que combinava NDFS i MapReduce podia simplificar tasques d'àrees diferents, així que van passar Nutch a un projecte independent anomenat Hadoop. Va ser en aquesta època que Doug Cutting va començar a treballar a Yahoo! i l'empresa li va proporcionar un equip dedicat i recursos per transformar Hadoop en un sistema més gran i complet. El primer èxit del projecte va ser quan Yahoo! va començar a emprar Hadoop per generar el seu índex de cerca[14].

Al gener de 2008 Hadoop era un dels projectes amb més èxit d'Apache i comptava amb una comunitat sòlida i diversa. Havia demostrat ser útil en moltes àrees i

l'empraven empreses tan diferents com Yahoo!, Last.fm, Facebook o New York Times[1].

Per exemple, New York Times va digitalitzar 4 TB d'articles antics, fent que estiguessin disponibles a la xarxa[15]. Processar tots aquests articles va tardar menys de 24 hores emprant 100 màquines, i la computació va ser duta a terme combinant el model de programació paral·lela de Hadoop i el servei d'Amazon EC2[16], que va permetre al NYT disposar d'un gran nombre de màquines per un curt període de temps.

A l'abril de 2008 Hadoop es va convertir en el sistema més ràpid en ordenar 1 TB de dades. Emprant un clúster de 910 nodes va acabar la tasca en 209 segons (una mica menys de 3 minuts i mig), superant la marca anterior de 297 segons[17]. Al novempre del mateix any, Google va emprar MapReduce per realitzar l'ordenació en 68 segons[18] i més endavant, a l'abril de 2009, Yahooo! ho va aconseguir en 62 segons[19].

La competició d'ordenació que van mantenir Yahoo! i Google durant aquesta època va donar peu a una tendència on les empreses ordenaven volums creixents de dades cada vegada més ràpidament. Al 2014 per exemple, un equip de Databricks[20] va ordenar 100 TB de dades en 1.406 segons emprant una agrupació de 207 nodes Spark (a un ritme de 4,27 TB per minut)[21].

El nom Hadoop prové de com va anomenar el fill de Doug Cutting al seu elefant de peluix. A l'ecosistema Hadoop sovint trobam que els projectes tenen un nom que no està relacionat amb la seva funció (com per exemple el projecte Pig[22]), mentre que les components tenen un nom més descriptiu(*Yet Another Resource Negotiator* (YARN), *Hadoop Distributed File System* (HDFS)).

3.1.2 Ecosistema Hadoop.

L'ecosistema Hadoop consta de molts projectes i components. Pel desenvolupament d'aquest treball emprarem únicament YARN i HDFS.

HDFS.

HDFS és un sistema de fitxers dissenyat per emmagatzemar arxius a un sistema de computació distribuïda.

A HDFS els arxius es troben dividits en particions de la mida d'un bloc (típicament 128MB), tot i que si l'arxiu és petit o un bocí ocupa menys d'aquesta mida, no queda la resta d'espai inutilitzat.

[1] Podeu trobar la llista completa a https://wiki.apache.org/hadoop/PoweredBy

La mida d'un bloc pot semblar desmesurada si la comparam amb la d'un disc normal, que sol ser 512 bytes. A HDFS els blocs són grans perquè es vol minimitzar el cost de les cerques. Si la mida del bloc és suficientment gran, el temps que suposarà transferir les dades serà considerablement superior al temps de posicionar el capçal. Per tant, transferir un fitxer gran (compost de múltiples blocs) tardarà aproximadament la mida del fitxer multiplicat pel rati de transferència, fent negligible el temps de cerca[4].

Aplicat a un sistema de fitxers distribuït, l'abstracció en blocs té els següents avantatges:

1. Un arxiu pot ocupar més de l'espai disponible a un únic disc dur. Com que l'arxiu es trobarà dividit, podrà estar emmagatzemat a discs diferents.
2. Els blocs proporcionen una unitat sobre la que realitzar la replicació de dades. Normalment, cada bloc es trobarà emmagatzemat a tres màquines diferents, fent possible que en cas de fallida es pugui recuperar la informació. A la figura 3.1 es pot veure com es replicarien les diferents parts d'un arxiu que ocupa 5 blocs.

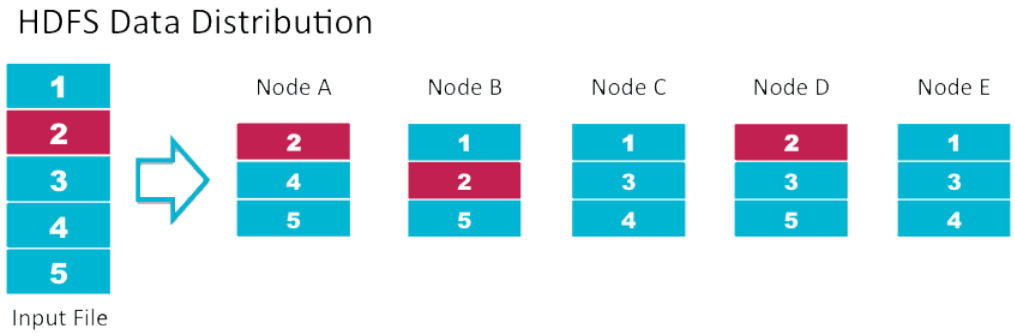

Figura 3.1: Rèplica de dades dins HDFS (Extreta de [3])

Per tal d'organitzar el sistema d'arxius, HDFS té dos tipus de nodes: *namenodes* i *datanodes*:

- Els *namenodes* mantenen l'arbre de directoris i les metadades dels fitxers i directoris de l'arbre. Sense el *namenode* el sistema no pot ser emprat ja que es perd el registre d'on es troba cada part de la informació.
- Els *datanodes* és allà on les dades estan emmagatzemades físicament i la seva funció és enviar i rebre dades.

YARN.

YARN és un sistema de gestió de recursos dissenyat per treballar amb Hadoop, però està programat de forma tan general que serveix per a qualsevol sistema de computació distribuïda.

YARN proporciona una *Application Programming Interface* (API) que permet al programador treballar amb els recursos dels diferents nuclis, tot i que quasi mai s'empra de forma explícita. El més normal és emprar *frameworks* d'alt nivell que estan construïts implementant YARN (com Hadoop o Spark) sense haver-se de preocupar per la gestió dels recursos.

Modes de funcionament.

Per tal de realitzar proves sobre las aplicacions que desenvolupem amb Hadoop, podem executar el codi de tres formes diferents:

- **Mode local:** No hi ha cap servei en marxa, el nostre programa s'executa dins la màquina virtual de Java local. Facilita la depuració de tasques MapReduce i s'empra principalment a les primeres fases de desenvolupament.
- **Mode pseudo-distribuït:** Els serveis de Hadoop s'executen en local. Permet simular un clúster d'un sol node.
- **Mode distribuït:** Els serveis de Hadoop s'executen damunt un grup d'ordinadors.

El codi de totes les tasques és molt similar, només canvia la forma de proporcionar els fitxers d'entrada i el programa principal. Per desenvolupar aquest treball s'emprarà el mode local i el mode pseudo-distribuït.

3.1.3 MapReduce.

MapReduce és un model de computació on les dades es tracten en forma de parelles clau-valor. Cada treball MapReduce està format per dues tasques que li donen nom: la tasca de Map i la de Reduce.

La tasca de Map reb les dades d'entrada i les transforma en un conjunt clau-valor intermedi. Una vegada processades, la llibreria MapReduce agrupa tots els valors intermedis que tenen la mateixa clau i els envia a la tasca Reduce.

La tasca de Reduce pren com a paràmetres una clau amb la llista de valors associats i els processa.

Quan Google va introduir el model l'any 2004[13] va posar el següent exemple:

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
```

```
    result += ParseInt(v);
Emit(key, AsString(result));
```

Aquest pseudo-codi representa una tasca MapReduce que compta el nombre de vegades que apareix cada paraula a una col·lecció de documents. Com es pot veure el codi que ha d'escriure el programador és molt clar i directe, ja que la llibreria MapReduce s'encarrega del flux de dades i la paral·lelització de tasques.

El mètode `map` emet una parella clau-valor intermèdia (paraula, comptador), en aquest cas "1". Si una paraula *w* apareix més d'una vegada en un document la funció de map generarà dues vegades (*w*, "1").

El mètode `reduce` reb cada clau amb la llista de valors que té associats, per exemple (*w*, ["1", "1"]) i els suma per obtenir quantes vegades ha aparegut una paraula. Es pot veure el flux de dades del treball MapReduce a la figura 3.2:

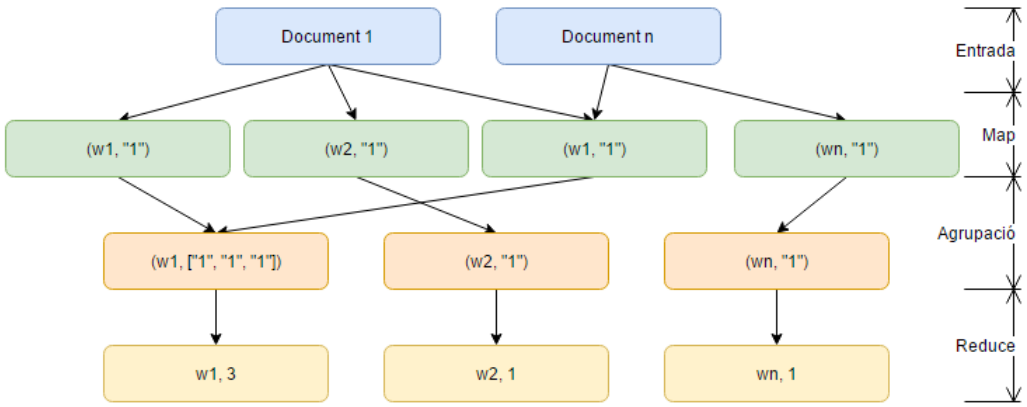

Figura 3.2: Flux de dades de l'exemple (Elaboració pròpia).

3.1.4 Exemples de tasques MapReduce.

Hi ha moltes tasques que es poden expressar emprant el model MapReduce, entre les que podem destacar[13]:

- **Comptador de la freqüència d'accés a una URL:** El mètode `map` processa els registres del servidor emetent parelles de l'estil (URL, 1) i el mètode de `reduce` les agrupa per obtenir (URL, accessos totals). D'aquesta forma sabrem quines pàgines son les més visitades.

- **Índex invers:** Podem processar un conjunt de documents de forma que el mètode `map` generi parelles de l'estil (paraula, ID document) i després el mètode `reduce` les agrupi en (paraula, llista(ID document)). D'aquesta forma donada una paraula sabem a tots els documents als que apareix.

3.2 Per què Hadoop?

Moltes empreses empren aquest *framework* per tractar tasques delicades, com Yahoo!, Last.fm, Facebook i New York Times tal i com s'ha esmentat a l'apartat anterior. Això és degut als avantatges que proporciona, que es poden resumir en cinc[23]:

1. **Escalable**: Distribueix les dades a analitzar entre distints servidors, permetent el processament en paral·lel.
2. **Econòmic**: Es pot considerar que Hadoop és econòmic des de dos punts de vista[24]:
   2.1. Un node de Hadoop només requereix un processador, una targeta de xarxa i un disc dur.
   2.2. Tradicionalment els negocis s'havien vist obligats a sel·leccionar quines dades es guardaven i quines s'eliminaven perquè resultava impossible guardar-les totes. Això suposava que si el negoci canviava i es necessitaven dades que no s'havien guardat, no es podia fer res i la companyia perdia temps i diners. Però amb Hadoop i HDFS es pot emmagatzemar tota la informació que faci falta.
3. **Flexible**: Es pot emprar per una gran varietat de tasques, com processament d'arxius, emmagatzematge de dades, anàlisi de mercat...
4. **Ràpid**: El processament de les dades es fa al mateix node on es troben emmagatzemades i només es transmeten els resultats.
5. **Tolerant a errors**: Com que les dades d'un node es troben replicades, en cas d'error sempre podrem recuperar-les a partir de les còpies.

3.3 Comparativa amb altres sistemes.

Hadoop no és el primer *framework* de computació distribuïda que permet l'emmagatzematge i anàlisi de dades, però sí que té unes característiques pròpies que el diferencien de qualsevol altre.

3.3.1 Hadoop comparat amb els sistemes de bases de dades relacionals.

Quan la quantitat de dades a tractar és petita, amb una base de dades tradicional i discs de capacitat suficient podem realitzar l'anàlisi sense emprar Hadoop.

Però a l'hora d'escalar emprant aquest sistema trobam que així com creix la mida del disc el temps de cerca[2] va millorant molt més lentament que el rati de transferència[3], de forma que si el patró d'accés es veu dominat per les cerques tardarem més en llegir un fitxer gran que en transmetre la informació que conté. Això és important perquè les

[2] El temps de cerca (t_s) és el temps que tarda el capçal del disc a moure's fins on volem llegir o escriure.
[3] El rati de transferència és la velocitat a la que el disc transmet les dades.

bases de dades tradicionals funcionen amb estructures de tipus arbre B que es veuen limitades per la velocitat a la que el disc realitza les cerques.

Per aquest motiu, si la nostra aplicació necessita realitzar moltes lectures i una sola escriptura, ens serà més útil el model MapReduce de Hadoop, mentre que si necessitam llegir només part de la informació i actualitzar les dades parcialment, ens aniran millor les *querys* d'un *Relational Database Management System* (RDBMS).

Una altra diferència entre els dos sistemes és com es troben estructurades les dades: a una base de dades relacional estan organitzades en taules o fitxers XML mentre que Hadoop tracta majoritàriament amb dades semi-estructurades (text, imatges, fulls de càlcul...). És per aquest motiu que disposar d'un model dinàmic torna encara més important (veure secció 2.5).

Però poc a poc les diferències entre una base de dades relacional i Hadoop es van fent més petites, ja que es van agafant idees mútuament. Per exemple, sistemes com Hive estan afegint característiques d'un RDBMS, com la possibilitat de crear índexs i realitzar transaccions.

3.3.2 Hadoop comparat amb la computació en malla.

La computació en malla és un tipus de computació distribuïda on els diferents nodes que hi participen poden disposar de recursos o arquitectures diferents. Està especialment dissenyada per distribuir tasques entre màquines que comparteixen un sistema de fitxers centralitzat i es comuniquen mitjançant l'enviament de missatges.

Mentre tots els nuclis es trobin funcionant constantment resulta una tècnica molt eficient per tasques que requereixen un anàlisi intensiu des del punt de vista de la CPU, però a l'hora d'escalar l'anàlisi la transferència de dades passa a ser el coll de botella i començam a tenir nuclis inactius. Com que amb Hadoop sempre treballarem amb dades locals, això no passarà.

A més, *Message Passing Interface* (MPI) [25] destaca per donar als programadors molta llibertat per controlar el flux dels programes però requereix un tractament explícit que afegeix dificultat a l'anàlisi. Per evitar-ho, Hadoop tracta el flux de dades de forma implícita i permet al programador pensar únicament en termes del model de dades.

També cal tenir en compte la tasca afegida de coordinar l'anàlisi i les fallides dels nodes, un problema recurrent en programació distribuïda. Controlar aquests dos punts suposa un codi més complexe quan s'empra MPI, però també es té més control sobre com actuar en cada cas. Amb Hadoop és trivial ja que les tasques Map que es duen a terme a cada node són independents i si alguna falla, s'executarà de nou al node on hi ha les dades replicades sense que el programador hagi d'intervenir.

3.3.3 Hadoop comparat amb la computació voluntària.

Un dels projectes més coneguts de la computació voluntària és SETI@home, on *Search for Extra-Terrestrial Intelligence* (SETI) envia fragments de dades a ordinadors de volun-

taris per a que les analitzin mentre estan inactius.

És a dir, es divideixen les dades a analitzar i es reparteixen per ser tractades en paral·lel i després els resultats s'envien al servidor.

Tot i que aquesta és la mateixa estratègia que empra Hadoop, la computació voluntària es centra en tasques on l'anàlisi és intens des del punt de vista de la CPU, no des del punt de vista del disc. A més, els projectes de computació voluntària s'executen a màquines compartides amb connexions i capacitats variables, mentre que Hadoop està dissenyat per executar-se a nuclis dedicats.

3.3.4 Hadoop comparat amb Spark.

Spark, desenvolupat per Apache Software Fundation, és el *framework* de Big Data més emprat juntament amb Hadoop. L'ús d'un o altre depèn de l'àmbit: Hadoop permet emmagatzemar grans quantitats d'informació de forma fiable i proporciona una plataforma de programació distribuïda per analitzar les dades, mentre que Spark és únicament per l'anàlisi d'informació a gran escala.

Els podem comparar des de cinc punts de vista[26]:

1. **Fiabilitat**: Un dels beneficis de Hadoop és que gràcies a HDFS és poc propens a fallides, permetent que les dades estiguin sempre disponibles. Spark és simplement per analitzar dades.
2. **Cost**: Els dos *frameworks* són propietat d'Apache Software Foundation i per tant són gratuïts i de codi obert. L'únic cost que suposen són el del hardware on s'executen i els recursos que se'ls destinen.
3. **Velocitat**: Spark s'anuncia com un *framework* 100 vegades més ràpid que Hadoop[27]. Això es deu principalment al plantejament, ja que Spark s'executa en memòria mentre que Hadoop va llegint i escrivint directament al disc.
4. **Universalitat**: Spark pot obtenir les dades emmagatzemades a qualsevol sistema, fins i tot de HDFS.
5. **Complexitat**: Els dos *frameworks* són difícils d'emprar i necessiten que gent formada s'hi dediqui si es volen treure bons resultats.

Fins aquests darrers anys Hadoop ha estat el *framework* més emprat per l'anàlisi de dades, però ara que Spark té una versió *cloud* el panorama està canviant. Aquesta nova funcionalitat obre moltes possibilitats a empreses de l'àmbit de l'aprenentatge automàtic o de l'*Internet of Things* (IoT) que fins ara no tenien una alternativa viable a Hadoop[28].

3.4 Exemple: Nota màxima de cada assignatura.

Per tal de tenir un exemple al que referir-nos, anem a suposar que disposam d'un registre on s'han anat guardant any rere any els alumnes que s'han matriculat a cada

assignatura i la nota que han tret. A partir d'aquest fitxer volem calcular la nota màxima de cada assignatura.

Amb aquest senzill exemple podrem veure com actua MapReduce i el podrem comprar amb altres eines d'anàlisi.

3.5 Estructura dels fitxers de dades.

Disposam de *n* fitxers, tants com siguin necessaris per emmagatzemar les notes.

```
Qualificacions1.gz
Qualificacions2.gz
Qualificacions3.gz
...
```

Cadascun dels fitxers contindrà múltiples línies, cada una amb les següents dades separades per tabuladors:

- Curs acadèmic.
- Codi del plà d'estudis.
- Nombre d'expedient de l'alumne.
- Codi de l'assignatura.
- Convocatòria.
- Data en que la qualificació és definitiva.
- Qualificació nominal (Excel·lent, Notable, ...).
- Qualificació numèrica.

Per exemple de la línia:

```
2011-12 GAEP 100 29259 JUN 10/07/2012 12:24:38 SU 3.5
```

significaria que al grau GAEP a la convocatòria de juny del curs 2011-12 de l'assignatura 29259 l'alumne amb nombre d'expedient 100 va suspendre amb un 3.5. La nota va ser definitiva dia 10-07-2012 a les 12:24:38. Si un estudiant no es presenta a l'examen, la qualificació nominal seria NP (No Presentat) i no hi hauria qualificació numèrica.

Les línies de cada fitxer no estan ordenades.

3.6 Anàlisi de les dades emprant eines Unix.

Si només volem saber la nota màxima de cada assignatura, podem trobar la resposta sense emprar Hadoop o cap altre *framework* de Big Data.

Una de les opcions que tenim per fer-ho és un *script* de Bash[29]:

```
#!/bin/bash

for file in input/*
do
    gunzip -c $file | \
        awk '{  split($0, line, "\t");
                scode  = line[4];
                status = line[7];
                mark   = line[8] + 0.0;
                if(status != "NP" && mark > max[scode]) {
                    max[scode] = mark;
                }
            }
            END {
                for(key in max) {
                    print key " " max[key];
                }
            }'
done
```

Aquest *script* iterarà per les línies de cada fitxer, agafant de cada una la qualificació nominal per veure si l'estudiant s'ha presentat a l'examen i si és així, comprovarà si supera la nota màxima provisional de l'assignatura. Al final s'imprimeix codi de cada assignatura acompanyat de la nota més alta.

Com que l'*script* és correcte, si disposam de temps suficient podrà iterar per tots els fitxers del directori mostrant cada assignatura amb la nota màxima. Però aquest temps serà massa elevat quan necessitem tractar grans quantitats de dades.

Per reduir el temps d'execució podríem estudiar quines parts del programa es poden paral·lelitzar. En aquest cas és evident que guanyaríem temps analitzant cada fitxer en un procés diferent, aprofitant tots els nuclis disponibles dels processadors. No obstant, això presenta problemes:

Primer, cal estudiar com dividir la càrrega de treball entre processos. En cas de que el tamany dels fitxers variï molt trobarem processos que acaben l'anàlisi molt abans que altres. Encara que quan un procés acabi comenci a tractar un altre fitxer, el fitxer més gran serà el coll de botella de l'anàlisi. Una millor aproximació a la solució seria dividir l'*input* en fitxers d'igual tamany.

Segon, cal tenir en compte que combinar els resultats dels diferents processos suposarà una càrrega addicional. En el nostre cas com que les assignatures son independents entre sí, combinar els resultats seria senzill.

Encara que aconseguim resoldre tots els problemes anteriors, ens seguim trobant

limitats per la potència de la màquina. Si a més el nostre conjunt de dades excedeix la memòria d'un ordinador i n'hem de combinar un parell per realitzar l'anàlisi, ens hem d'enfrontar també els problemes de la programació distribuïda (coordinar les màquines, tractament de fallides, ...).

Per tant podem concloure que tot i que seria possible realitzar l'anàlisi d'aquesta forma, seria un procés molt laboriós i difícilment escalable. Aquí és on entren en joc *frameworks* com Hadoop, que permetran encarregar-nos del problema sense les distraccions d'alguns aspectes d'implementació.

3.7 Anàlisi de les dades emprant Hadoop.

Abans d'entrar en matèria cal formalitzar un parell de conceptes que s'han emprat en apartats anteriors però encara no hem definit:

- Un **treball de MapReduce** és una unitat de treball que està formada per un conjunt de dades d'entrada, el codi d'un programa MapReduce i les opcions de configuració.
- Cada treball MapReduce es troba dividit en **tasques**, que poden ser de dos tipus: tasques Map i tasques Reduce. Aquestes tasques estan programades per executar-se a un conjunt de nodes i si alguna falla, s'envia a un nou node per a tornar a ser executada.
- Hadoop divideix el conjunt de dades d'entrada en **particions**, i a cada una se li assigna una tasca de Map.

3.7.1 MapReduce.

La tasca de Map rebrà el conjunt de dades a analitzar en forma de parelles clau-valor, on la clau serà el nombre de fila del fitxer i el valor serà la línia en sí. Per exemple, suposant que la línia
`2011-12 GAEP 100 29259 JUN 10/07/2012 12:24:38 SU 3.5`
és la primera del fitxer, la tasca de Map rebria:
`(0, 2011-12 GAEP 100 29259 JUN 10/07/2012 12:24:38 SU 3.5)`

D'aquí la tasca de Map ha d'extreure la informació necessària i convertir-la en una nova parella clau-valor que pugui ser processada per la tasca de Reduce. La nova clau i el valor associat són triats pel programador segons la naturalesa de l'anàlisi. En aquest cas podríem agafar el codi de l'assignatura com a clau i la nota com a valor, de forma que la sortida seria "`(29259, 3.5)`".

Una vegada s'ha aplicat Map a totes les línies del fitxer, tenim un conjunt de parelles de l'estil:

```
(29259, 3.5)
```

```
(29259, 7)
(17935, 9.1)
(29259, 2)
...
```

Abans de ser enviat al Reduce, Hadoop processa aquestes parelles agrupant-les segons la clau:

```
(29259, [3.5, 7, 2])
(17935, [9.1])
...
```

Després, la tasca de Reduce iterarà pels valors associats a cada clau per obtenir el resultat. En el nostre cas, el que hauria de fer és trobar el màxim de la llista de notes per cada codi d'assignatura:

```
(29259, 7)
(17935, 9.1)
```

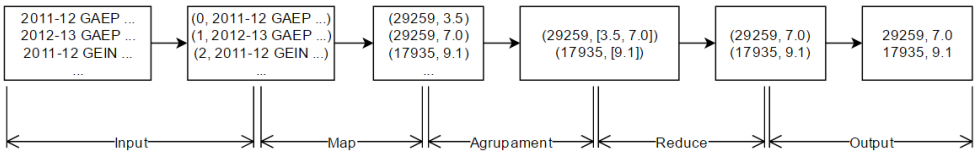

Figura 3.3: Esquema del funcionament de MapReduce (Elaboració pròpia).

3.7.2 MapReduce en Java.

Expressar la idea de MapReduce en forma d'algoritme és directe, només necessitam programar el mètode de `map` i el de `reduce`.

Codi de la tasca de Map:

```
import java.io.IOException;

import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxAssignaturaMapper
    extends Mapper<LongWritable, Text, Text, DoubleWritable> {
```

```
    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String[] line = value.toString().split("\t");
        String scode  = line[3];
        String status = line[6];

        if (!status.equals("NP")) {
            double mark = Double.parseDouble(line[7]);
            context.write(new Text(scode), new DoubleWritable(mark));
        }
    }
}
```

La tasca de Map s'ha de situar dins el mètode `map` d'una classe que hereti de la classe Mapper. Com que la classe pare és genèrica podrem indicar les entrades i sortides que siguin més adequades pel nostre programa. Per exemple, `extends Mapper<LongWritable, Text, Text, DoubleWritable>` significa que l'entrada és una parella clau-valor on la clau és de tipus LongWritable i el valor és de tipus Text, mentre que la sortida és una altra parella clau-valor on la clau és Text i el valor un DoubleWritable.

Els tipus `LongWritable`, `DoubleWritable` i `Text` estan definits per Hadoop i són similars al `long`, `double` i `String` de Java, però s'han optimitzat per funcionar millor durant les transferències de dades.

La nostra funció rebrà el nombre de línia amb el contingut de la línia més un context, que és on es guardaran els resultats. Com ja s'ha mencionat, la clau que reb la funció és el nombre de línia del fitxer on llegim i el valor serà el text de la línia. Per tant, l'únic que hem de fer és dividir la línia en atributs, comprovar que l'alumne s'hagi presentat a l'exàmen i si ha estat així escriure al context el codi de l'assignatura amb la nota de l'estudiant.

Una vegada el mètode `map` s'hagi aplicat a tot el fitxer acabarà la fase de Map i començarà la de Reduce:

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxAssignaturaReducer
    extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values,
        Context context) throws IOException, InterruptedException {

        double max = -1;
        for (DoubleWritable value : values) {
```

```
            max = Math.max(max, value.get());
        }
        context.write(key, new DoubleWritable(max));
    }
}
```

De forma similar a com hem fet per Map, hem de posar el codi dins el mètode `reduce` d'una classe que hereti de Reducer. En aquest cas es rebrà una parella Text-DoubleWritable (la sortida de la funció `map`) i emetrà una altra parella Text-DoubleWritable que seran els resultats.

L'únic que ha de fer la funció és iterar per tots els valors d'una clau —totes les notes d'una assignatura— i trobar-ne el màxim.

Finalment cal declarar el treball de Hadoop al programa principal i indicar-li d'on agafar les dades, on guardar-les i quines són les funcions de `map` i `reduce`:

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxAssignatura {

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Error en l'especificació " +
                "d'arguments. Crideu el programa com:");
            System.err.println("\t hadoop MaxAssignatura <input path>" +
                "<output path>");
            System.err.println("\t Recordeu especificar " +
                "HADOOP_CLASSPATH");
            System.exit(-1);
        }

        Job job = new Job();
        job.setJarByClass(MaxAssignatura.class);
        job.setJobName("Max assignatura");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(MaxAssignaturaMapper.class);
        job.setReducerClass(MaxAssignaturaReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Una vegada compilat cal definir la variable d'entorn `HADOOP_CLASSPATH` i després invocar l'executable de Hadoop de la següent forma: `hadoop <Nom del fitxer class (sense extensió)> <fitxer o directori d'entrada> <directori de sortida>`

```
export HADOOP_CLASSPATH=exemple02.jar

hadoop MaxAssignatures input/qualificacio.txt output
```

Si aquestes comandes s'executen correctament Hadoop s'iniciarà i ens informarà per consola del progrés de l'anàlisi. Per exemple, ens dirà quantes tasques Map i Reduce han fet falta, la ID que els hi ha donat i algunes estadístiques de rendiment. A més, si en acabar consultam el directori `output`, veurem que hi ha un fitxer que conté els resultats.

```
cat output/part-r-00000

20100 10.0

20101 9.2

20102 7.6

20103 9.4

...
```

Si comparam els continguts del fitxer amb amb els obtinguts amb Bash, veurem que són idèntics.

3.8 Consideracions a l'hora d'escalar el nostre anàlisi.

En aquest exemple hem pogut veure com podem emprar Hadoop per analitzar quantitats petites de dades a una sola màquina, però que hem de tenir en compte a l'hora d'escalar l'anàlisi?

3.8.1 Flux de dades.

El primer que cal decidir a l'hora de preparar un treball de MapReduce distribuït és la mida de les particions d'entrada. El més comú és que cada partició tengui la mida d'un bloc HDFS (per defecte 128 MB). Això assegura que la partició sencera es troba emmagatzemada de forma local al node que l'ha de tractar i que no s'ha d'ocupar part de l'ample de banda per transferir dades.

Quan s'ha acabat una tasca de Map, el resultat es guarda dins el disc de la màquina hoste i no dins HDFS. Això és degut a que aquest resultat només és una passa intermèdia que servirà d'entrada a la tasca de Reduce, i tenir-la guardada dins HDFS provocaria que es repliqués i no és necessari. Si el Map falla abans de completar-se, es tornarà a executar a un altre node i es seguirà l'anàlisi.

Una vegada acabades les tasques de Map, els resultats obtinguts s'empren com entrada de la tasca de Reduce. Com que hi ha menys tasques Reduce que Map, cal enviar a través de la xarxa els resultats intermedis als nuclis on s'executin les tasques Reduce.

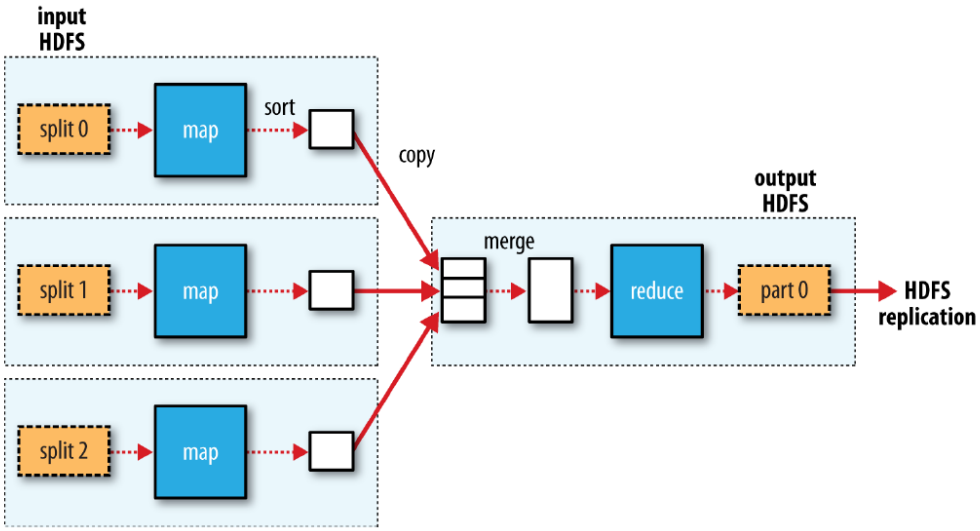

Figura 3.4: Flux de dades a un sistema Hadoop amb una única tasca de Reduce (Extreta de [4]).

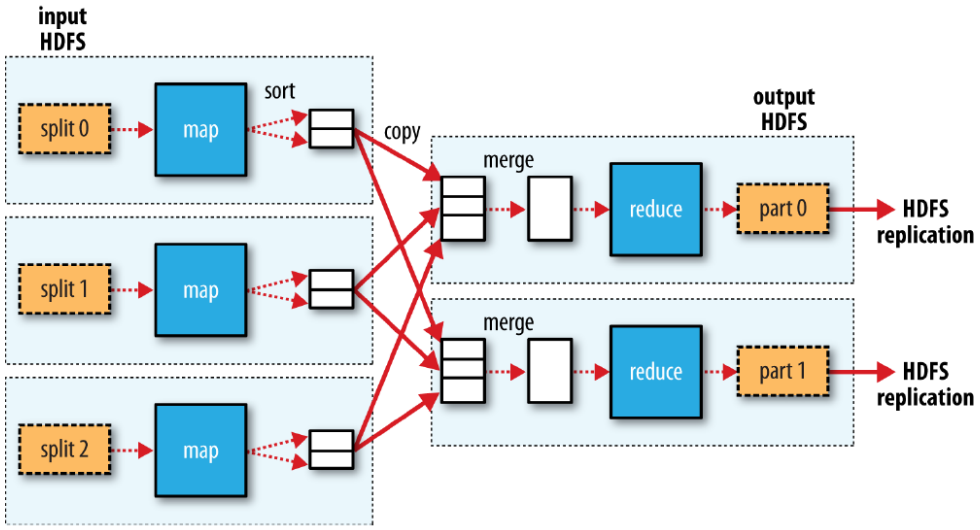

Figura 3.5: Flux de dades a un sistema Hadoop amb més d'una tasca Reduce (Extreta de [4]).

A la figura 3.4 podem veure com seria el flux de dades si només hi ha una tasca

Reduce. Si n'hi ha més d'una, llavors el flux seria com el de la figura 3.5.

3.8.2 Combinadors.

Quan l'ample de banda suposa el coll de botella del nostre sistema i necessitam reduir la quantitat d'informació que envien les tasques de Map quan acaben, pot resultar interessant implementar un combinador. Un combinador és una funció que s'executa abans d'enviar les dades a la tasca de Reduce i redueix el tràfic de dades per la xarxa.

Suposem que el fitxer que hem emprat per fer l'exemple resulta molt gran i l'hem de repartir entre dues tasques de Map de forma que els registres de l'assignatura 23853 no es tracten tots al mateix nucli.

Un map ens donarà unes quantes notes:

```
(23853, 7.5)

(23853, 10)

(23853, 4.5)
```

I l'altre map, la resta:

```
(23853, 3)

(23853, 8)
```

La tasca de Reduce rebria:

```
(23853, [7.5, 10, 4.5, 3, 8])
```

I ens donaria el resultat final de `(23853, 10)`, ja que 10 és la nota màxima de l'assignatura.

Per reduir la transferència de dades es pot programar un combinador que directament agafi el màxim de cada assignatura en acabar un Map. Així el primer Map per l'assignatura 23853 només produiria `(23853, 10)` i el segon, `(23853, 8)`. La tasca de Reduce rebria `(23853, [10, 8])` i guanyaríem tant en temps de transferència de dades com en temps d'execució del *reducer*.

Implementar un combinador en Java és senzill. Degut a les similituds amb una tasca de Reduce, un combinador també hereta de `Reducer`. És més, pel nostre exemple la tasca de Reduce i el combinador són idèntiques així que podem aprofitar el mateix codi. Afegir el combinador al nostre exemple és tan senzill com:

```
public class MaxAssignatura {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Error en l'especificació " +
        "d'arguments. Crideu el programa com:");
```

```
      System.err.println("\t hadoop MaxAssignatura <input path>" +
        "<output path>");
      System.err.println("\t Recordeu especificar " +
        "HADOOP_CLASSPATH");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxAssignatura.class);
    job.setJobName("Max assignatura");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxAssignaturaMapper.class);
    job.setCombinerClass(MaxAssignaturaReducer.class); // Única línia diferent
    job.setReducerClass(MaxAssignaturaReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(DoubleWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

CAPÍTOL 4

APLICACIÓ DE HADOOP DAMUNT DADES ACADÈMIQUES.

4.1 Estudi a realitzar.

Una vegada explicats els fonaments de Hadoop ha arribat el moment de realitzar un cas pràctic més gran on realment veurem els avantatges del *framework*. Un estudi que resulta interessant és veure el fracàs que hi ha al primer curs de cada pla d'estudis en funció de la nota d'accés. Per exemple, quin percentatge d'estudiants amb un notable a les Proves d'accés a la Universitat (PAU) que comencen el grau en enginyeria informàtica no acaben primer.

El resultat d'aquest estudi serà una gràfica on es representarà la relació entre fracàs i nota d'accés en funció del pla d'estudis, de forma que es puguin comparar els diferents estudis de l'universitat.

4.2 Dades disponibles.

Per tal de dur a terme l'estudi disposam d'un conjunt de dades distribuïdes en fitxers on cada línia és un registre i els atributs estan separats per tabuladors. Hi ha 7 fitxers:

- **Accés**: 42.572 registres.
- **Pla d'estudis de grau**: 71 registres.
- **Expedient**: 33.690 registres.
- **Matrícula**: 145.151 registres.
- **Assignatura**: 955.471 registres.

- **Qualificació**: 1.249.599 registres.

- **Assoliment de titulació**: 20.166 registres.

4.3 Procediment.

Abans de començar l'anàlisi dels fitxers cal definir el resultat que volem obtenir i quines dades necessitam per arribar-hi.

Podem definir l'índex de fracàs d'un grau com

$$1 - \frac{nombre\,d'estudiants\,que\,han\,passat\,primer}{nombre\,d'estudiants\,matriculats}$$

on el nombre d'estudiants matriculats es pot extreure del fitxer de matrícules. El nombre d'estudiants que han passat primer resulta una mica més complicat de calcular ja que no sabem el curs al que pertany una assignatura. A més, als nous plans d'estudis hi pot haver estudiants cursant matèries de segon sense haver aprovat totes les de primer.

Això ens duu a definir que és "passar primer curs". Per senzillesa direm que un estudiant passa primer quan ha aprovat 10 assignatures d'un pla d'estudis. Encara que les assignatures siguin mesclades de primer i segon, el fet d'haver aprovat matèries de segon curs vol dir que no ha abandonat.

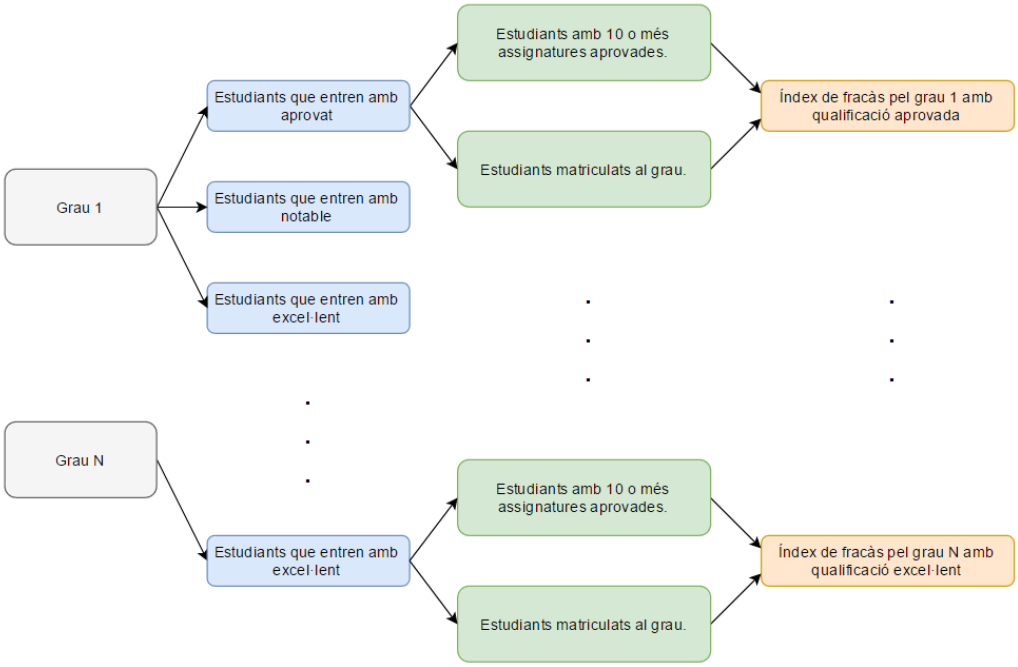

Figura 4.1: Esquema de l'anàlisi (Elaboració pròpia).

Aquest càlcul l'haurem de realitzar per cada pla d'estudis i cada qualificació d'accés (aprovat, notable i excel·lent) com podem veure a la figura 4.1. El resultat seran n índexs, on $n = nombre\,de\,plans\,d'estudis * 3$.

Les qualificacions d'accés es transformaran en aprovat, notable o excel·lent perquè d'aquesta forma disposarem de més dades per cada índex i els càlculs seran més clars.

4.3.1 Tractament de registres incomplets.

Dins els fitxers de dades podem trobar registres als que falten atributs, però com que són pocs s'ha decidit ignorar-los. No obstant, si es troba que molts registres d'un mateix pla d'estudis són erronis aquest no es contemplarà a l'hora de representar els resultats.

Aquest cas només es dona al pla Grau d'Infermeria (Pla 2016) (GIF2).

4.3.2 Tractament de valors atípics.

No s'aplicarà cap tractament especial als valors atípics ja que són interessants per extreure conclusions.

El que si es farà a l'hora de representar els resultats és filtrar els graus amb pocs registres associats de forma que només ens quedem amb els resultats més fiables.

4.4 Programant la solució.

El primer que cal fer és veure quin és el mínim de fitxers necessaris per extreure tota la informació. Per aquest estudi són dos: del fitxer "accés" extraurem les notes dels estudiants a les PAU i de "qualificació" podrem agafar quantes assignatures ha aprovat un estudiant a un pla d'estudis. L'estructura d'aquests fitxers és la següent:

- **Accés**:
  - ID alumne (dada dissociada).
  - Curs acadèmic.
  - Convocatòria (mes del curs acadèmic).
  - Modalitat de prova d'accés.
  - Qualificació de l'accés (apte / no apte / no presentat).
  - Qualificació de la prova realitzada.
  - Qualificació final de l'accés.
- **Qualificació assignatura**:
  - Curs acadèmic que l'alumne inicia els estudis.
  - Codi pla d'estudis.
  - Nombre d'expedient de l'alumne.
  - Codi de l'assignatura matriculada.

- Convocatòria (mes del curs acadèmic).
- Data en que la qualificació és definitiva.
- Qualificació alfanumèrica.
- Nota numèrica de la qualificació.

A continuació s'ha de decidir quins atributs s'empraran de clau i quin serà el valor associat. És evident que farà falta més d'un treball MapReduce, ja que primer hem d'agrupar els registres de qualificacions en estudiants i després els estudiants en plans d'estudis tal i com s'il·lustra a la figura 4.2.

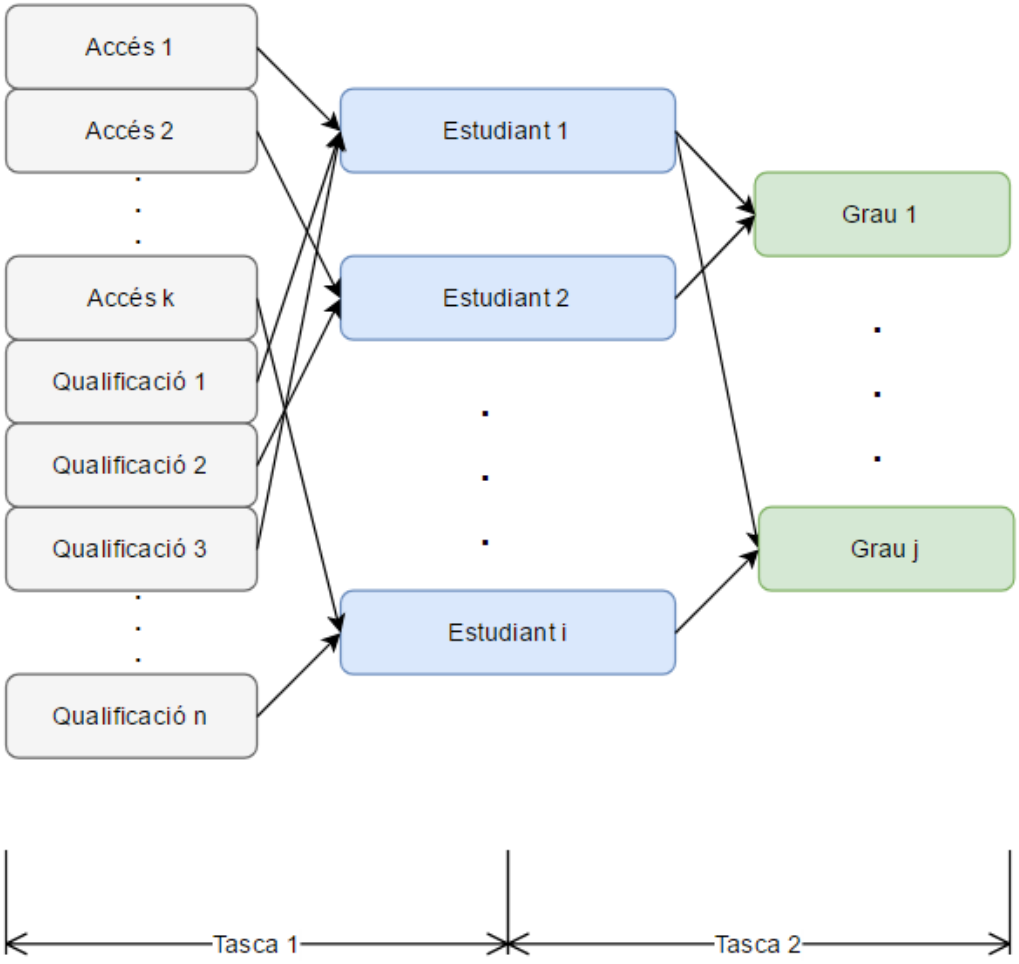

Figura 4.2: Distribució de les tasques MapReduce (Elaboració pròpia).

Pel que fa a les claus, per les qualificacions i les notes d'accés no ens en fa falta cap ja que al ser l'entrada del mètode `map` Hadoop assignarà al contingut de la línia el nombre de línia com a clau. Pels estudiants emprarem l'identificador d'estudiant i pels plans d'estudis, el codi que tenen assignat.

En el cas dels estudiants es podria haver emprat el nombre d'expedient ja que és únic dins un grau. És a dir, dins la universitat hi pot haver més d'un estudiant amb el mateix nombre d'expedient sempre que cursin graus diferents. No es fa així perquè quan un estudiant realitza les PAU no se li assigna un nombre d'expedient.

Tenir els registres d'accés i de qualificació amb la mateixa clau —identificador d'estudiant— ens permet tractar-los en funcions de `map` diferents però amb un *reducer* comú.

4.4.1 Primera tasca MapReduce.

El primer treball MapReduce extreu dels fitxers d'accés i de qualificacions quins estudiants matriculats a cada pla d'estudis amb una determinada nota d'accés han superat el primer curs.

Per fer-ho es farà ús de la classe `MultipleInputs`, que permet definir un *mapper* diferent per cada fitxer d'entrada. Els dos mètodes de `map` són senzills i directes:

Mètode de `map` per qualificacions:

```
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

  String[] line = value.toString().split("\t");
  String plaEstudis  = line[1];
  String idEstudiant = line[2];
  String codiAssign  = line[3];
  String aprovat     = line[6];
  if(!aprovat.equals("SU") && !aprovat.equals("NP")){
    context.write(new Text(idEstudiant), new Text(plaEstudis +"\t"+ codiAssign));
  }
}
```

En cas de que l'estudiant hagi aprovat l'assignatura, es produirà un *output* de l'estil (id de l'estudiant, codi del pla d'estudis + codi de l'assignatura aprovada).

Mètode de `map` per notes d'accés:

```
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

  String[] linia = value.toString().split("\t");
  String any = linia[1];
  if(linia.length == 7 && !any.equals("2015-16")) {
    String idEstudiant = linia[0];
    String nota = linia[6];
    context.write(new Text(idEstudiant), new Text(nota));
  }
}
```

Tractar les notes d'accés també és senzill, ja que l'únic que s'ha de fer és mirar si l'estudiant ha aprovat les PAU i si és així emetre un *output* de l'estil (id de l'estudiant,

nota d'accés). Eliminam les notes del curs anterior perquè són d'estudiants que ara estan cursant primer i encara no podem saber si aprovaran o no.

Noteu que la nota d'accés es guarda com a `Text` i no com a `Float`. Tot i que això pot semblar ineficient, a l'hora de transferir dades hi ha tant d'*overhead* que l'ús de Text no suposarà un augment notable del temps d'execució.

Una vegada les dues funcions de `map` han processat els seus fitxers, el *reducer* rebrà per cada clau una llista d'elements on cada un pot ser la nota d'una assignatura o una nota d'accés, de forma similar al que es veu a la figura 4.3.

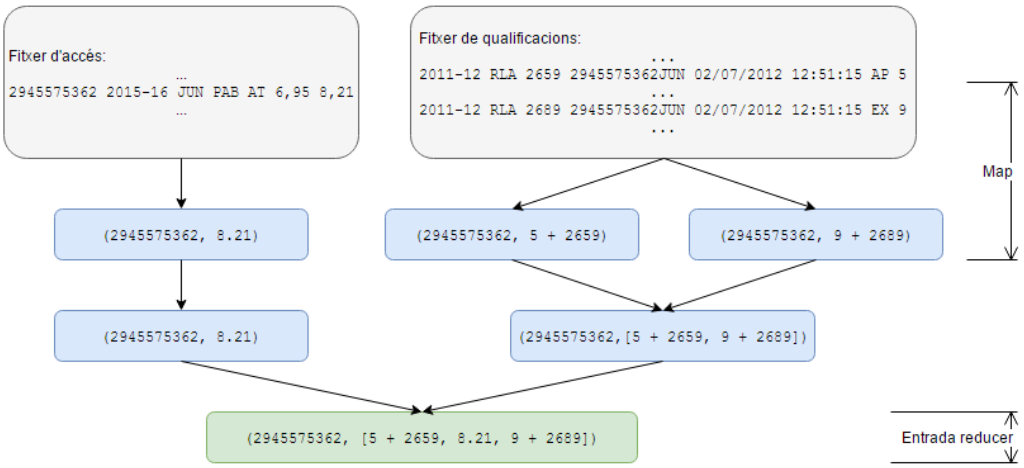

Figura 4.3: Flux de dades entre les funcions de `map` i `reduce` de la primera tasca (Elaboració pròpia).

Arribats a aquest punt tenim tres possibilitats:

- Un estudiant que ha realitzat les proves d'accés va a estudiar fora. En aquest cas tendrem un registre d'accés sense cap nota d'assignatura associada.
- No disposam de la nota d'accés d'un estudiant. Com que no disposam de les dades anteriors al curs 2009-10, hi pot haver estudiants cursant assignatures dels que no disposem de nota d'accés.
- Un estudiant té associada una nota d'accés i una llista d'assignatures.

La nostra funció de `reduce` ha de comprovar en quins dels tres casos ens trobam i només generar *output* pel tercer. El codi és el següent:

```
public void reduce(Text key, Iterable<Text> values, Context context)
    throws IOException, InterruptedException {
  double notaAcces = 0.0f;
  Map<String, Integer> assignaturesGrau = new HashMap<String, Integer>();

  for (Text value : values) {
```

```
    String[] items = value.toString().split("\t");
    if(items.length == 1){
      try {
        NumberFormat format = NumberFormat.getInstance(Locale.FRANCE);
        Number nombre = format.parse(items[0]);
        notaAcces = nombre.doubleValue();
      } catch (Exception e ) {}
    } else {
      String plaEstudis = items[0];
      if(assignaturesGrau.get(plaEstudis) != null) {
        assignaturesGrau.put(plaEstudis,assignaturesGrau.get(plaEstudis)+1);
      } else {
        assignaturesGrau.put(plaEstudis, 1);
      }
    }
  }
  if(notaAcces > 0.0f) {
    for(Map.Entry<String, Integer> pla : assignaturesGrau.entrySet()) {
      String grau = pla.getKey();
      int nAssignatures = pla.getValue();

      if(nAssignatures >= 10) { // Ha acabat el primer curs
        context.write(key, new Text(notaAcces + "\t" + grau + "\t" + "1"));
      } else {
        context.write(key, new Text(notaAcces + "\t" + grau + "\t" + "0"));
      }
    }
  }
}
```

La funció es divideix en dues parts: la primera recorre la llista de valors associats a la clau i enregistra la nota d'accés i el nombre d'assignatures aprovades a cada grau, mentre que la segona mira si s'ha trobat nota d'accés i si l'estudiant ha aprovat primer curs de cada grau al que s'ha matriculat.

Finalment tenim el programa principal, similar al del primer exemple amb la diferència de que s'empra la classe `MultipleInputs` per tractar a la vegada el fitxer d'accés i el de qualificacons.

```
public class Final1 extends Configured implements Tool {
  public int run(String[] args) throws Exception {
    Job job;
    if(args.length != 3){
      System.err.println("Error en l'especificació d'arguments.");
    }
    job = new Job();
    job.setJarByClass(Final1.class);
    job.setJobName("Recompte d'assignatures i notes d'acces");

    MultipleInputs.addInputPath(job, new Path(args[0]),
    TextInputFormat.class, AMapper.class);
    MultipleInputs.addInputPath(job, new Path(args[1]),
    TextInputFormat.class, QMapper.class);
    FileOutputFormat.setOutputPath(job, new Path(args[2]));

    job.setReducerClass(FirstReducer.class);
```

```
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new Final1(), args);
    System.exit(exitCode);
  }
}
```

Aquesta tasca MapReduce produirà una sortida de l'estil:

```
1000024166 5.868 GFIS 0
1000024166 5.868 GEAM 1
1001066981 5.316 GFIS 1
...
```

On els atributs són l'identificador d'estudiant, la nota d'accés, el codi del grau on s'ha matriculat i si ha aprovat o no el primer curs respectivament.

4.4.2 Segona tasca MapReduce.

A partir de l'*output* de la tasca anterior es pot calcular el fracàs als diferents graus de forma senzilla: només cal realitzar un recompte d'estudiants.

A més del recompte en aquesta tasca s'agruparan les notes d'accés en aprovat, notable i excel·lent de forma que hi hagi més mostres per cada grup.

També és necessari fer un canvi de clau ja que a partir d'ara treballarem amb graus. Com que el resultat que volem obtenir és un índex de fracàs per cada parella [nota d'accés, grau] emprarem aquests dos atributs com a clau doble.

L'única informació que s'haurà d'associar a la clau són els estudiants que han aprovat 10 assignatures, de forma que es puguin realitzar tots els càlculs. El flux d'informació es pot veure a la figura 4.4.

Com també es pot veure a la figura 4.4, la funció `reduce` emetrà un valor addicional: el nombre d'estudiants a partir dels que s'ha calculat l'índex. Aquesta dada servirà per filtrar els graus a l'hora de representar el resultats.

```
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

  String[] line = value.toString().split("\t");

  double val = Double.parseDouble(line[1]);
```

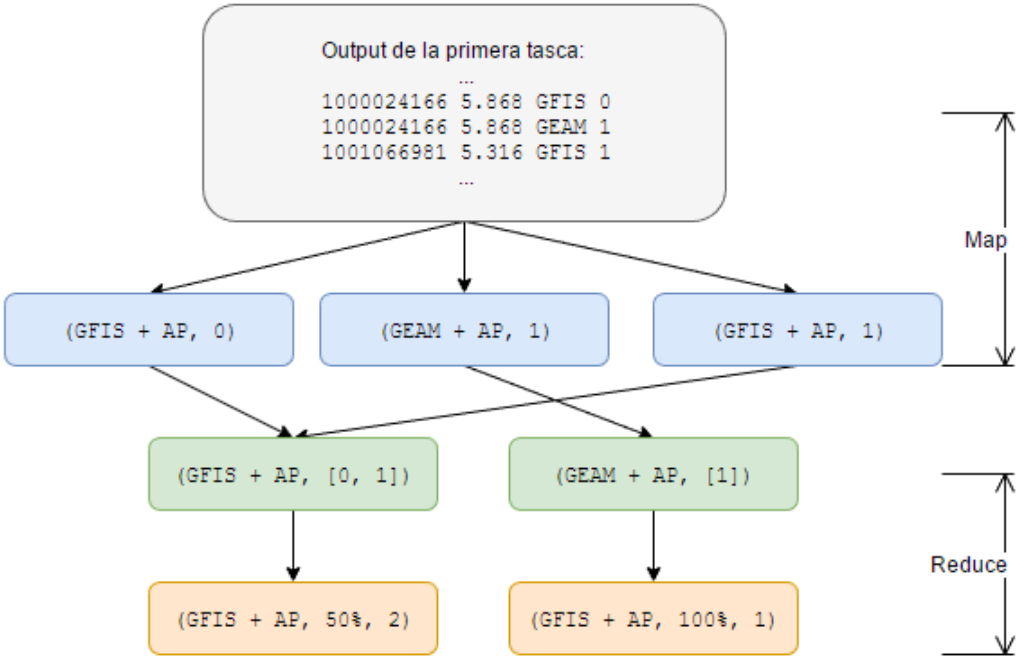

Figura 4.4: Flux de dades entre les funcions de `map` i `reduce` de la segona tasca (Elaboració pròpia).

```
  String qual;
  if(val >= 9.0f) {
    qual = "EX";
  } else if (val >= 7.0f) {
    qual = "NT";
  } else {
    qual = "AP";
  }
  String plaEstudis = line[2];
  String aprovat    = line[3];

  context.write(new Text(plaEstudis + "\t" + qual), new Text(aprovat));
}
```

El codi de la funció de `map` és molt senzill ja que només ha de transformar la qualificació d'accés i transmetre les dades al *reducer*.

```
public void reduce(Text key, Iterable<Text> values, Context context)
    throws IOException, InterruptedException {

  int totals = 0;
  int aprovats = 0;
  for (Text value : values) {
    if(value.toString().equals("1")) {
      aprovats++;
    }
    totals++;
```

```
  }

  float indexFracas = 1 - (float) aprovats / totals;
  context.write(key, new Text(indexFracas + "\t" + totals));
}
```

La funció de `reduce` també és simple ja que només ha d'iterar pels valors associats a cada clau —codi del grau i nota d'accés—, comptar quants estudiants han passat primer i aplicar la formula que apareix anteriorment en aquest capítol.

Finalment el codi del programa principal és exactament igual al vist al primer exemple:

```
public class Final2 extends Configured implements Tool {
  public int run(String[] args) throws Exception {
    Job job;

    if(args.length != 2){
      System.err.println("Error en l'especificació d'arguments.");
    }

    job = new Job();
    job.setJarByClass(Final2.class);
    job.setJobName("Agrupació per graus i índex de fracàs.");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(RMapper.class);
    job.setReducerClass(SecondReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new Final2(), args);
    System.exit(exitCode);
  }
}
```

El resultat serà de la forma:

```
GADE AP 0.50774527 1162
GADE NT 0.22869956 223
GADE EX 0.16666669 6
GAEI AP 0.16666669 138
GAEI NT 0.33333333 3
GAEP AP 0.37948716 195
...
```

4.4.3 Representació dels resultats.

Finalment l'únic que ens falta fer és representar gràficament l'obtingut a l'apartat anterior per tal de que sigui comprensible. Per mantenir la coherència amb la resta de codi, també es farà en Java.

Tot i que hi ha moltes llibreries que ens permeten fer això s'emprarà JFreeChart[30] pel fet de que és gratuïta i està ben documentada.

4.5 Executant el programa mitjançant Hadoop.

Ara que ja està tot llest, només falta executar el programa. En aquest cas serà diferent de quan ho vam fer a l'exemple del capítol 3 ja que hem de tenir en compte tres aspectes:

1. Per executar Hadoop en mode distribuït necessitam HDFS i YARN.
2. Hadoop agafarà les dades de HDFS, no del sistema de fitxers de la màquina hoste.
3. S'han d'executar dues tasques on la sortida de la primera és l'entrada de la segona.

El primer punt és senzill ja que només s'han d'executar tres *scrpits*:

```
cd hadoop-2.7.3/sbin/
printf "\nStarting HDFS...\n"
./start-dfs.sh
printf "\nStarting YARN...\n"
./start-yarn.sh
printf "\nStarting historyserver...\n"
./mr-jobhistory-daemon.sh start historyserver
printf "\nDone!\n"
```

Una vegada s'ha preparat l'anterior, ja es pot començar a executar l'anàlisi. El primer que hem de fer és eliminar fitxers d'execucions anteriors amb la comanda `rm -r`, que com que s'ha d'executar damunt HDFS ha d'anar precedida de "`hadoop fs -`".

Amb els fitxers eliminats ja es poden transmetre els nous fitxers d'entrada a HDFS amb la comanda `put <Host path> <HDFS path>`:

```
#Eliminam dades d'execucions previes
hadoop fs -rm -r input
hadoop fs -rm -r output
hadoop fs -rm -r final

#Passam els inputs a HDFS
hadoop fs -put input input
```

A continuació s'executen les tasques. La sintaxi per executar-les és idèntica a la de l'exemple del capítol 3, només que per la tasca 1 hem d'especificar els dos fitxers d'entrada als paràmetres:

```
#Job 1
export HADOOP_CLASSPATH=p03-final-part1-4.0.jar
hadoop Final1 input/acces.txt input/qualificacio.txt output

#Job 2
export HADOOP_CLASSPATH=p03-final-part2-4.0.jar
hadoop Final2 output/part-r-00000 final
```

Finalment es transmeten els fitxers de sortida, emmagatzemats a HDFS, al directori de treball. Per fer-ho basta executar la comanda `get`:

```
#Recuperam els resultats
rm -r output
rm -r final
hadoop fs -get output output
hadoop fs -get final final
```

Quan aquest *scrpit* acabi tendrem dues carpetes noves al directori de treball: output i final. Dins "output" hi trobarem la sortida de la primera tasca i dins "final", els resultats de la segona que posteriorment representarem.

Amb el fitxer que trobarem dins la carpeta "final" executam el programa que representa les gràfiques i ens dibuixarà el que veurem al següent apartat.

Per acabar s'han d'aturar els serveis i *daemons* iniciats abans d'executar la tasca, en l'ordre invers de com s'han iniciat:

```
cd hadoop-2.7.3/sbin/
printf "\nStopping historyserver...\n"
./mr-jobhistory-daemon.sh stop historyserver
printf "\nStopping YARN...\n"
./stop-yarn.sh
printf "\nStopping HDFS...\n"
./stop-dfs.sh
printf "\nDone!\n"
```

4.5.1 Seguiment d'una tasca distribuïda.

Si executam Hadoop a una única màquina és senzill fer el seguiment de la tasca ja que mentre s'executa la consola va mostrant un percentatge, però això no és possible si treballam en clústers.

Per seguir una tasca distribuïda Hadoop ens proporciona una direcció que es pot accedir a través del navegador i proporciona una interfície web que permet visualitzar l'estat de la tasca i dur a terme algunes accions.

A la figura 4.5 hi ha un exemple de com seria si tenguéssim dues tasques funcionant concurrentment.

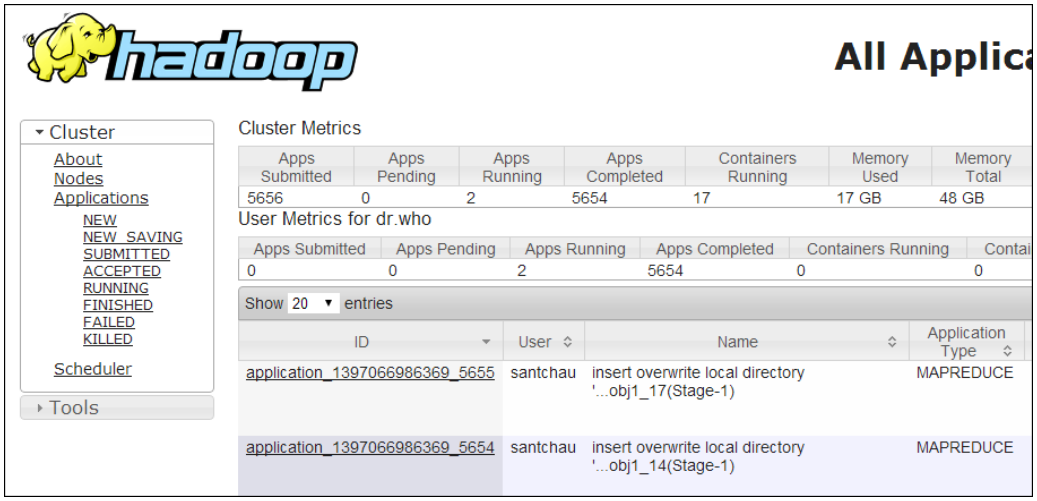

Figura 4.5: Monitor propi de Hadoop (Extreta de [5]).

4.6 Resultats.

Després d'executar els dos treballs MapReduce i representar els resultats finalment assolim l'objectiu del treball (figura 4.6).

Abans de res, cal esmentar que per representar la figura s'han filtrat els graus amb menys de 150 matriculats. De no fer-se això, trobaríem plans d'estudis nous o minoritaris que no ens deixarien apreciar la tendència general.

A la figura podem veure representat el fracàs d'alguns plans d'estudis de la nostra universitat. Es pot apreciar que la majoria dels graus tenen entre un 20% i un 55% de fracàs a la franja de suficient i que es va reduint segons puja la nota d'accés. En general podem dir que índex de fracàs (suficient) > índex de fracàs (notable) > índex de fracàs (excel·lent), indicant que la formació prèvia dels alumnes influeix de forma evident en el seu èxit a la universitat.

Cal destacar el cas de Grau en Enginyeria Informàtica (GIN2), un pla d'estudis que va començar el curs acadèmic 2014-15. El primer que crida l'atenció d'aquests estudis és que només amb 2 anys acadèmics (2014-15 i 2015-16) ja superi el filtre, amb una mitja de 80 matriculats anuals.

També crida l'atenció que l'índex de fracàs d'aquest pla d'estudis és més alt que el de la resta. Si miram les estadístiques d'aquests estudis al fitxer que ens ha generat Hadoop obtenim el següent:

```
GIN2 AP 0.6967213 122
GIN2 NT 0.3333333 36
GIN2 EX 0.5 2
```

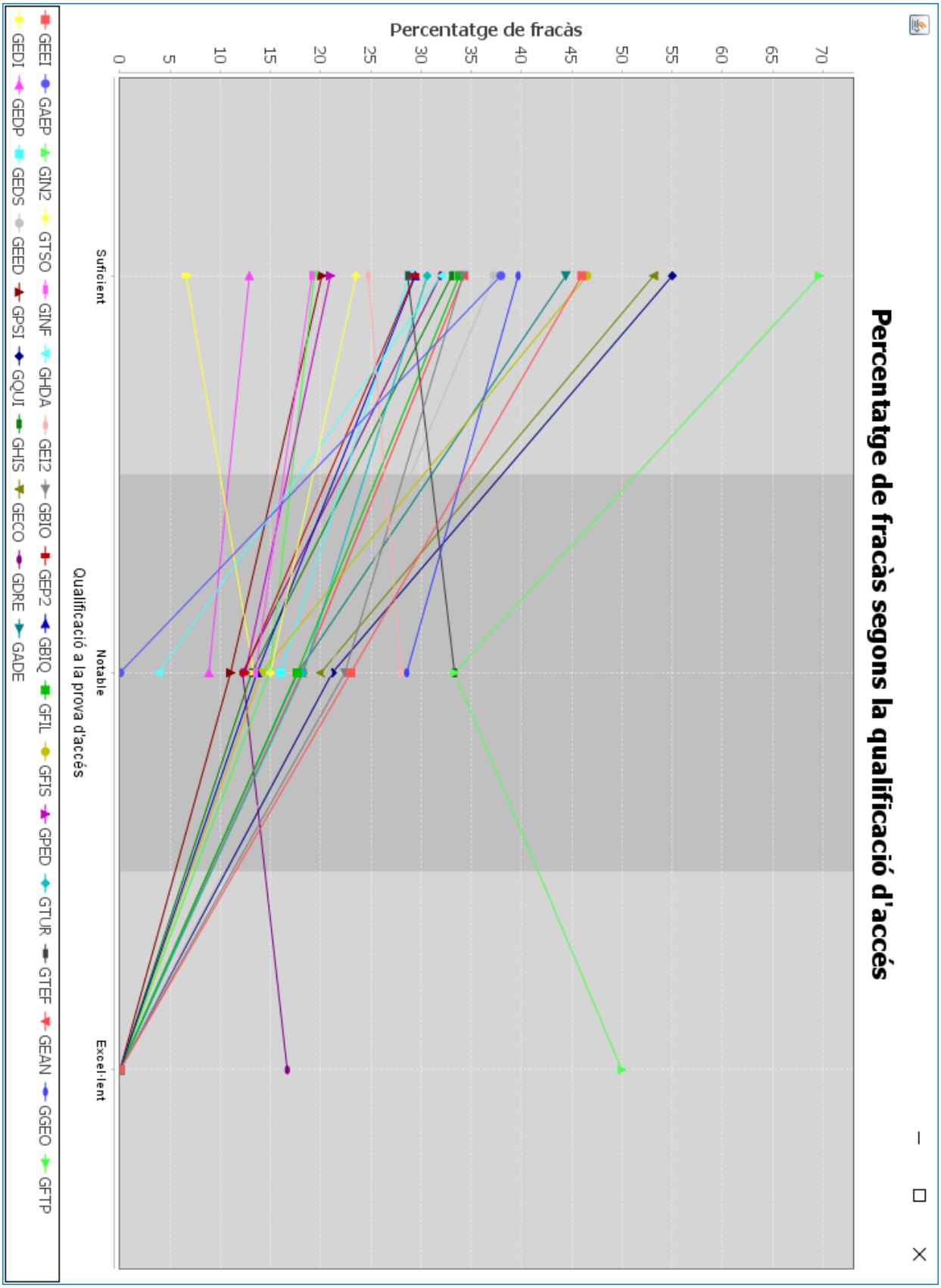

Figura 4.6: Representació dels graus amb més de 150 matriculats (Elaboració pròpia).

Dels estudiants que han entrat amb un suficient, 95 de 122 —un 69.67%— no han passat primer. Tot i que en principi pot semblar impactant tenir un fracàs tan alt cal tenir en compte que al ser un grau nou hi pot haver gent que se'l treu poc a poc, i que al llarg dels anys aquest índex anirà baixant fins estabilitzar-se. No obstant, que 2 de cada 3 estudiants no passin primer és preocupant.

Finalment, podem observar que la majoria de matriculats d'aquests estudis —un 76.25%— tenen un suficient a les PAU, mentre que només dos estudiants —un 1.25%— tenen un excel·lent. Aquest patró és comú a la majoria de graus sense nota de tall o amb una nota de tall baixa, però en aquest cas resulta encara més exagerat.

CAPÍTOL 5

CONCLUSIONS

En aquest treball s'han pogut veure els principis de Hadoop i l'anàlisi de grans quantitats de dades. Se li ha donat un enfocament diferent a l'estudiat durant el grau: mentre que en assignatures d'estadística i mineria de dades hem après com es pot fer un estudi i les tècniques que es poden emprar, aquest treball està més enfocat a fer que l'anàlisi acabi en un temps raonable.

Cal tenir en compte que al disposar de totes les dades de la Universitat s'ha simplificat l'estudi ja que no s'han fet prediccions, només una representació. A més, el fet de que les dades siguin properes al dia a dia dels estudiants ha resultat gratificant, ja que els resultats són més "palpables".

A nivell personal m'ha agradat treballar amb Hadoop. El patró MapReduce que empra resulta molt clar, tant per programar com per revisar programes ja fets. Tot i això, degut a que és molt diferent de qualsevol cosa que haguem vist durant la carrera, al principi pot resultar confós si no es compta amb la bibliografia adequada. El fet de que no estigui integrat a cap *Integrated Development Environment* (IDE) ha fet la costa d'aprenentatge més pronunciada ja que compilar els programes s'havia de fer amb Maven, cosa que no hem fet a cap assignatura.

Respecte els resultats obtinguts, excepte el cas de GIN2 tots han estat tal i com s'esperava: com més alta sigui la nota d'accés, més alta és la probabilitat de que un estudiant passi primer. El que no esperava era que la tendència fos tan clara i que la seguissin la majoria de graus.

BIBLIOGRAFIA

[1] ScaleDB. Big data. [Accedit: 01-05-2017]. [Online]. Available: http://www.scaledb.com/big-data-php.php (document), 2.1

[2] J. Zhu and A. Wang, "Data modelling for big data," 2012. (document), 2.2, 2.5

[3] M. Kaczanowski, "Finding mutual friends with openmpi and map-reduce," 2015, [Accedit: 28-03-2017]. [Online]. Available: http://mkaczanowski.com/finding-mutual-friends-with-openmpi-and-map-reduce/ (document), 3.1

[4] T. White, *Hadoop, The Definitive Guide*, 4th ed. O'Reilly, 2009. (document), 2.2, 3.1.2, 3.4, 3.5

[5] A. Docs. Monitoring mapreduce jobs. [Accedit: 12-06-2017]. (document), 4.5

[6] Seagate. Seagate st-41600n datasheet. [Accedit: 19-01-2017]. [Online]. Available: http://bk0010.narod.ru/DRIVESPECS/SEAGATE/4328.txt 2.2

[7] Apache. Apache cassandra. [Online]. Available: http://cassandra.apache.org 2.4

[8] ——. Apache hive. [Online]. Available: https://hive.apache.org 2.4

[9] ——. Apache hadoop. [Online]. Available: https://hadoop.apache.org 3.1.1

[10] ——. Apache lucene. [Online]. Available: https://lucene.apache.org/core/ 3.1.1

[11] ——. Apache nutch. [Online]. Available: https://nutch.apache.org/ 3.1.1

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," 2003, [Accedit: 28-03-2017]. [Online]. Available: https://research.google.com/archive/gfs.html 3.1.1

[13] S. Ghemawat and J. Dean, "Mapreduce: Simplified data processing on large clusters," 2004, [Accedit: 28-03-2017]. [Online]. Available: https://research.google.com/archive/mapreduce.html 3.1.1, 3.1.3, 3.1.4

[14] E. Baldeschwieler. (2008) Yahoo! launches world's largest hadoop production application. [Accedit: 28-03-2017]. [Online]. Available: http://yahoohadoop.tumblr.com/post/98098649696/yahoo-launches-worlds-largest-hadoop-production 3.1.1

[15] D. Gottfrid, "Self-service, prorated supercomputing fun!" 2007, [Accedit: 28-03-2017]. [Online]. Available: https://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/?_r=1 3.1.1

[16] Amazon. Amazon elastic compute cloud. [Accedit: 28-03-2017]. [Online]. Available: https://aws.amazon.com/es/ec2/ 3.1.1

[17] O. O'Malley, "Terabyte sort on apache hadoop," 2008, [Accedit: 28-03-2017]. [Online]. Available: http://sortbenchmark.org/YahooHadoop.pdf 3.1.1

[18] Google, "Sorting 1pb with mapreduce," 2008, [Accedit: 28-03-2017]. [Online]. Available: https://googleblog.blogspot.com.es/2008/11/sorting-1pb-with-mapreduce.html 3.1.1

[19] O. O'Malley and M. Arun C., "Winning a 60 second dash with a yellow elephant," 2009, [Accedit: 28-03-2017]. [Online]. Available: http://sortbenchmark.org/Yahoo2009.pdf 3.1.1

[20] Databricks. [Online]. Available: https://databricks.com 3.1.1

[21] R. Xin, P. Deyhim, A. Ghodsi, X. Meng, and M. Zaharia, "Sorting 1pb with mapreduce," 2014, [Accedit: 28-03-2017]. [Online]. Available: http://sortbenchmark.org/ApacheSpark2014.pdf 3.1.1

[22] Apache. Apache lucene. [Online]. Available: https://pig.apache.org 3.1.1

[23] M. Nemschoff. (2013) Big data: 5 major advantages of hadoop. [Accedit: 09-04-2017]. [Online]. Available: http://www.itproportal.com/2013/12/20/big-data-5-major-advantages-of-hadoop/ 3.2

[24] B. Proffitt. (2012) Hadoop could save you money over a traditional rdbms. [Accedit: 09-04-2017]. [Online]. Available: http://www.computerworlduk.com/it-management/hadoop-could-save-you-money-over-traditional-rdbms-3329092/ 3.2

[25] B. Barney. Message passing interface (mpi). [Accedit: 01-04-2017]. [Online]. Available: https://computing.llnl.gov/tutorials/mpi/ 3.3.2

[26] S. Carey. (2016) Hadoop vs spark: Which is right for your business? pros and cons, vendors, customers and use cases. [Accedit: 10-04-2017]. [Online]. Available: http://www.computerworlduk.com/data/hadoop-vs-spark-which-is-right-for-your-business-pros-cons-vendors-customers-use-cases-364288 3.3.4

[27] [Accedit: 17-05-2017]. [Online]. Available: http://spark.apache.org/ 3.3.4

[28] S. Carey. (2016) Big data and business intelligence trends 2017: Machine learning, data lakes and hadoop vs spark. [Accedit: 10-04-2017]. [Online]. Available: http://www.computerworlduk.com/data/big-data-bi-trends-2017-machine-learning-data-lakes-hadoop-vs-spark-3652166/ 3.3.4

[29] GNU. Gnu bash. [Online]. Available: https://www.gnu.org/software/bash/ 3.6

[30] A. Viklund. [Online]. Available: http://www.jfree.org/jfreechart/ 4.4.3