UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques
Departament de Matemàtica Aplicada i Anàlisi

TREBALL FINAL DE GRAU

K-means web clustering amb Hadoop MapReduce

Autor:
Alberto HUÉLAMO SEGURA

Director:
Eloi PUERTAS I PRATS

Primavera 2013

Aquesta obra està subjecta a una llicència de
Reconeixement-NoComercial-CompartirIgual 3.0 No adaptada de Creative
Commons
http://creativecommons.org/licenses/by-nc-sa/3.0/

A la meva familia i a l'Estefania.

Machines take me by surprise with great frequency.
Alan Turing

Agraïments

Vull donar les gràcies al meu director de projecte, l'Eloi Puertas i Prats per tot el seu suport, ajut i seguiment durant tot el procés de desenvolupament d'aquest projecte.

Donar les gràcies també al Francesc Dantí, que m'ha donat molt suport per instal·lar la infraestructura d'experimentació.

Agrair al Lluís Garrido que m'hagi proporcionat documentació i exemples de LaTeXque han sigut de gran ajut per l'elaboració d'aquesta memòria.

I per acabar, gràcies al Noël Torres Caballé per proporcionar-me documentació sobre Hadoop i MapReduce.

Abstract

This paper proposes a solution to the problem of clustering large amount of web documents. The Hadoop framework, implementation of MapReduce distributed programming paradigm, developed by Google, plays a very important role in this field due to its scalability and ease to parallelize software. This is the reason why it is used in this project.

Meanwhile, K-means clustering algorithm is easily adaptable to MapReduce programming model and provides proper results for web documents. The documents will be represented as a frequence vectors of terms and keywords and this is what algorithm needs to work.

The developed software uses Hadoop in order to perform both tasks which make up the overall process: document processing and the clustering. Web documents are in HTML, which is not suitable for K-means. It is necessary preprocess them to extract descriptors and to pass them to the clustering algorithm. This is the first part of the process.

The second part, K-means on Hadoop, goes beyond typical Hadoop execution, using most of the tools which Hadoop provides to make document clusters, from descriptors obtained from first part of the process.

Resum

Aquest treball proposa una solució a la problemàtica del clústering web per grans volums de documents. El framework Hadoop, implementació del paradigma de programació distribuïda MapReduce, desenvolupat per Google, juga un paper molt important en aquesta disciplina per la seva escalabilitat i la facilitat per paral·lelitzar un software. És per això que és el que s'utilitza en aquest projecte.

Per la seva banda, l'algorisme de clústering K-means és fàcilment adaptable al model de programació MapReduce i proporciona uns resultats adients pels documents web. Els documents quedaran representats com a vectors de freqüències de termes i paraules clau o *keywords* i és el que li cal a l'algorisme per funcionar.

El software desenvolupat fa servir Hadoop per tal de dur a terme les dues tasques que globalment conformen el procés: el processament dels documents i el clústering. Els documents web vénen en format HTML, cosa que no és adequada pel K-means. Cal preprocessar-los per extreure els descriptors i poder passar-los per l'algorisme de clústering. Aquesta és la primera part del procés.

La segona part, l'algorisme K-means sobre Hadoop, va un pas més enllà de l'execució típica d'un programa Hadoop, fent servir la majoria de les eines que dóna el framework per realitzar les agrupacions dels documents, a partir dels descriptors obtinguts de la primera part del procés.

Índex

Agraïments **v**

Abstract **vii**

Resum **ix**

Índex de figures **xiii**

Índex de taules **xv**

Índex d'algorismes **xvii**

1 Introducció **1**

1.1 Motivació 1

1.2 Descripció del problema 3

1.3 Contribució 4

1.4 Resum dels capítols 5

2 Antecedents i anàlisi del problema **7**

2.1 Antecedents del Machine Learning 7

2.1.1 K-means 9

2.2 Tècniques Big Data 9

2.2.1 MapReduce 9

2.2.2 Hadoop 12

2.3 Conclusió de l'anàlisi 15

3 Disseny **17**

3.1 Les dades: representació de característiques 18

3.2 Indexador 20

3.2.1 Filtratge de l'índex 21

3.3 Agrupador (clústering) 21

4 Implementació **25**
4.1 Indexador 26
4.2 Agrupador (clústering) 28
4.2.1 Documents “stub” 31

5 Experimentació i resultats **33**
5.1 Experiments 33
5.1.1 Infraestructura 33
5.1.2 Dataset 34
5.2 Proves qualitatives 34
5.3 Proves de rendiment 37
5.4 Gestió del projecte i costos 39

6 Conclusions i treball futur **41**
Apunt final personal 42

Bibliografia **43**

A Instal·lació de Hadoop **45**

B Manual d’ús **49**

Índex de figures

1.1 Les 3 Vs del Big Data 2
1.2 Procés de tractament de les dades 3

2.1 Arquitectura MapReduce 12
2.2 HDFS 14
2.3 Hadoop MapReduce 15

3.1 Fases globals del procés 18
3.2 Representació vectorial dels documents 19
3.3 Representació vectorial dels documents amb *keywords* 20
3.4 Iteració de K-means amb MapReduce 22
3.5 Programa controlador 23

4.1 Implementació de l'indexador 27
4.2 Taules de l'índex 28
4.3 Assignació de documents per fases 29

5.1 Gràfica de rendiment de l'índex 37
5.2 Gràfica de rendiment 38
5.3 Diagrama de Gantt 39

B.1 Directoris al Makefile per canviar 50
B.2 Monitors d'activitat 52

Índex de taules

5.1 Taula de mesures qualitatives 35
5.2 Mètriques des dels clústers 36
5.3 Mesures qualitatives de configuració 36
5.4 Horari setmanal 39

B.1 Comandes del Makefile de control. 51
B.2 Comandes del Makefile de control. 52

Índex d'algorismes

1 Algorisme K-means [9] 8
2 Funció Map 10
3 Funció Reduce 11

Capítol 1

Introducció

1.1 Motivació

Big Data. En els darrers anys aquesta expressió s'ha anat estenent en l'àmbit de les ciències de computació. Es defineix Big Data com un conjunt de tècniques per emmagatzemar, capturar i processar, en un temps raonable, una gran quantitat de dades[1].

L'evolució dels dispositius mòbils, com smartphones o tables, ha propiciat la generació de quantitats de dades immenses en l'àmbit casolà i domèstic. Al 2011, es generaven 200 milions de tweets al dia [8]. Fent un càlcul ràpid, això suposa al dia la generació de 28GB de dades[2].

Un dispositiu mòbil pot recollir la seva posició GPS i enviar-la a un servidor per fer un registre i analitzar a posteriori patrons de moviment o trajectòries. Si això es fa per cada dispositiu, el volum de dades passa a ser ingent.

Però Big Data no només és present al món de la mobilitat. El Gran Col·lisionador d'Hadrons (LHC), a Ginebra, disposa 150 milions de sensors que proporcionen dades 40 milions de vegades per segon. A més, cada segon es produeixen 600 milions de colisions de les quals, després d'haver filtrat el 99.999%, només 100 són d'interès. Anualment, i encara treballant amb menys del 0.001% de les dades generades, això es tradueix en 25 petabytes[3] abans de la replicació i 200 petabytes després de la replicació.

Es troben encara més exemples d'ús de Big Data [11]:

- Originàriament, decodificar el genoma humà comportava 10 anys de processament. Actualment s'ha aconseguit en 1 setmana.

[1] Data sets amb una mida de l'ordre de centenars o milions de gigabytes.
[2] En realitat en són més, perquè s'ha considerat que cada caràcter ocupa 1 byte.
[3] $1PB = 1.000TB = 1.000.000GB$

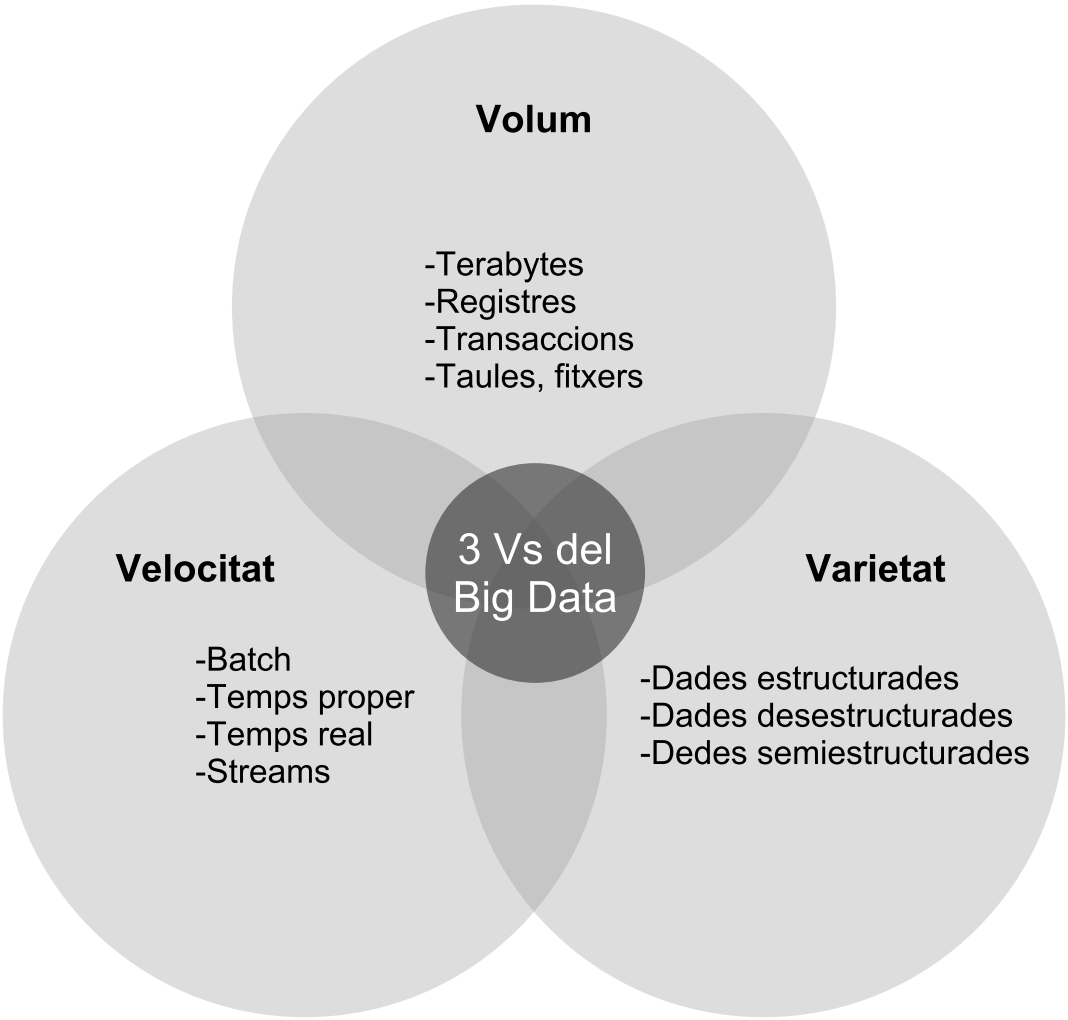

Figura 1.1: Les 3 Vs del Big Data. No tan sols es tracta de grans quantitats de dades, sino també de la varietat de les mateixes i la rapidesa amb la que aquestes dades han de ser tractades.

- El *NASA Center for Climate Simulation* (NCCS) emmagatzema 32 petabytes d'observacions climàtiques i simulacions.

- L'Administració Obama va posar en marxa la *Big Data Research and Development Initiative*, que investiga com el Big Data pot ser usat per abordar els problemes importants que afronta el govern.

- Amazon gestiona milions d'operacions al dia. El nucli d'Amazon el suporten les tres bases de dades sobre Linux més grans del món, amb capacitats de $7.8TB$, $18.5TB$ i $24.7TB$.

- Facebook gestiona 50 mil milions de fotos dels seus usuaris.

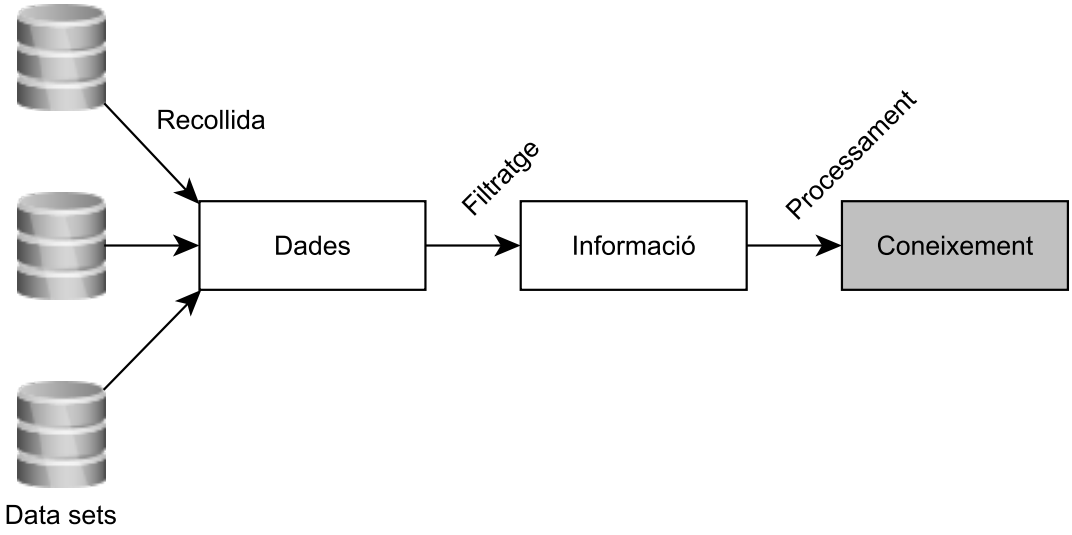

Figura 1.2: Procés de tractament de les dades. De la recopilació de dades crues es vol estreure un coneixement.

Totes aquestes dades a priori, en el moment de recollir-les, són crues. Necessiten refinament per convertir-les en informació que generi un coneixement (figura 1.2). Necessiten ser processades. Com s'ha dit, el món del Big Data també aborda les tècniques de processament d'aquestes dades i s'ha generat una necessitat de noves tècniques, nous paradigmes i nous frameworks per tal d'abordar aquests problemes d'una forma eficient i relativament senzilla i no només focalitzant la implementació de l'algorisme que resol el problema. Frameworks que aposten per la paral·lelització de les dades contra la optimització computacional, ja que arriba el moment en què és millor tenir més unitats computacionals per grans volums de dades. MapReduce i Apache Hadoop avui en dia aporten el suficient nivell d'abstracció i la potència necessària per dur a terme tasques de programació paral·lela i distribuïda, aconseguint adaptacions d'algorismes al paradigma MapReduce.

1.2 Descripció del problema

A la secció 1.1 s'ha fet esment a diversos problemes i situacions que comporten grans volums de dades i que són relativament recents. Però no s'ha tingut en compte l'escenari més evident però que més desapercebut passa, possiblement per la seva trivialitat. I és la manera més senzilla amb la que un usuari pot accedir a la informació del món global que és el d'Internet: la *World Wide Web*.

Comptabilitzar la quantitat de dades web és una feina impossible i inexacta a

raó de que les pàgines han deixat de ser estàtiques. PHP, entre d'altres, i JavaScript generen els documents web de forma dinàmica amb un contingut emmagatzemat en una base de dades. Per tant, el nombre de pàgines o la quantitat d'espai físic que ocupa el conjunt de documents web és arbitrari. Degut a aquest fet, la cerca de continguts web ha hagut d'adaptar-se a la dinamicitat de la xarxa. Els *bots* dels cercadors fan un recorregut de la xarxa per mitjà dels enllaços entre documents web, i d'aquesta manera indexar les pàgines a posteriori. Si les pàgines són dinàmiques s'ha de fer un recorregut pels enllaços d'accés que generen un contingut o un altre.

D'altra banda, la cerca pròpiament dita ofereix els resultats de la *query* d'usuari contrastats amb un índex. La cerca es podria facilitar si els documents web, a part d'indexats, estiguéssin categoritzats o **agrupats** a priori, de tal manera que es reduiria l'índex de cerca, éssent el nombre de documents menor en una categoria que tot un índex. Es pot enfocar des del punt de vista contrari: fer agrupacions en els resultats d'una cerca per tal de facilitar un accés previ als continguts de la mateixa. Fins i tot, fora del marc de la cerca de pàgines web, és d'interès agrupar-les a partir del contingut i construir directoris de webs. El repte és oferir aquestes agrupacions per un nombre inmens de documents web.

A part de la cerca de documents pròpiament dita, l'oferiment de resultats relacionats suposa un reforç molt positiu i útil en el marc de la recerca d'informació i el fet de tenir un conjunt de pàgines englobades en una mateixa categoria pot proporcionar aquests resultats relacionats de forma més eficient i acurada, polint la qualitat de la cerca. Tot i que cercadors genèrics com Google no precisen aparentment d'aquesta característica pel fet de ser cerques més genèriques, pot interessar realitzar l'agrupació de documents en un sol website per tal de categoritzar-los.

Existeixen molts algorismes d'agrupació o clustering. K-means n'és un d'ells, el qual proporciona agrupacions de forma no supervisada. Per la seva banda, Hadoop MapReduce permet una implementació de l'algorisme K-means per agrupacions de gran quantitat de pàgines web. La unió d'aquests conceptes permet desenvolupar una solució per oferir cerques més acurades o directoris web, entre d'altres.

1.3 Contribució

Els objectius d'aquest projecte posen en comú el clústering web i el Big Data. Es proposa una implementació sobre el framework Apache Hadoop de l'algorisme K-means per l'agrupació de documents web, havent-los indexat a priori.

Més específicament, els objectius són:

- Aprenentatge del framework Apache Hadoop i del paradigma MapReduce per

a la programació paral·lela i distribuïda per grans volums de dades.

- Implementació d'un software d'indexació de documents web amb Apache Hadoop.
- Implementació de l'algorisme K-means per clústering web amb Apache Hadoop.

1.4 Resum dels capítols

1.4.1 Anàlisi del problema

En aquest capítol s'introdueixen les diferents bases dels conceptes que es fan servir en aquest projecte així com d'alternatives a la tria que s'ha realitzat per portar a terme la solució del problema. Es farà un repàs de l'algorisme K-means. Tot seguit, es contemplarà el paradigma MapReduce i la seva implementació més coneguda: Hadoop.

1.4.2 Disseny

Una de les parts més importants en el desenvolupament de software és la d'elaborar un model abstracte de l'aplicació que es vol implementar a posteriori: el disseny. Aquest model representa el que es vol implementar sense entrar en detall de com s'implementa a nivell de plataforma. En aquest capítol s'exposa quin és el disseny de la solució software del projecte.

1.4.3 Implementació

La posta en escena d'un disseny és la seva implementació. D'implementacions n'hi pot haver tantes o més com plataformes en les quals dur-les a terme. La implementació del disseny ha de ser tant fidel i optimitzada com sigui possible, de tal manera que les limitacions de la plataforma siguin mínimes. En aquest capítol es revisarà quina ha estat la implementació del disseny proposat per aquest projecte, aptopant al codi l'explicació sense arrivant a verure-ho per tal d'aclarir allò que no hagi quedar del tot clar en l'explicació del disseny.

1.4.4 Experimentació i resultats

L'obtenció de resultats experimentals pot donar una aproximació de la qualitat del software desenvolupat a més de donar peu a unes conclusions que podrien no

esperar-se. En aquest capítol s'explica quins experiments s'han realitzat amb el software desenvolupat a més d'altres aspectes d'interès relatius al projecte.

1.4.5 Conclusions i treball futur

En aquest darrer capítol s'examinaran les fites plantejades a l'inici i s'oferirà una conclusió englobant del projecte sencer. A banda es realitzaran propostes de treball futur destinades tant a millorar el software com a construir-ne de nou a partir d'aquest.

Capítol 2

Antecedents i anàlisi del problema

En aquest capítol s'introdueixen les diferents bases dels conceptes que es fan servir en aquest projecte així com d'alternatives a la tria que s'ha realitzat per portar a terme la solució del problema. Es farà un repàs de l'algorisme K-means. Tot seguit, es contemplarà el paradigma MapReduce i la seva implementació més coneguda: Hadoop.

La cerca en documents web ha suposat una eina indispensable des de l'aparició d'Internet, però si es remunta en el temps, la cerca eficient d'un patró en una cadena de caràcters és un dels problemes més populars, més bàsics i més abordats en les ciències de la computació. El sol fet que aquestes són les bases de la cerca es mereix un esment en aquesta memòria.

L'accés a una determinada informació ha sigut vital al llarg de l'existència de la xarxa. Mirant enrere, en l'època dels primers cercadors, les querys s'atenien de forma trivial, llençant-les contra un índex que contrastava la cadena de la query amb el contingut d'un document i oferia els resultats de forma seqüencial. No va ser fins l'aparició del PageRank de Google que aquests resultats s'oferiren amb un ordre de rellevància no basat en el contingut. Per la seva banda, apareixren directoris de webs que ofereixen una classificació, duta a terme de dues formes: supervisada o no supervisada. El fet que la primera exigeixi un entrenament previ fa que la construcció categòrica sigui farragosa i poc pràctica, donada la dinàmica de canvi constant que té Internet.

Eines com les *tag clouds* [10], que sorgiren per la categorització de pàgines, poden ser contruïdes a partir de termes comuns en subconjunts d'un conjunt de documents.

2.1 Antecedents del Machine Learning

Diverses propostes sobre aquesta àrea parlen de realitzar agrupacions basades en el contingut de documents sobre el resultat d'una cerca. Alguns, com a [12], sobre

els snippets retornats pels motors de cerca i d'altres, com [6], sobre el conjunt de documents sencers que retorna una cerca.

Comparant algorismes de clústering, a [12] se n'esmenten diversos. Els algorismes AHC (*Agglomerative Hierarchical Clustering*) són els més citats en la literatura, però no són bons candidats quan la col·lecció de documents és prou gran, arribant a complexitats de $O(n^2)$ o fins i tot $O(n^3)$ segons el mètode. Es demostra que aquests algorismes són lents per tal de fer agrupacions d'un miler de documents [12]. D'aquesta manera, es passen a considerar els mètodes de clústering linial per la seva major velocitat. Fa especial èmfasi en K-means, amb complexitat $O(nKT)$ (on n és el nombre de documents, K és el nombre de clústers desitjats i T és el nombre d'iteracions). El problema d'aquest mètode és que els clústers han de tenir una forma esfèrica, cas que no es dóna sempre si es tracten documents.

Deixant de banda el contingut dels documents, existeixen altres tècniques per fer agrupacions. A [4] es fa la proposta de realitzar una agrupació basada en l'estructura dels documents: en el DOM. Les característiques dels documents vindrien donats per l'estructura del DOM i les diverses propietats que tenen, com la posició d'elements, l'estil, etc.

Algorithm 1: Algorisme K-means [9]

Input: $E = \{e_1, e_2, \ldots, e_n\}$ (items per agrupar)
k nombre de clusters
Output: $C = \{c_1, c_2, \ldots, c_n\}$ (conjunt de centroids)
$L = \{l(e)|e = 1, 2, \ldots, n\}$ (conjunt d'etiquetes de clúster de E

```
begin
    foreach c_i ∈ C do
        c_i ← e_j ∈ E (p.e. sel·lecció arbitrària)
    end
    foreach e_i ∈ E do
        l(e_i) ← argminDistancia(e_i, c_j)j ∈ {1, ..., k}
    end
    canviat ← false; iter ← 0; repeat
        foreach c_i ∈ C do
            ActualitzaCluster(c_i)
        end
        foreach e_i ∈ E do
            minDist ← argminDistancia(e_i, c_j)j ∈ {1, ..., k}
            if minDist ≠ l(e_i) then
                l(e_i) ← minDist canviat ← true
            end
        end
    until canviat = true;
end
```

2.1.1 K-means

El primer pilar del projecte és l'algorisme K-means (Algorisme 1), un dels més populars algorismes de classificació no supervisada. Es parteix d'un nombre K de clústers definit a priori i la definició de les característiques dels ítems per agrupar. Per començar, l'algorisme defineix un nombre K de centroides que han de tenir les mateixes característiques d'un ítem. Es calcula la distància de cada ítem amb cada centroide i s'assigna l'ítem al centroide més proper. A partir de la mitjana (*mean*) dels items assignats a cada centroide, es recalculen els centroides. Aquest procés és iteratiu i s'atura quan els centroids recalculats no es mouen respecte la iteració anterior. La part de càlcul de distàncies és la més costosa a nivell computacional i més si les característiques dels ítems són moltes. Si es consideren n objectes que es volen assignar a k clusters a cada iteració l'algorisme du a terme nk càlculs de distància.

2.2 Tècniques Big Data

Pel que fa a la implementació paral·lela i distribuïda de l'algorisme que es faci servir, existeixen diverses eines viables per a la implementació. Llibreries com OpenMP, OpenMPI i OpenCL ofereixen un marc de possibilitats pel desenvolupament d'aplicacions paral·lelitzables[1]. Tot i aquestes eines, Google desenvolupà un paradigma de programació paral·lela adaptable a gran quantitat de problemes: MapReduce. A més, existeixen diverses implementacions d'aquest paradigma, però la més popular és Hadoop, de la Apache Software Foundation. El que comporta la utilització de Hadoop és la transparència al programador pel que fa a transferències per xarxa i gestió distribuïda de fitxers, la qual cosa comporta una descàrrega important de treball.

2.2.1 MapReduce

MapReduce és un model de programació distribuïda desenvolupat per Google amb l'objectiu de permetre el processament paral·lel i distribuït de grans quantitats de dades. El paradigma rep el seu nom de les dues funcions en les que es basa (map i reduce) i la seva implementació més adoptada i més desenvolupada és Hadoop, la qual es veurà amb més detall en futures seccions. Val a dir que encara que la solució de molts problemes pot tenir una implementació amb MapReduce, no tots els algorismes es poden paral·lelitzar, o adaptar-los al paradigma. A més, el fet

[1] OpenMPI és l'únic que permet programació distribuïda.

de treballar generalment amb grans quantitats d'informació fa que es facin servir sistemes de fitxers distribuïts (DFS – Distributed File System).

MapReduce basa les seves entrades i sortides i el seu model de dades en parells clau-valor $(K - V)$ els quals són processats per les funcions map i reduce, escrites per l'usuari. L'entrada i la sortida de les dues funcions és un conjunt de parells $(K - V)$. La funció map és executada de forma paral·lela en els diferents nodes del clúster. Rep els parells $(K - V)$ del sistema de fitxers, els processa, i en genera un altre conjunt intermedi. Un cop processades les entrades i generades les sortides, aquestes s'agrupen les que tinguin la mateixa clau. D'altra banda, la funció reduce, que pot ser executada també en paral·lel però generalment en menys nodes, recull totes aquestes sortides agrupades de map de forma que aquí, als parells $(K - V)$, Ki correspon a una de les claus i V_i, a una llista agrupada de valors amb la mateixa K_i. Aquest conjunt de valors se sol reduir a un altre valor més petit o a un conjunt de valors més petit. Típicament, la sortida de reduce és la sortida del programa MapReduce.

$$\begin{aligned} map\,(k_1, v_1) &\rightarrow list\,(k_2, v_2) \\ reduce\,(k_2, list\,(v_2)) &\rightarrow list\,(v_2) \end{aligned} \tag{2.1}$$

Per il·lustrar millor el principi de funcionament d'aquest paradigma convé considerar el següent exemple: Suposar un fitxer de text i considerar un programa MapReduce que compta els mots d'aquest fitxer de text. Suposar que l'entrada del programa són les línies del fitxer, de tal forma que la funció map (Algorisme 2) es cridarà tants cops com línies tingui el fitxer i reduce (Algorisme 3) es cridarà un cop en aquest cas.

Algorithm 2: Funció Map

Input: $E = (K, L)$ on K és arbitrari i L una línia de text.
Output: $M = (W, 1)$
begin
  for $w \in L$ **do**
    $outIntermediate(w, int(1))$
  end
end

Notar que els tipus de sortida de map i els d'entrada de reduce són els mateixos. En l'exemple, la funció map emet una ocurrència de cada paraula que troba i reduce les suma. Per veure-ho més clar:

Notació. Denotar $\langle K, V \rangle$ com un parell clau-valor amb K com clau i V com valor.

Algorithm 3: Funció Reduce

Input: $M = (W, N)$ on W són paraules i N un conjunt de 1.
Output: $R = (W, K)$
begin
  $result \longleftarrow 0$ **for** $k \in N$ **do**
    $result \longleftarrow result + k$
  end
  $out(W, result)$
end

Denotar tl com una línia de text i t_i com un mot del text.

1. L'entrada de map serà $\langle "", tl \rangle$.
2. La sortida de map serà $\langle t_1, 1 \rangle, \langle t_2, 1 \rangle, ..., \langle t_n, 1 \rangle$
3. L'entrada de reduce serà $\langle t_1, [1, 1, 1, 1, 1, 1, 1, ..., 1] \rangle$ ...
4. La sortida de map serà $\langle t_1, sum([1, 1, 1, 1, 1, 1, 1, ..., 1]) \rangle$ ...

Arquitectura MapReduce

La figura 2.1 mostra l'arquitectura d'elements bàsics d'una aplicació MapReduce. Com s'observa, les màquines del clúster, o nodes, es divideixen en un node màster i diversos nodes workers. El màster és l'encarregat de repartir les tasques map i reduce als altres nodes workers i aquests, s'encarreguen de portar-les a terme.

Entrant en detall, l'algorisme MapReduce [5] segueix el següent esquema (els números es corresponen amb els de la figura 2.1):

1. Es parteixen els fitxers d'entrada en M peces, que pesen típicament entre 16 i 64MB, mentre es distribueix una còpia del programa a cada node.
2. Una de les còpies del programa és el procés màster. La resta de workers esperen a rebre tasques del master. Aquest envia tasques Map o Reduce als workers que en aquell moment no tinguin cap tasca assignada (idle).
3. Un worker al que se li assigni tasca Map llegeix l'entrada de la partició d'entrada que li correspongui. Transforma cada partició en un parell clau-valor que serà l'entrada de la funció map. La funció genera un conjunt intermig de parells clau-valor que quedan emmagatzemats en un buffer.
4. De forma periòdica, els parells al buffer intermedi es particionen en R conjunts, la ubicació dels quals és retornada al master. Aquest és responsable de retransmetre aquestes ubicacions als workers encarregats de tasques reduce.

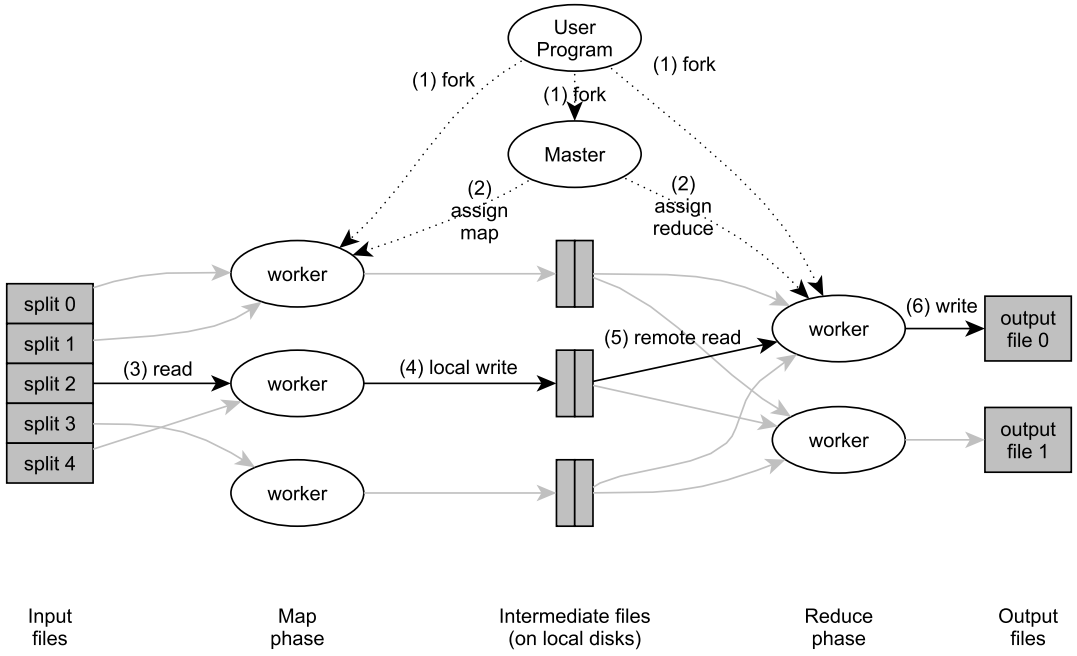

Figura 2.1: Representació d'una arquitectura MapReduce distribuïda [5] amb un node Master i cinc Workers. Com s'observa els map workers processen els splits d'entrada i generen les dades intermèdies que processen els reduce workers per generar la sortida final.

5. En el moment en què un reduce worker és notificat pel màster de les ubicacions intermèdies, llegeix les dades per mitjà de crides remotes del disc dur dels map workers. Quan es llegeixen les dades, es combinen els parells de forma que queden agrupats els valors amb la mateixa clau.

6. Ja a la funció reduce, s'itera sobre els valors d'una mateixa clau i s'escriu part de la sortida.

7. Quan s'acaben les tasques de map i reduce, el master *desperta* el procés d'usuari i es retorna de la crida MapReduce.

Després de l'execució, la sortida queda emmagatzemada en R particions en un o diversos fitxers a disc. Aquesta sortida, a més, pot servir com a entrada per una altra crida MapReduce.

2.2.2 Hadoop

Apache Hadoop és un altre dels pilars principals d'aquest projecte, juntament amb MapReduce i el K-means. Inicialment desenvolupat per Yahoo i actualment per Apache Software Foundation, Hadoop és un framework open source que ens permet

executar aplicacions distribuïdes MapReduce de forma abstracta, és a dir, el programador no s'ha de preocupar de l'intercanvi de missatges entre nodes del clúster, pas de dades, etc. És, doncs, un framework d'alt nivell, si es compara, per exemple, amb OpenMPI.

Les aplicacions es poden executar sobre una màquina en local o sobre un clúster de màquines, tot depenent de la configuració que s'esculli. Apache [3, 1] especifica tres configuracions, de les quals només una és realment útil:

- **Configuració local**. Aquesta configuració és només d'un sol node i és la que ve per defecte al paquet descarregable i consta d'un sol node al qual s'executa un sol procés. S'utilitza per comprovar que la instal·lació de Hadoop es fa de forma correcta.

- **Configuració pseudodistribuïda**. Aquesta configuració és també d'un sol node. La diferència és que s'executa més d'un procés al node. És la configuració que s'ha fet servir per desenvolupar les aplicacions.

- **Configuració de clúster**. És la configuració de treball de Hadoop. Múltiples processos són executats en múltiples màquines d'un clúster.

Hadoop es distribueix juntament amb el *Hadoop Distributed File System* o HDFS. Com el nom indica, és el sistema de fitxers distribuït que s'utilitza per emmagatzemar les dades de forma lògica, és a dir, el conjunt d'emmagatzematges físics dels diferents nodes de treball conforma una única unitat lògica en la qual s'emmagatzemen les dades de les tasques MapReduce. La forma d'emmagatzemar les dades i la ubicació física dels fitxers és transparent al programador, la qual cosa permet tenir un disc dur virtualment infinit si es considera un cluster de màquines infinites. Per gestionar grans volums de dades això és ideal.

Cadascun dels clusters HDFS (Figura 2.2) es conformen d'un servidor anomenat *Namenode* que gestiona l'espai de noms del sistema de fitxers, determina el mapeig dels blocs als Datanodes i regula l'accés. Els altres nodes són els *Datanodes* i s'encarreguen de el propi emmagatzematge de dades i la gestió dels blocs.

HDFS fa servir una política de rèplica de les dades per tal de mantenir un backup. Les rèpliques es distribueixen a través de la xarxa i són normalment tres. Això ho hereta del Google File System, el qual manté una política de *una escriptura – múltiples lectures*, que és un tipus d'accés per dades que no requereixen ser continuament actualizades.

La detecció i gestió de caigudes del sistema també està implementada a Hadoop, així com en HDFS. Quan un node que executa una tasca falla, Hadoop ho detecta

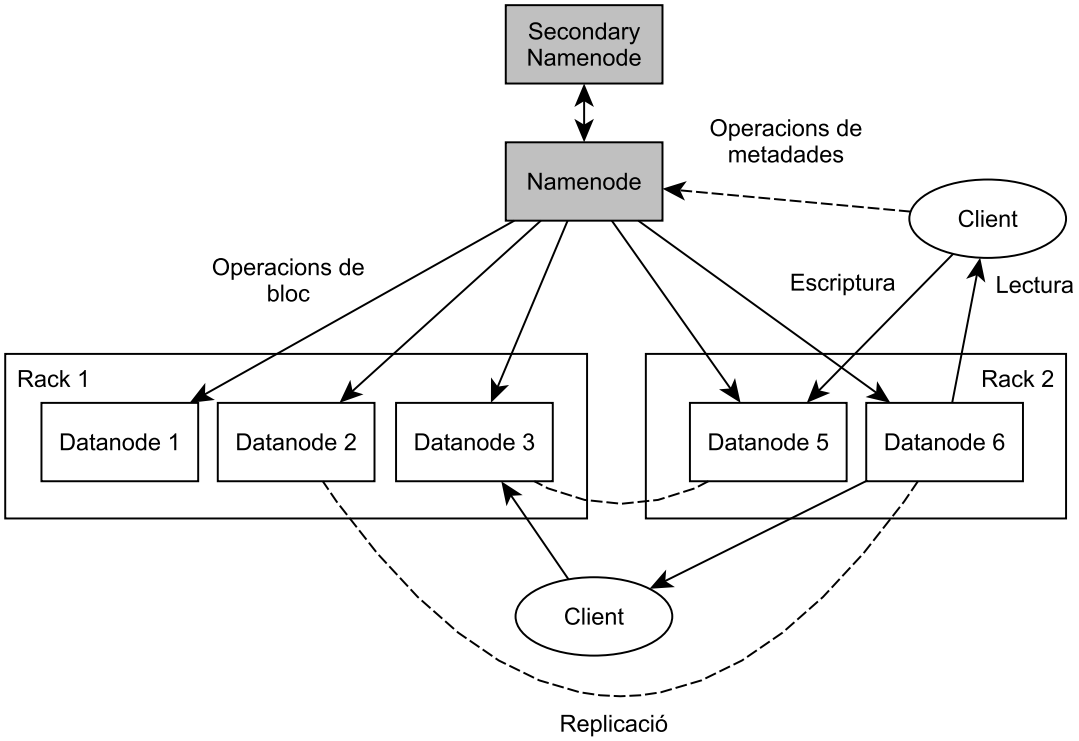

Figura 2.2: Arquitectura del HDFS amb un node primari (Namenode) i cinc nodes d'emmagatzematge (Datanode). El namenode gestiona les rutes de les particions dels fitxers i els processos d'usuari interactuen amb les dades.

i re-executa la tasca en un altre node actiu. A més, crea una còpia dels blocs de dades perduts a partir de les rèpliques que ja té. Existeix però, un risc de fallada del sistema, i és en el cas que el Namenode caigui. Tot i que es pot configurar un Namenode secundari, això no garantitza robustesa al 100% en aquest punt, ja que el secundari s'encarrega de mantenir snapshots de l'estat del primari per facilitar una recuperació ràpida i reiniciar el Namenode primari.

Per a la correcta execució de les tasques map i reduce, Hadoop fa servir un motor d'execució MapReduce que assegura tant l'execució de les tasques com la tolerància a caigudes. Cada conjunt d'execució MapReduce es denota com a *Job*. El motor manté un *Job Tracker*, encarregat del control d'execució dels Jobs. Les aplicacions client envien les peticions de traball al Job Tracker i aquest gestiona i controla l'execució i distribueix les tasques entre els *Task Trackers*, encarregats de l'execució de cada tasca map o reduce. Si una tasca map falla o és massa lenta, s'assigna a un altre node. Si falla una tasca reduce les dades intermèdies dels mappers, executats a priori, poden ser utilitzades per un altre reducer. Aquesta configuració modular i amb independència de tasques afavoreix la robustesa del sistema.

Amb aquest sistema s'aconsegueix la implementació MapReduce de Hadoop.

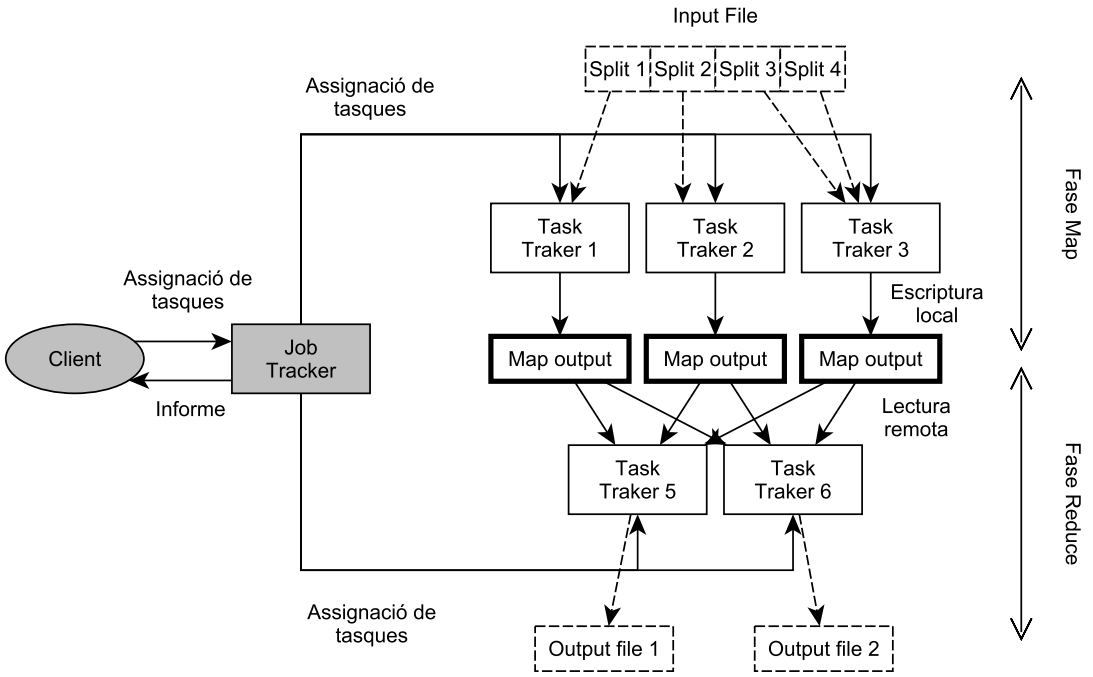

Figura 2.3: Job Hadoop MapReduce. El procés d'usuari interactúa amb el job tracker, el qual realitza l'assignació de tasques map i reduce els nodes task tracker. Els task tracker executen les tasques map i reduce i interactuen amb el HDFS per a l'entrada i sortida de dades.

Notar que tant la figura 2.1 i la figura 2.3 són equivalents, però la segona està en el context i amb la nomenclatura de Hadoop.

El framework disposa d'interficies per definir tipus propis que permeten la serialització i el pas de les dades. A més a més, es dóna també la possibilitat de definir els formats d'entrada i sortida propis, la qual cosa implica més flexibilitat.

2.3 Conclusió de l'anàlisi

Concluint aquesta secció, s'exposa que finalment s'ha triat fer el clústering amb K-means sobre el framework Hadoop, tot i que l'algorisme, com s'ha vist, no és el més ideal, però sí que és ràpid. La unió d'aquests dos conceptes ha d'oferir com a resultat final una aplicació que s'ajusti al problema del Big Data.

Capítol 3

Disseny

Una de les parts més importants en el desenvolupament de software és la d'elaborar un model abstracte de l'aplicació que es vol implementar a posteriori: el disseny. Aquest model representa el que es vol implementar sense entrar en detall de com s'implementa a nivell de plataforma. En aquest capítol s'exposa quin és el disseny de la solució software del projecte.

Recopilar pàgines web i fer un data set a partir de les mateixes és una tasca relativament fàcil. Tot i així, la premisa inicial és que es disposa d'un data set de pàgines web[1] sobre les que poder realitzar el clústering. Tenint aquest fet en compte, caldrà que el software que s'implementi preprocessi aquestes dades (pàgines web) per obtenir uns vectors de característiques els quals farà servir l'algorisme K-means. La conclusió és que tot aquest procés es divideix en dues fases: la fase d'indexació i la fase de clústering.

Cada fase es tradueix en una aplicació executada sobre Hadoop de forma seqüencial (primer l'indexador, després l'agrupador), en la qual es genera una entrada i una sortida (figura 3.1).

Per fer el disseny, s'ha de tenir en compte el model de programació que es vol utilitzar (MapReduce) i quina serà l'algorísmia del programa (K-means). Per tant, s'ha de idear una aplicació que posi en comú els dos conceptes. En les següents seccions, es descriurà el model de cadascuna de les dues aplicacions, aportant una idea general de com s'han adaptat els conceptes perquè encaixin i poder dur a terme després una implementació que dóni una solució al problema.

Val a dir que a banda de concebre un bon disseny des de zero, cal coneixer quines són les eines que es pretenen fer servir (Hadoop, en aquest cas) per a poder optimitzar la implementació de la millor manera possible. L'avantatge de Hadoop en aquest punt, des del punt de vista de la programació, és que la seva API té una correspondència molt gran amb el paradigma que implementa (MapReduce),

[1] A la xarxa es poden trobar múltiples conjunts de pàgines web recopilades per crawling.

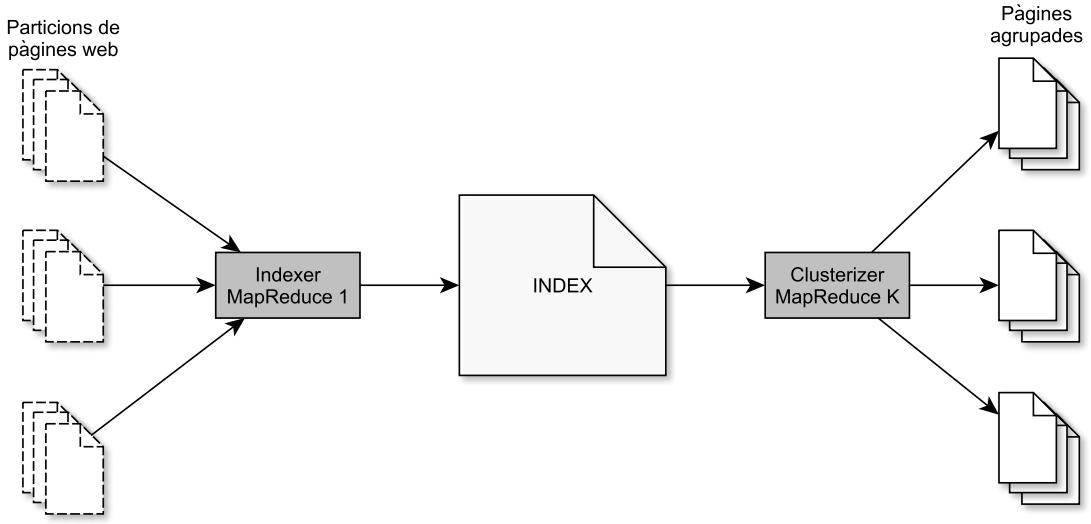

Figura 3.1: Execució gobal del procés. Les pàgines web s'indexen i es construeix un únic índex amb tots els vectors de característiques dels documents. Aquests vectors després es passen al programa de clústering, que fa les agrupacions.

en comparació amb altres eines que per limitacions del llenguatge o la plataforma poden no ser tant fàcils de programar. Aquest fet afavoreix que el disseny i la implementació mantinguin, encara més, una distància més estreta i que, per tant, la implementació a partir del disseny sigui quelcom més trivial.

A continuació es presenten les seccions que expliquen el disseny de les dues aplicacions del procés. Totes dues segueixen un model MapReduce per poder accelerar el temps de processament. La primera aplicació, com s'ha citat anteriorment, es correspon amb l'indexador de pàgines, el qual aglutina i preprocessa els documents web. Abans, però, de parlar de l'aplicació, convé fer una explicació de les dades i de com han d'entrar a l'algorisme K-means per tal que siguin útils.

3.1 Les dades: representació de característiques

Recordant el que s'ha dit en la introducció d'aquest capítol, les dades d'entrada i de les quals es disposa, es conformen d'un conjunt de documents HTML extrets per mitjà d'un procés de *crawling* previ. Així doncs, cada pàgina es representa en aquest punt com un document HTML (amb els tags, els elements, etc.).

Per tal de realitzar el clústering, cada document ha de quedar expressat en forma de vectors de característiques (figura 3.2), als quals fós possible aplicar una mesura de distància. El dilema es conèixer què es vol comptabilitzar, és a dir, quines són les mètriques per tenir en compte i expressar els vectors.

URL 1	0	3	10	0	0	1	43	2	4	5	7	3	0	0
URL 2	2	3	0	1	3	0	0	57	15	5	3	1	0	10
URL 3	5	0	3	0	30	0	5	3	1	0	0	0	1	0
URL 4	30	0	5	3	0	0	1	2	3	3	1	0	10	3
URL 5	15	5	3	1	0	1	2	0	1	3	0	5	3	1

Figura 3.2: Representació vectorial dels documents. Cadascun dels documents queda representat per un vector numèric de característiques.

En els cas de fitxers o documents de text, el més trivial és comptabilitzar les aparicions dels mots. I és, de fet, la opció que es tria en aquest disseny. Amb tot, dos documents seràn semblants i, per tant, pertanyeran al mateix clúster si tenen els suficients mots en comú i en quantitats similars. Aquests vectors de freqüències han de tenir la mateixa mida per tots els documents, i és per això que a l'índex s'han d'aglutinar tant termes com documents, de tal forma que en cada vector apareguin tots els termes, encara que el document no en tingui uns quants (figuren amb valor 0).

A banda dels termes, els quals es tenen en compte al tag HTML *body*, en els documents web, a la capçalera, sovint s'hi troba més informació que queda invisible al resultat final però aporta informació del contingut de la web, de cara a la web semàntica. Aquesta informació podria ser també recollida en l'índex i formar part del vector de característiques dels documents, afegint-ho al vector de termes. Ara bé, no tota la capçalera és útil. Un dels tags comuns per afegir informació semàntica és el tag *meta*, el qual és utilitzat, entre altres usos, per afegir *keywords* a la pàgina (per qüestions d'indexació de cercadors i posicionament).

Així, aquestes *keywords* passaràn a formar part del vector de característiques, aportant més informació per agrupar. Tanmateix, l'ús d'aquestes *keywords* comporta un problema i és que en no tots els documents web hi figuren, fet que pot provocar *soroll* a l'hora de mesurar distàncies. És per això que a la mesura de les distàncies se li haurà de donar un pes tant als termes com a les *keywords* per pal·liar aquesta diferència.

En conclusió, cada document queda expressat com un conjunt de dos vectors (figura 3.3) en els quals figuren les freqüencies tant de termes com de *keywords*. La distància entre els vectors amb els pesos, donarà la diferència entre documents.

URL 1	fk_1	fk_2	fk_3	fk_4	...	fk_m
	ft_1	ft_2	ft_3	ft_4	...	ft_n
URL 2	fk_1	fk_2	fk_3	fk_4	...	fk_m
	ft_1	ft_2	ft_3	ft_4	...	ft_n
URL 3	fk_1	fk_2	fk_3	fk_4	...	fk_m
	ft_1	ft_2	ft_3	ft_4	...	ft_n

Figura 3.3: Representació vectorial dels documents amb *keywords*. Cadascun dels documents queda representat per dos vectors, un de termes (t) i un de *keywords* (k). A la figura, fk_i indica la freqüència de *keywords* en un document per la *keyword* i per m *keywords* en total i ft_i indica la freqüència de termes en un document pel terme i per n termes en total. Generalment $m \ll n$.

3.2 Indexador

A la secció 3.1 s'ha parlat de com s'han de representar les dades. Ara que queda clar quina ha de ser la sortida de l'indexador, en aquesta secció s'explicarà com aconseguir aquesta sortida seguint el model MapReduce.

S'ha de tenir ben present com es volen repartir les tasques Map i Reduce al llarg del procés, és a dir, quines parts del codi es poden realitzar en paral·lel (map) i quines d'aquestes tenen dades en comú o que necessiten ser agrupades (reduce).

L'aproximació més senzilla a quines han de ser les entrades dels maps, és fer que cada document en sigui una. Cada document, doncs, serà processat per un map. El procés constarà de la neteja del codi propi de HTML i la obtenció de la informació que és d'interès: el text del cos i les *keywords*. La ID del document (url) i els vectors **propis del document** de termes i de *keywords* formen el conjunt del document processat. Aquest és el codi paral·lelitzable.

Malauradament, de reducer només n'hi pot haver un, ja que cal aglutinar tots els termes i totes les *keywords* per tal d'obtenir els vectors amb el total de característiques. D'aquesta manera, l'entrada del reducer serà una llista de documents processats. A partir d'aquest conjunt, s'ha de construir l'índex complet. Aquest índex constarà d'una matriu per termes i una matriu per keywords. Cada fila de les matrius representarà un document i cada columna, el terme o la *keyword* de torn. Cada cel·la, doncs, serà la freqüencia en la qual un terme/*keyword* apareix en un document.

3.2.1 Filtratge de l'índex

Un cop recopilades totes les característiques, és necessari un procés de filtratge per a l'índex sencer. Aquest filtratge elimina el *soroll* que poden causar termes que no siguin definitoris pel clústering. Que un terme no sigui definitori vol dir que proporciona informació errònia a les caracterítiques d'un document, podent fer entrar en un cluster un document que no hi ha d'entrar. Existeixen dos casos pels termes no definitoris:

- **El terme no apareix en un mínim de documents**. Per exemple, un terme que només apareix en un document, o en un percentatge molt petit de documents, no aporta cap informació per fer el clústering, ja que es tracta d'un terme tan exclusiu que no serveix de res comparar-ho amb els altres documents que no el tenen, que són la majoria.
- **El terme apareix en la majoria de documents**. En aquest cas, el problema rau en què aquests termes són tan comuns que fan que els documents s'assemblin massa. A l'hora de fer el clústering, aquests termes s'han d'eliminar, o contràriament, tots els documents aniran a parar al mateix cluster.

Un cop estudiats els casos que s'han d'eliminar cal, havent construit l'índex, determinar quines són les característiques que s'han d'eliminar. Per fer-ho cal comptabilitzar les columnes de les matrius anteriors i comptar quantes cel·les són > 0, o d'altra manera, aquells documents que contenen aquesta característica. Un cop obtinguda la mesura, cal fer la raó del número d'ocurrències entre el número total de documents (3.1) i comprovar que estigui entre un màxim i un mínim. Si no es compleix això, el terme s'elimina.

$$r_j = \frac{\sum_{i=0}^{m} T_{ij} | T_{ij} > 0}{n_{docs}} \tag{3.1}$$

Amb l'índex contruït i filtrat, la sortida del reducer ha de ser el parell de matrius de característiques, les quals s'escriuen a disc (HDFS).

3.3 Agrupador (clústering)

La segona aplicació és el propòsit del projecte i és l'aplicació que fa les agrupacions, el clústering. Implementa l'algorisme K-means adaptat al paradigma MapReduce i és la part del procés amb més càrrega computacional. En aquesta secció s'explica el disseny d'aquesta aplicació, que per entendre'l, s'ha de passar a veure com s'implementa K-means amb MapReduce.

Identificar el codi paral·lelitzable és el primer pas i és el més important, perquè defineix l'estructura base del programa. És el primer que s'explicarà en aquesta secció. Això passa per identificar les parts del codi en les quals no existeixen les dependències entre les dades. Normalment, aquestes parts es corresponen amb els mappers.

Si s'examina l'algorisme K-means (algorisme 1) hi ha una part que fàcilment es pot correspondre amb el model dels mappers i és el càlcul de la distància de cada mostra (document) amb cada centroide. Si s'observa, no hi ha dependència de cap dada, ja que per calcular aquesta distància només cal llegir dades que són constants. Així doncs, cada mapper calcula la distància entre un document (o conjunt de documents) i cada centroide i assigna, a partir d'aquestes distàncies, el document a un centroide (cluster).

El que s'obté d'això és un conjunt de K centroides amb una llista de documents assignats. Aquest model (figura 3.4) és ideal per passsar-ho des dels mappers als reducers i fer que cada reducer recalculi cada centroid a partir de les llistes de documents assignats que té.

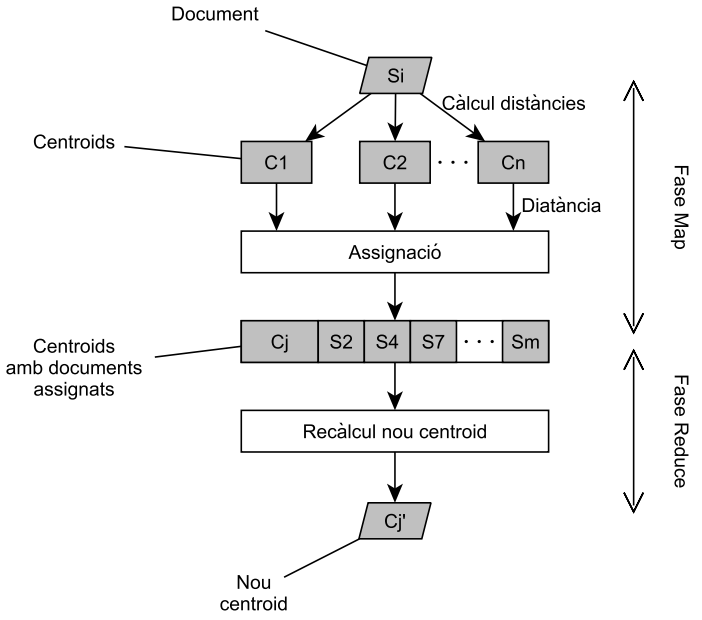

Figura 3.4: Iteració de K-means amb MapReduce. Cada mapper calcula la distància dels documents amb cada centroide i l'assigna a un clúster. Els reducers recalculen cada centroide.

Pel que fa a les entrades i sortides de cada iteració, el mapper rep un conjunt de descriptors de documents, els quals es processen. La sortida queda conformada amb un identificador del centroid (key) i el document o documents assignats a aquest

centroide (value). Pel que fa el reducer, a l'entrada reb l'identificador de centroide (key) amb la llista de documents assignats (value) i a la sortida dóna la url d'un document (key) amb l'identificador del centroid assignat[2] (key).

El que s'explica fins aquí es correspon amb una iteració de l'algorisme. Paral·lelitzar un procés iteratiu no és senzill (existeix una dependència de dades entre iteracions), però si es paral·lelitzen les seves iteracions de forma independent s'aconsegueix un millor rendiment.

Amb tot, cada iteració de l'algorisme consistirà en un job MapReduce que es llençarà des d'un programa controlador (figura 3.5). Aquest programa és l'encarregat de la gestió de cada job i és qui manté el bucle de K-means, comprovant a cada iteració que es compleix la condició d'aturada. La condició és que la diferència entre els centroides d'una iteració i l'anterior no canvien o que s'arriba a un nombre màxim d'iteracions. A part del control del bucle, l'aplicació gestiona els paràmetres necessaris (alguns itroduïts per l'usuari) per l'execució.

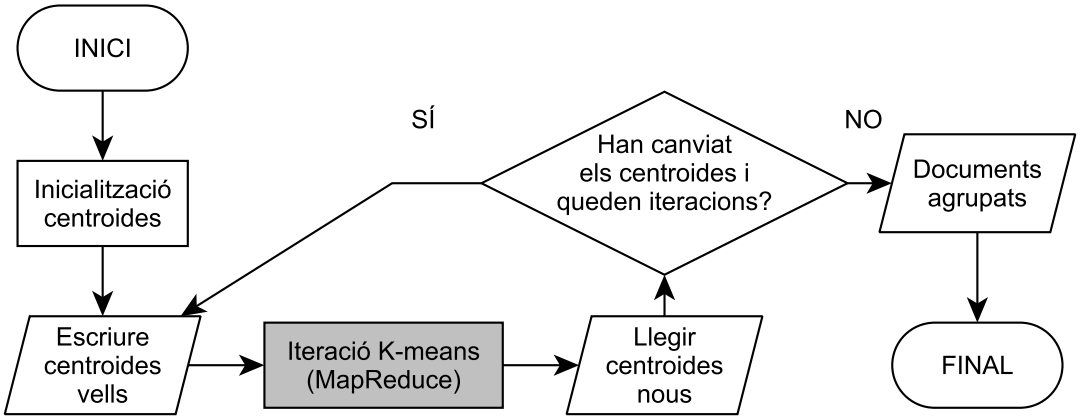

Figura 3.5: Programa controlador. El flux d'execució contempla diversos jobs MapReduce, fent que cada job sigui una iteració del K-means. El programa controla la condició d'aturada, que en aquest cas és comprovar que els centroides han canviat de forma suficient entre una iteració i una altra. La 'Iteració K-means es correspon amb la figura 3.4.

[2]Recordar que a banda de passar els centroides assignats es recalculen els nous per la següent iteració.

Capítol 4

Implementació

La posta en escena d'un disseny és la seva implementació. D'implementacions n'hi pot haver tantes o més com plataformes en les quals dur-les a terme. La implementació del disseny ha de ser tant fidel i optimitzada com sigui possible, de tal manera que les limitacions de la plataforma siguin mínimes. En aquest capítol es revisarà quina ha estat la implementació del disseny proposat per aquest projecte, apropant al codi l'explicació sense arrivar a verure-ho per tal d'aclarir allò que no hagi quedar del tot clar en l'explicació del disseny.

MapReduce pot tenir moltes implementacions. Una d'elles és Hadoop, però n'hi ha més. De forma més o menys anàloga, el disseny proposat anteriorment pot tenir diverses implementacions. Per exemple, enlloc de Hadoop es podria haver fet servir un altre framework, fent que la implementació canvïi dràsticament respecte a la que aquí es proposa. Tot i així, el disseny, en general, s'ha de manternir.

Veure la capa d'implementació suposa baixar el nivell d'abstracció, és a dir, observar el programa de forma més propera al codi que no pas si només es té en compte el disseny. No obstant, no cal baixar a nivell de codi per entendre com s'ha implementat el programa, i és per això que aquí el codi tampoc s'explicarà.

Aquest capítol, doncs, està estructurat de forma similar a l'anterior, fent una explicació del flux d'execució de cada programa però a un nivell més baix. Naturalment, no es pretén puntualitzar cada aspecte concret dels programes.

Serà important, a més, fer mencions especials a conceptes o aspectes relacionats amb Hadoop i com el framework gestiona dades i funcionalitats. El sistema distribuït de fitxers de Hadoop, HDFS, juga també un rol molt rellevant en aquesta implementació i serà convenient veure com s'utilitza.

4.1 Indexador

Es podria dir que l'indexador és una extensió d'un comptador de paraules. Apache proporciona un tutorial de MapReduce [2] on fa servir com exemple un programa d'aquest tipus. El programa executa un job MapReduce en el qual llegeix cada entrada, compta paraules (map) i les suma (reduce). Per a l'indexador (figura 4.1) s'ha partit d'aquesta base, amb la diferència que els termes s'han d'agrupar d'una altra forma i la sortida no ha de ser la mateixa. A més, cal considerar que l'entrada no es composa de fitxers de text plans, sino que són documents HTML i, per tant, hi han cadenes que no s'han de computar.

A la fase map, cada un dels mappers agafa una entrada o document. L'entrada a priori té tot el text pla propi d'un document HTML, la qual cosa vol dir que cal netejar-ho. Existeixen llibreries que processen un text HTML i permeten accedir als elements d'una forma més senzilla. En aquest cas s'ha fet servir jsoup, un parser HTML per Java. Processat el text HTML, s'accedeix al text del tag *body*, el qual se separa en tokens (termes), que s'afegeixen al llistat del document. Es fa el mateix accedint al tag *meta* amb l'atribut *name=keywords* per recollir les paraules clau. Tant el llistat de termes com el de *keywords* s'adjunten a una tupla, juntament amb la URL del document, la qual actua com identificador.

Les tuples de documents processats són conformades amb tres atributs: la url del document, una taula hash per termes i una taula hash per *keywords*. Les taules hash indiquen el terme o la *keyword* i la seva freqüència al document.

Cada map dóna com a sortida un enter (1) com a clau i la tupla amb la url, la llista de termes i la llista de *keywords* del document com a valor. Realment només els valors són d'interès, ja que la clau per a tots els maps ha de ser la mateixa. La raó és que el reducer ha de rebre tots i cada un dels documents processats per construir l'índex global, i la forma d'aconseguir un sol reduce és que la clau de la sortida dels maps hagi de ser la mateixa.

La fase reduce s'encarrega de construir les matrius globals per tots els termes i *keywords* de tots els documents. L'entrada del reducer és la clau dels mappers (que no aporta informació) i una llista de documents processats. Es fa un recorregut d'aquesta llista obtenint les freqüències de termes i *keywords* de cada document. Cada llista passa a ser una fila d'una matriu en un objecte Index (el qual s'explicarà més endevant). Aquest índex pateix un procés de filtratge un cop examinats tots els documents, per tal d'eliminar els ítems que apareixen massa o massa poc en el conjunt de documents.

L'índex global és l'objecte més important de la fase reduce. És l'estructura on totes les característiques queden agrupades i del qual s'obtenen les matrius de

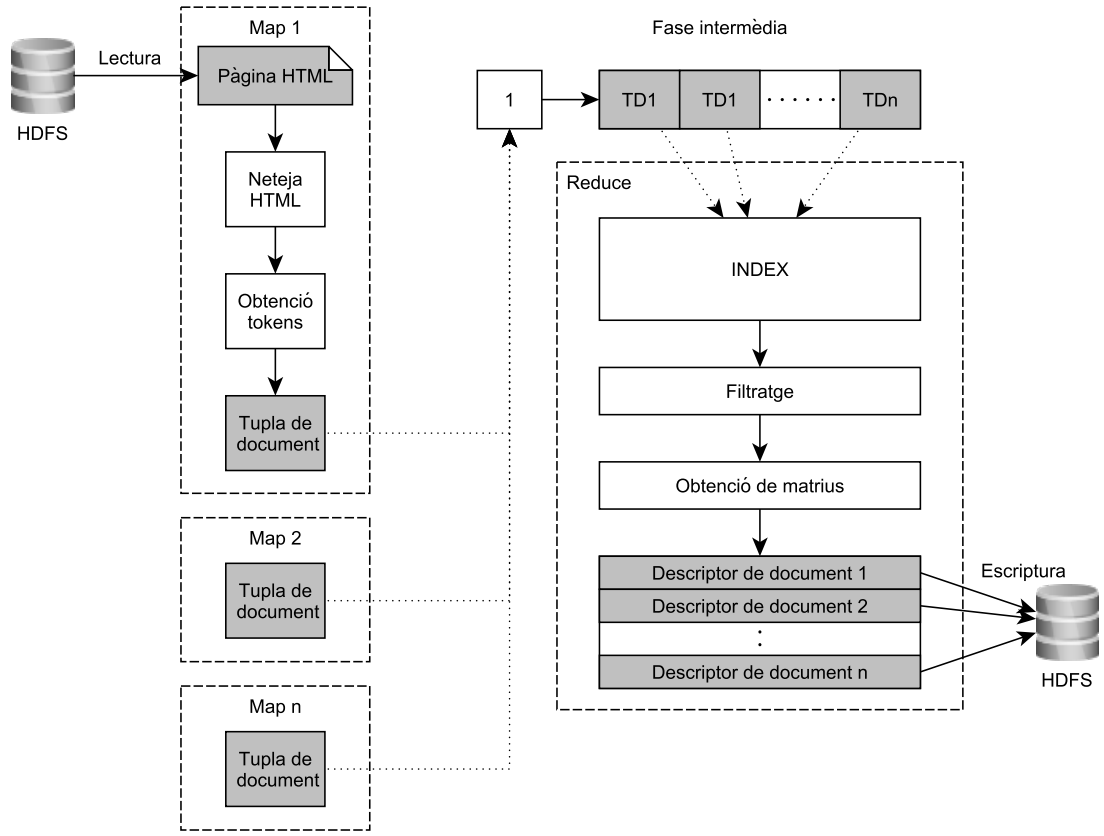

Figura 4.1: Implementació de l'indexador. Cada map realitza les mateixes tasques que el map 1, però no s'han inclòs als altres per simplificació. Notar que totes les tuples de document es passen al reducer amb la mateixa clau per tal que només hi hagi un reducer que processi totes les tuples.

característiques dels documents. És molt necessari ja que per fer el clustering es necessita que tots els documents estiguien representats per vectors de la mateixa mida, cosa que no es així a la sortida dels mappers perquè es desconeix el nombre total de paraules.

L'estructura base interna de l'index té una taula de documents (comú per termes i *keywords*), una taula de termes i una taula de freqüències. La mateixa estructura s'aplica per les *keywords*. La taula de documents guarda la url del document i un identificador sencer. La taula de termes guarda els termes i un identificador sencer. Per la seva banda, la taula de freqüències guarda la correspondència entre identificadors de terme (files) i identificadors de document (columnes). Aquest sistema permet guardar només una vegada els strings de paraules i documents. A més l'accés i la manipulació de dades són senzills i econòmics en termes de memòria al fer servir Hashs. Afegir un nou document implica actualizar la taula de documents, la taula de termes (si cal) i la taula de freqüències.

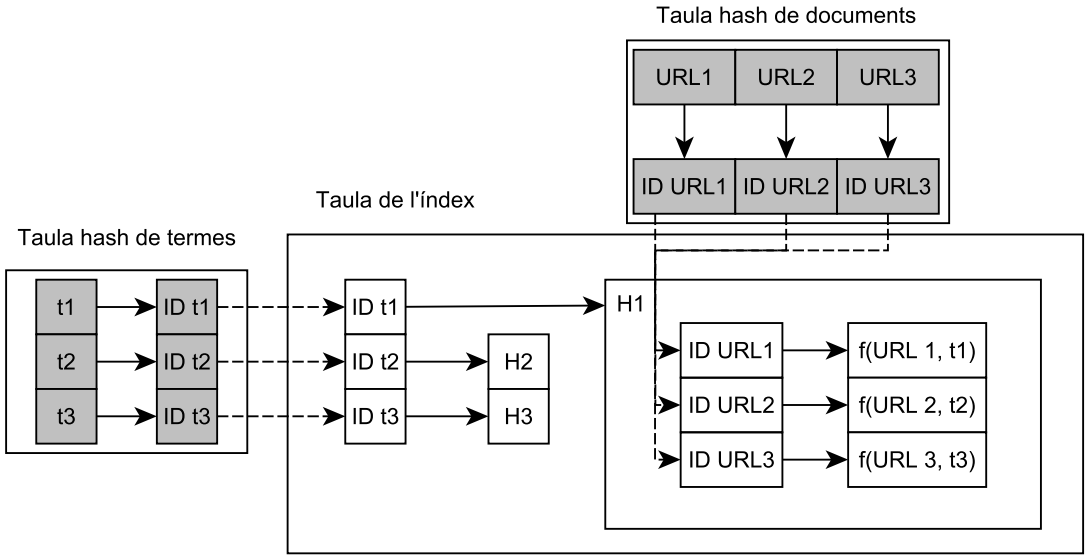

Figura 4.2: Taules de l'índex. La taula de l'índex està conformada per una taula hash de taules hash. El primer nivell indica el terme en qüestió i el segon, el document. El valor del segon nivell és la freqüència d'aquell document per aquell terme.

El procés de filtratge es realitza un cop omplertes les taules. Es fa un recorregut per la taula de freqüències i es comptabilitzen els documents en els quals apareix el terme. S'eliminen aquells termes no apareixen en menys d'un cert percentatge de documents, o bé apareixen en més d'un altre percentatge de documents.

Acabat el filtratge, la sortida del reducer està preparada per ser escrita al sistema de fitxers en forma de text pla. Cada línia del fitxer (o fitxers) de sortida es correspon amb un document i els seus vectors de característiques filtrats i separats convenientment amb caràcters delimitadors. Això conforma un descriptor de document.

4.2 Agrupador (clústering)

L'aplicació principal és més complicada, ja que li cal un programa destinat a controlar l'execució de les iteracions del K-means. A més, aquest programa ha de gestionar la comparació de centroides i el llençament dels jobs Hadoop de les iteracions.

El pas de dades dins Hadoop és possible mitjançant taules de paràmetres que implementa el controlador dels jobs. Els paràmetres necessaris per l'execució del clústering es transfereixen amb aquest sistema i seràn visibles des de qualsevol punt de l'execució Hadoop (mappers, reducers, entrada i sortida). Aquests paràmetres (com el nombre de clústers K) són especificats a la crida del programa per línia de

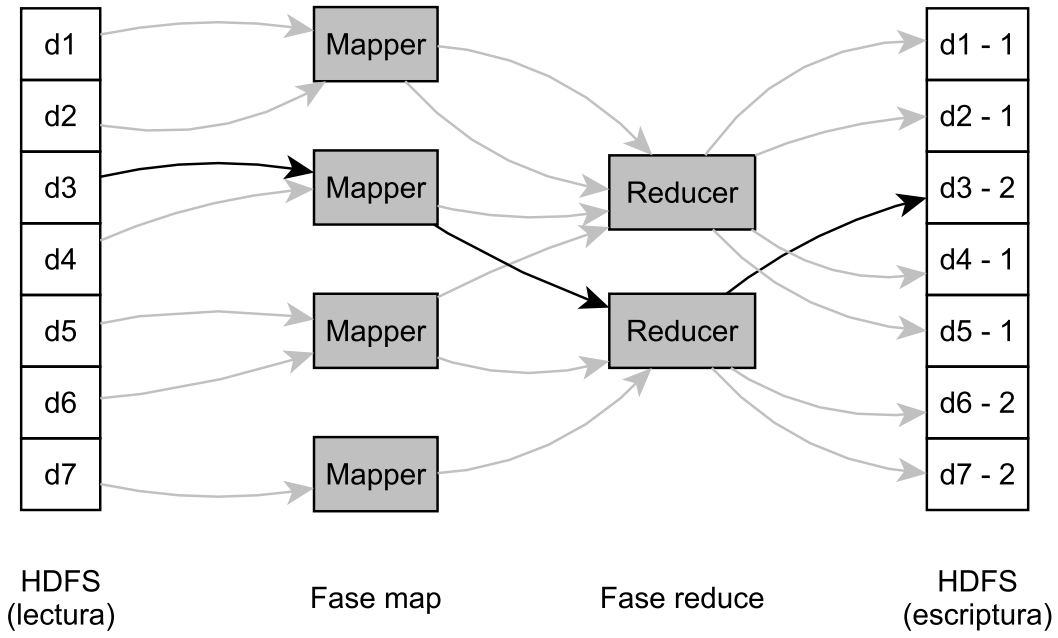

Figura 4.3: Assignació de documents per fases. El sistema de la figura es composa de set documents, cuatre mappers (hi han cuatre màquines) i dos reducers ($K = 2$). El nombre de documents per mapper es calcula de forma linial. Observar que el reducer de destí depèn de la distància respecte el centroide. El resultat final, que s'escriu al HDFS, és el conjunt de documents etiquetats.

comandes.

A la secció anterior s'ha explicat quina és la sortida de l'indexador. Aquesta sortida serà l'entrada dels mappers de les iteracions del clústering. Aquesta entrada no varia ja que els documents sempre són els mateixos[1]. És convenient examinar la granuralitat de la repartició dels documents als mappers. Es millor que cada mapper processi un sol document o que un mapper processi un conjunt de documents? S'ha de tenir en compte que un tasca, sigui map o reduce, implica un *overhead* causat per la comunicació per xarxa i la creació de la tasca. Un temps que no aporta res tangible al processament.

La càrrega associada a una tasca ha de ser suficientment gran per superar el temps que costa crear-la (l'*overhead*). Una repartició amb granuralitat molt fina pot donar la situació on el temps d'*overhead* és superior al temps de processament propi de la tasca, fent que no hi hagi guany a l'hora de paral·lelitzar el codi. Tenint en compte que en el cas de l'índex, cada document ha quedar reduït a una llista de sencers, fora convenient fer una repartició de més elements en menys tasques per obtenir un rendiment més òptim.

[1]La qual cosa vol dir que l'indexador només cal executar-lo un cop.

Aquest fet ha propiciat que la implementació de la lectura de l'índex tingui una granuralitat més gruixuda. El nombre de mappers és més petit que el de documents i per aconseguir una màxima paral·lelització minimitzant el temps d'*overhead* es fa que el nombre de mappers sigui igual al nombre de màquines (figura 4.3) amb un TaskTracker, o, el que es el mateix, màquines esclaves, encarregades del processament. Aquest paràmetre és introduït per l'usuari. Cada línia del fitxer d'índex representa un document i per saber quants documents corresponen a cada mapper, convé saber-ne quants n'hi ha. L'índexador ha de guardar al HDFS un fitxer amb aquesta dada abans d'acabar l'execució, fitxer que es llegirà a la fase de lectura dels mappers.

Com ja s'ha dit, els documents són a l'entrada del mapper, però falten els centroides, que no formen part de l'entrada. La comunicació per taules de paràmetres no és aconsellable en aquest cas, degut a que els centroides són formats per més dades que no pas una cadena o un sencer. Cal fer operacions amb l'HDFS. Cada tasca map o reduce llegeix o escriu els centroides del HDFS. Abans de llençar la iteració de K-means, el programa controlador escriu a disc els centroides "vells" i al final de la iteració llegeix els "nous" per fer la comparativa. A la primera iteració, com que no existeixen centroides, s'inicialitzen uns d'aleatoris (veure figura 3.5).

Dins el mapper, cada document del conjunt d'entrada es compara amb els K centroides llegits del HDFS i s'assignen al més semblant, és a dir, el centroide amb la distància més petita. La comparació es fa a terme mitjançant la distància euclidiana dels vectors de termes (4.1) i de *keywords* i ponderant-ho amb uns pesos introduits per l'usuari com a paràmetre (4.2).

$$d_e(u, v) = \sqrt{\sum_{i=1}^{n} (v_i - u_i)^2} \tag{4.1}$$

$$d(D, C) = w_k \cdot d_e(D_k, C_k) + w_t \cdot d_e(D_t, C_t) \tag{4.2}$$

On $w_k \in [0, 1]$ i $w_t \in [0, 1]$.

Amb tots els documents assignats, la sortida del mapper es compòn dels enters identificadors dels centroides com a claus i els documents com a valors.

Cada reducer, per la seva banda, ha de recalcular un dels centroides. Es recorda que l'entrada aquí és l'enter del centroide com a clau i el conjunt de documents assignats com a valor. Per fer el càlcul del nou centroide cal calcular el vector de mitjanes dels documents assignats i per fer-ho es construeix una matriu amb els vectors de freqüències. L'element i d'un dels vectors del nou centroide serà, doncs, la mitjana de l'element i dels vectors dels documents.

Quan es calcula el nou centroide n de la iteració i, a més, es calcula la distància respecte el centroide anterior n de la iteració $i - 1$. I, per acabar, s'escriu el nou centroide a l'HDFS.

Així conclou el job Hadoop d'una iteració del K-means. Tot i així aquí no acaba. Un cop acabat el job, el programa controlador recull les distàncies dels centroides calculades als reducers i si totes són 0 o s'arriba a un nombre d'iteracions màxim es considera que el procés acaba i els clústers són fets.

4.2.1 Documents "stub"

Val la pena fer una menció a aquest mecanisme de la implementació. I és que en el pas del mapper al reducer, es pot donar una situació que podria derivar en uns resultats incorrectes o no tan bons. El problema és causat pel propi paradigma MapReduce. Recordant la sortida del mapper, cada document és assignat al centroide més proper a ell. Però, què succeeix si hi ha un centroide al qual no se li ha assignat cap document? El resultat seria que el reducer d'aquest centroide no s'executaria i, per tant, no es recalcularia, és a dir, faltarien clústers.

La forma que s'ha ideat per solucionar aquesta problemàtica és fer servir uns documents "stub"[2] al mapper si es dóna la situació abans descrita. Això és, si no hi ha cap document assignat a un centroide, el centroide es passa a la fase reduce d'igual manera, però amb un document fals marcat com a document stub. El reducer d'aquest centroide rebrà d'igual forma l'identificador del centroide i una sèrie de documents, en els quals hi poden haver documents stub, que s'ignoren. Si tots els documents rebuts són documents stub, el nou centroide serà el mateix que l'antic, és a dir, no es mourà.

[2] Aquí, el terme *stub* s'utilitza fent referència al testeig del software. En aquesta matèria, un stub és un tros de codi que simula un comportament, com passa en el cas d'aquests falsos documents.

Capítol 5

Experimentació i resultats

L'obtenció de resultats experimentals pot donar una aproximació de la qualitat del software desenvolupat a més de donar peu a unes conclusions que podrien no esperar-se. En aquest capítol s'explica quins experiments s'han realitzat amb el software desenvolupat a més d'altres aspectes d'interès relatius al projecte.

5.1 Experiments

En aquesta secció s'exposaran les diferents proves fetes amb el software desenvolupat. A partir d'aquestes proves i l'anàlisi de les mateixes es podrà extreure una sèrie de conclusions.

Les proves que es presenten en aquest apartat analitzen el rendiment per veure com es comporta Hadoop amb una configuració de clúster o una altra. Tot i així també es presenta un test qualitatiu, per comprovar l'eficàcia del clústering i veure que es realitza de forma encertada i el més acurat possible.

5.1.1 Infraestructura

Les proves locals de depuració i els tests qualitatius s'han fet sobre una sola màquina amb Hadoop 1.0.4 amb configuració psudodistribuïda d'un node sobre Mac OS X v10.8.3 (Mountain Lion). Aquesta màquina és un MacBook Pro (mitjans 2012) amb una CPU Intel Core i5 a 2.5GHz i 4GB de memòria RAM. Pels experiments s'ha instal·lat Hadoop 1.1.2 en un clúster d'una vintena de màquines connectades amb xarxa local (LAN). Aquesta distribució de Hadoop corre sobre Kubuntu, estant cada màquina equipada amb un processador Intel Core i7 i 8GB de memòria RAM.

5.1.2 Dataset

La idea original consistí en fer servir els serveis web d'Amazon (AWS) tant per fer l'execució dels programes desenvolupats com per la obtenció de les dades. Tot i així, davant la possibilitat de disposar d'un clúster de màquines propi i poder instal·lar i executar-hi les aplicacions desenvolupades, s'ha optat per descartar l'execució a Amazon.

El problema rau en les dades. Amazon, al seu servei d'emmagatzematge S3, disposa d'una sèrie de data sets d'ús públic. Entre ells, el *Common Crawl Corpus*. Un conjunt de 81TB de documents web obtingut en una data determinada. L'accés a aquestes dades però, és la raó per la qual no s'han acabat fent servir aquí, i és que no es poden descarregar de forma estàndar. L'accés es fa mitjançant aplicacions executades als serveis d'Amazon.

Davant d'aquest problema, l'enfocament ha estat el de fer una extracció pròpia de documents web. Obtenir dades del tipus documents web és un procés relativament senzill i barat, basat en fer un recorregut dels enlaços de les pàgines web a partir d'una (o més d'una) URL llavor. Aquest procés és conegut com *crawling*.

L'organització *Internet Archive* posa a disposició el software Heritrix [7], una aplicació que, donada una llista de URLs llavor, realitza un procés de crawling i emmagatzema el contingut de les pàgines en fitxers amb format ARC[1] (ARChive). Aquest format emmagatzema els documents en text pla i afegeix a cadascun una capçalera amb informació de l'extracció. És comú trobar aquests fitxers comprimits amb *deflate* (amb l'extensió .arc.gz) la qual cosa, a més a més, permet guardar, en un sol fitxer contenidor físic diversos fitxers comprimits concatenats. Aquest format és el format d'entrada de l'indexador del projecte. Cada mapper de l'indexador reb un registre ARC de cada fitxer d'entrada i el processa.

Fent servir el software Heritrix, doncs, s'ha fet l'extracció de les dades necessàries per fer les proves, tot i que no són tantes com les que es poden aconseguir a Amazon.

5.2 Proves qualitatives

Les proves qualitatives mesuren l'eficàcia del clústering amb diferents configuracions. L'índex, en aquest cas, serà el mateix per a totes les proves i construït a partir de les mateixes dades.

El clúster per aquestes dades està configurat amb 8 màquines més 1 node màster (que no realitza processament de dades). Les dades que s'utilitzen per aquesta prova pesen, comprimides amb *deflate* (GZ) 79.3MB. S'han obtingut mitjançant el crawler

[1] No confondre amb el format de compressió ARC

Heritrix de dos websites diferents, de gencat.cat i de commoncrawl.org. La primera té com idioma principal el català i la segona, l'anglés. La prova, majoritàriament consistirà en fer dos grups dels dos dominis per separat.

Amb aquesta configuració, el procés d'indexació triga 20 segons. En total es comptabilitzen 881 documents, 324 *keywords* i 52 termes després del procés de filtratge. Abans de passar a veure els resultats però, convé fer una menció a l'existència de soroll a les dades. El crawler realitza un seguiment dels enllaços de les pàgines. Pot passar que alguns enllaços mostrin apuntin a un contingut inexistent o que estigui fora dels dominis abans mencionats. Per exemple, hi han enllaços de Gencat que apunten a una imatge perquè en el moment de cridar l'enllaç el contingut no existeix i es retorna un document HTML d'error 404. Això implica introduir unes dades que aporten soroll.

A continuació es passa a veure les mesures de les proves. Per una banda es mesurarà l'error del clústering amb la configuració que ha donat els millors resultats i tot seguit, quines aquelles configuracions han donat més bons resultats, quines no i per què.

Com s'ha dit, es pretén fer dues agrupacions de dos conjunts de documents i veure si els clústers es formen de manera correcta. Per mesurar això, s'examinarà la sortida del K-means en fitxers de l'HDFS. L'estructura d'aquests fitxers és que a cada línia s'escriu la url del document, un caràcter tabulador i un sencer corresponent amb l'etiqueta del clúster. Realitzant un primer recompte del fitxer s'obtenen les dades expressades en la taula 5.1.

Mètrica	**Valor**
Documents de Gencat	535
Documents de Commoncrawl	162
Documents externs	184
Percentatge d'encert amb Gencat	99,6%
Percentatge d'encert amb Commoncrawl	36,1%

Taula 5.1: Taula de mesures qualitatives. Els percentatges d'encert es calculen amb la raó entre el nombre de documents d'un dels dominis en el clúster que li correspon entre el nombre total de documents del mateix domini.

A la taula 5.1 es pot veure com la classificació dels documents de Gencat és força més bona que la classificació de documents de Commoncrawl.

Canviant el punt de vista de les mètriques, es pot fer una mesura per clústers. A la taula 5.2 es poden veure aquestes mesures.

En contra de la taula 5.1, la taula 5.2 proporciona uns resultats més bons pel

Mètrica	**Clúster 1**	**Clúster 2**
Nombre de documents	61	820
Documents de Gencat	2 (3.3%)	504 (61.5%)
Documents de Commoncrawl	53 (86.9%)	109 (13.3%)
Documents externs	6 (9.8%)	207 (25.2%)

Taula 5.2: Mesures qualitatives des del punt de vista dels clústers. Indica el nombre de documents de cada tipus per clúster.

conjunt del primer clúster, on la homogeneïtat dels elements és major que al segon, al qual la majoria de docuemnts que han de pertanyer al clúster és menor. Tot i així, en contrast, gairebé tots els documents del segon conjunt (Gencat) han caigut al cúster que els hi toca.

A banda del soroll introduït per les pàgines externes, la causa del soroll pot venir per la sel·lecció de característiques. El fet de realitzar el filtratge i de la pròpia homogeneitat de les dades propicia uns valors que no són del tot fiables a primera vista.

La taula 5.3 mostra, per diferents configuracions, el nombre de documents per clúster.

Pes keywords	**Pes termes**	**Clúster 1**	**Clúster 2**
0.0	1.0	0	881
0.1	0.9	61	820
0.2	0.8	61	820
0.3	0.7	61	820
0.4	0.6	61	820
0.5	0.5	0	881
0.6	0.4	60	821
0.7	0.3	881	0
0.8	0.2	881	0
0.9	0.1	881	0
1.0	0.0	881	0

Taula 5.3: Mesures qualitatives des del punt de vista de la configuració. Indica el nombre de documents de cada tipus per clúster segons la configuració dels pesos de càlcul de distàncies.

La configuració dels pesos és molt important a l'hora d'obtenir uns resultats acceptables. Observant els resultats obtinguts amb cada configuració, els millors s'obtenen amb el pes de les *keywords* de 0.1 a 0.4 i el pes dels termes de 0.9 a 0.6. En la majoria de conjunts de dades és important donar un pes major als termes

que a les *keywords*, ja que de termes en tenen tots els documents, però de *keywords* no, per tant, els que defineixen en major mesura un document són els termes. Les *keywords* reforcen la definició dels documents.

5.3 Proves de rendiment

Aquest conjunt de proves mesurarà el rendiment temporal del sistema amb diferents configuracions tant de clúster com d'arguments.

Per aquest conjunt de proves s'ha realitzat un procés de crawling situant les llavor a diversos lloc web. La mida d'aquest dataset, comprimit amb *deflate* (GZ) és de 400.6MB, havent-hi un total de 7834 documents HTML.

Pel procés d'indexació, la figura 5.1 mostra el que es triga en realitzar l'índex del dataset. S'observa com per un clúster de 5 a 8 màquines la variació en el temps és a la pràctica la mateixa. A partir d'unes 10 màquines però, el temps augmenta notablement. La raó és que tot i que hi ha més màquines que poden realitzar un processament, aquest fet introdueix un temps extra d'overhead (causat per la saturació de la xarxa i sincronització de nodes), el qual fa que no s'aprofiti al 100% la capacitat del clúster.

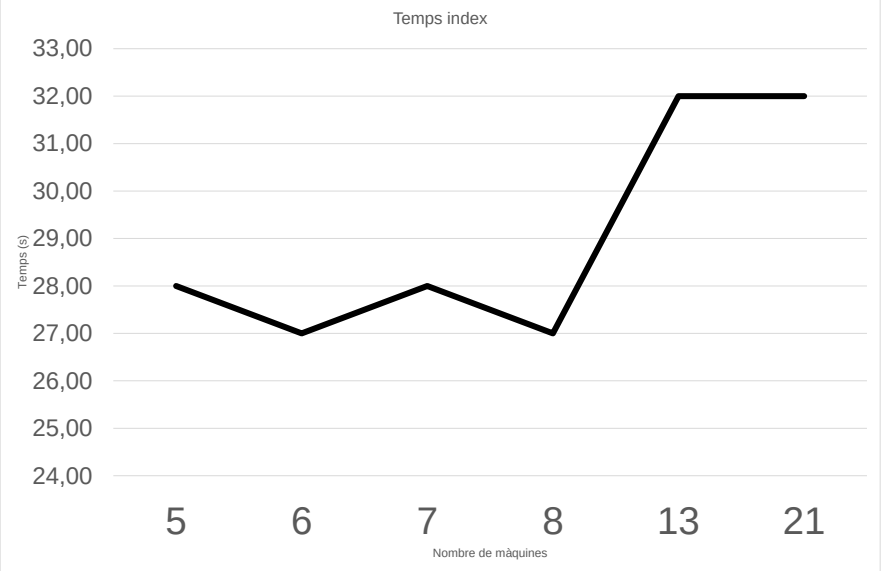

Figura 5.1: Gràfica de rendiment de l'índex. L'eix d'abcisses mostra el nombre de màquines amb el que s'ha configurat el clúster i l'eix d'ordenades, el temps en el qual s'ha fet la prova.

Pel que fa els test del programa de clústering, cal tenir en compte abans, com en l'altre conjunt de proves, certes consideracions per la interpretació dels resultats. Com se sap, l'algorisme de clústering necessita inicialitzar els centroides de forma aleatòria. Aquesta aleatorietat provoca que la convergència de l'algorisme no sempre sigui la mateixa per a tots els casos, fent que pugui trigar més o menys iteracions en convergir. És per aquesta raó, per la qual la mètrica que s'ha emprat com a mesura de rendiment del procés de clústering és el nombre d'iteracions per segon que el programa ha fet en cada prova per diverses configuracions del clúster (diversos nombres de màquines). El gràfic de la figura 5.2 mostra aquesta mètrica.

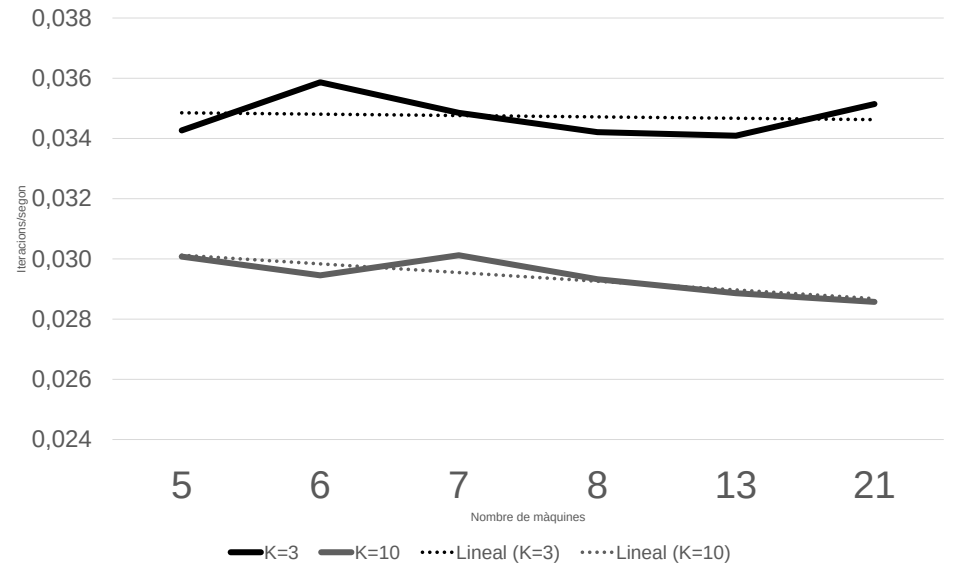

Figura 5.2: Gràfica de rendiment. L'eix d'abcisses mostra el nombre de màquines amb el que s'ha configurat el clúster i l'eix d'ordenades, el nombre d'iteracions per segon amb el que s'ha fet la prova. S'afegeixen, a més, les rectes de regressió per observar la tendència.

Per aquestes proves, les dades utilitzades són les mateixes indexades per a la prova de rediment de l'indexador i fent servir també les mateixes configuracions de clúster. Per cada configuració s'ha fet dues passades a l'algorisme: una amb $K = 3$ i una altra amb $K = 10$ i s'han comptabilitat les iteracions per segon que ha fet el programa en cada cas. El límit d'iteracions s'ha fixat a 20 en tots dos casos.

El primer que cal notar és que amb $K = 3$ les iteracions són més curtes que amb $K = 10$, naturalment pel fet que s'han de calcular distàncies per una quantitat menor de centroides i, a més, es recalculen menys centroides. D'altra banda, cal notar també que la tendència és fer menys iteracions per segon a mesura que s'augmenta

el nombre de màquines. Com a l'altre cas, aquest problema ve donat per la càrrega extra del framework que no és temps de processament (overhead).

Tots aquests resultats porten a la conclusió que cal configurar el clúster de Hadoop amb el nombre de màquines adient per a cada quantitat de dades (i.e. Si les dades són poques, es triga més en fer tasques de xarxa i sincronització que en fer les tasques de processament). A més, tot i que ja es té en compte en aquest programa, és important balancejar la feina de processament en cada fase de forma equitativa entre els nodes del clúster per evitar nodes sense feina i/o nodes que treballin en excés, aconseguint així una paral·lelització òptima.

5.4 Gestió del projecte i costos

En aquesta secció es farà un resum de la metodologia de treball i la gestió de projecte, així com una repassada als costos temporals.

En quant a metodologia de treball, s'ha establert un horari (taula 5.4) de treball setmanal i una reunió d'entre una hora i dues amb el director del treball, també setmanal.

Dilluns	Dimarts	Dimecres	Dijous	Divendres	Dissabte	Diumenge
0	0	0	1	9	5	4

Taula 5.4: Horari setmanal. Les hores expressades són en promig i no han estat les mateixes totes les setmanes.

Pel que fa la planificació temporal que s'ha seguit durant el semestre de primavera, a la figura 5.3 s'observa el diagrama de temps i de dependències del projecte, marcant el període aproximat de cada tasca general.

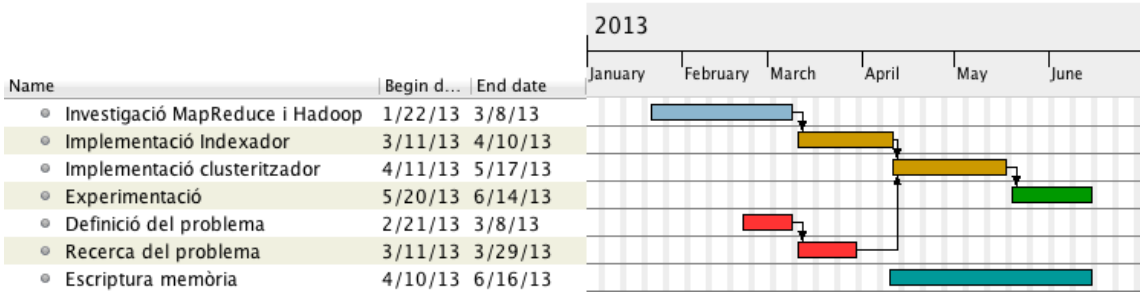

Figura 5.3: Diagrama de Gantt del projecte.

Per acabar la secció, s'explicarà en què consisteixen les diferente tasques abans citades al diagrama.

- **Investigació MapReduce i Hadoop.** Durant aquesta fase inicial s'ha realitzat la recerca del paradigma MapReduce i de Hadoop, la familiarització amb el framework i la provatura de diversos exemples d'execució.

- **Definició del problema.** Inicialment, la idea del projecte va ser atacar alguna problemàtica que impliqués grans quantitats de dades amb Hadoop. Durant aquesta fase s'ha investigat sobre quin seria el problema a tractar i la definició del mateix.

- **Recerca del problema.** Durant aquesta fase s'ha fet la recerca de la forma de abordar el problema, combinant els conceptes de machine learning amb MapReduce.

- **Implementació Indexador.** Un cop investigada la manera d'abordar una part del problema i havent adquirit certa familiaritat amb Hadoop, es procedeix a implementar la primera de les aplicacions, necessària per fer la segona.

- **Implementació Clusteritzador.** Acabada la implementació de l'indexador es passa a programar el programa principal. En aquest punt ja s'assoleix la maduresa suficient per començar a escriure la documentació final.

- **Experimentació.** Una de les fases final correspon a fer experimentació i testeig amb les aplicacions desenvolupades per extreure resultats i conclusions.

- **Escriptura de la memòria. Documentació.** Durant les fases més posteriors, en paral·lel, es va escrivint la documentació final, que és el present document.

Capítol 6

Conclusions i treball futur

En aquest darrer capítol s'examinaran les fites plantejades a l'inici i s'oferirà una conclusió englobant del projecte sencer. A banda es realitzaran propostes de treball futur destinades tant a millorar el software com a construir-ne de nou a partir d'aquest.

L'apunt final de la memòria correspon a aquest capítol. Repassant els objectius específics plantejats a la introducció, s'has assolit les següents fites:

- Coneixement del paradigma de programació distribuïda MapReduce i del framework Apache Hadoop.
- S'ha implementat una aplicació d'indexació de documents web basada en Apache Hadoop.
- S'ha implementat una aplicació de clústering amb l'algorisme K-means basada en Apache Hadoop.

Addicionalment, s'ha realitzat una experimentació amb les aplicacions desenvolupades tant en matèria de qualitat com de rendiment amb diferents configuracions i s'han presentat els resultats en aquesta memòria, així com un anàlisi d'aquests resultats.

Com a treball futur, sorgeixen diverses idees per millorar o enriquir el software desenvolupat. Tot seguit es presenten aquestes propostes.

Les operacions de reducció són les principals enemigues de la paral·lelització de codi. Funcions com la de calcular el màxim o el mínim requereixen fer servir totes les dades o, en el cas que ocupa en aquest projecte, per construir l'índex de documents, cal tenir tots els documents amb tots els termes i totes les paraules clau. Si la quantitat de dades comença a ser d'una mida important l'eficiència dels mòduls que construeixen l'índex comporten un punt crític a l'hora de programar-los, estalviant memòria i, a més, temps degut a que s'executa tot en una sola màquina.

L'alternativa a construir l'índex *on demand* és disposar d'un diccionar de termes per tal d'acceptar només aquells termes que apareguin en aquest diccionari. Amb això el que s'aconseguiria seria reduir la càrrega dels reducers i augmentar la dels mappers, però com que els mappers formen part de la fase paral·lela de l'indexador és interessant contemplar aquesta implementació amb diccionari.

Anant més enllà, la construcció d'aquest diccionari podria fer-se a partir d'un clústering previ de documents i extraient els termes més definitoris amb tècniques com les *tag clouds* o núvols de termes. Aquests núvols es conformen amb els termes més rellevants d'un conjunt de documents. A banda de construir un diccionari, amb aquests termes seria possible etiquetar els clústers.

En el capítol de proves s'ha vist com documents com les pàgines d'error 404 aporten soroll a les dades. Una millora en aquest sentit a l'indexador seria filtrar-les per no incloure-les al conjunt final de documents indexats.

Pel que fa a l'algorisme K-means, la inicialització aleatòria dels centroids podria millorar-se tenint en compte mètriques de les dades sobre les quals es farà el clústering. D'aquesta manera es podrien evitar clústers buits i una convergència més ràpida.

De cara a utilitzar aquest software dins d'un altre àmbit que no sigui el de l'agrupació de documents web, una proposta molt interessant és la de definir un framework de clústering no només de documents web sinó per tot tipus d'ítems amb qualsevol tipus de característiques. Algunes parts del codi ja s'han programat pensant en aquesta idea. Tot i així cal desacoplar més el codi.

Apunt final personal

La realització d'un projecte final comporta un procés gran i costós. Un treball que queda reflectit en aquesta memòria. En l'àmbit universitari ha suposat per mi la manera d'acabar de polir la formació impartida al llarg de la carrera, consolidant matèries impartides i recorrint a les eines del desenvolupament de software.

A nivell professional, la temàtica del treball propicia una bona signatura final pel meu currículum. Una temàtica que està a l'alça i que es consolida i s'obre cada cop més espai amb el temps.

Per acabar, a títol personal, aquest treball representa un resum de tot el que ha suposat realitzar la carrera d'Enginyeria Informàtica. Un reflex dels coneixements, la maduresa i la consolidació d'unes metodologies de treball obtingudes a partir de l'esforç i la constància.

Bibliografia

[1] Apache Software Foundation. Cluster setup. http://hadoop.apache.org/docs/stable/cluster_setup.html, 2013.

[2] Apache Software Foundation. Mapreduce tutorial. http://hadoop.apache.org/docs/stable/mapred_tutorial.html, 2013.

[3] Apache Software Foundation. Single node setup. http://hadoop.apache.org/docs/stable/single_node_setup.html, 2013.

[4] Crescenzi, V., Merialdo, P., and Missier, P. Clustering web pages based on their structure. Tech. rep., Università Roma Tre and University of Manchester, 2004.

[5] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. Tech. rep., Google, Inc., 2004.

[6] He, G. Authoritative k-means for clustering of web search results. Tech. rep., Norwegian University of Science and Technology, 2010.

[7] Internet Archive. Heritrix user manual and kristinn sigurđsson and michael stack and igor ranitovic. http://crawler.archive.org/articles/user_manual/.

[8] Twitter. 200 millones de tweets por día. http://blog.es.twitter.com/2011/06/200-millones-de-tweets-por-dia.html, 2011.

[9] Wikibooks. K-means. http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/K-Means, 2009.

[10] Wikipedia. Tag cloud. http://en.wikipedia.org/wiki/Tag_cloud, 2011.

[11] Wikipedia. Big data. http://en.wikipedia.org/wiki/Big_data, 2012.

[12] Zamir, O., and Etzioni, O. Web document clustering: A feasibility demonstraiton. Tech. rep., University of Washington, 1998.

Apèndix A

Instal·lació de Hadoop

Aquest apèndix explica a grans trets els passos necessaris per la instal·lació de Hadoop al sistema operatiu Linux.

Prerequisits

- **Java JDK** 1.6 o superior.

Descàrrega

Apache ofereix una sèrie de distribucions i versions de Hadoop. La recomanació és sempre fer servir la versió estable en el moment en que es faci la descàrrega (`http://hadoop.apache.org/releases.html#Download`). La distribució que s'utilitza en aquesta guía correspon amb el fitxer `hadoop-x.y.z.tar.gz`.

Es descomprimirà el fitxer en el directori on es desitgi realitzar la instal·lació (e.g. `/opt/hadoop`).

```
$ tar -xzvf hadoop-x.y.z.tar.gz
```

Configuració de la xarxa

Abans de configurar Hadoop cal que els hostnames de les màquines que conformin el cluster figurin al fitxer `/etc/hosts`.

Un cop configurat el fitxer de hostnames, és recomanable crear un usuari Hadoop dedicat per qüestions de seguretat.

```
$ sudo addgroup hadoop
$ sudo adduser --ingroup hadoop hduser
```

Hadoop fa servir SSH per fer la comunicació pel clúster. Cal generar, doncs, les claus SSH i autoritzar-les.

```
$ su - hduser
$ ssh-keygen -t rsa -P ""
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Per autoritzar les claus sense password a les màquines caldrà executar:

```
hduser@master:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@slave
```

On master i slave són els hostnames de les màquines.

Fitxers de configuració de Hadoop

La distribució de Hadoop disposa d'uns fitxers de configuració editables per l'usuari. Aquí es presenta la configuració bàsica per tal de fer funcional la distribució en clúster.

conf/hadoop-env.sh

Configura diverses variables d'entorn. Les que s'han de canviar es llisten a continuació:

```
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
export JAVA_HOME=/path/to/java/home
# export JAVA_HOME=‘/usr/libexec/java_home‘ per OS X 10.7
```

conf/core-site.xml (totes les màquines)

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://master:54310</value>
</property>
```

conf/mapred-site.xml (totes les màquines)

```
<property>
  <name>mapred.job.tracker</name>
  <value>master:54311</value>
</property>

<property>
  <name>mapred.child.java.opts</name>
  <value>-Xmx864m</value>
</property>

<property>
  <name>mapred.tasktracker.map.tasks.maximum</name>
  <value>2</value>
</property>

<property>
  <name>mapred.tasktracker.reduce.tasks.maximum</name>
  <value>1</value>
</property>
```

conf/hdfs-site.xml (totes les màquines)

```
<property>
  <name>dfs.name.dir</name>
  <value>/opt/hadoop/hdfs/name</value>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>/opt/hadoop/hdfs/data</value>
</property>

<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
```

Els directoris es poden canviar a gust de l'usuari.

conf/masters (màquina màster)

Figura la llista de hostnames de màquines que actuen com a màster. Normalment només n'hi ha una per clúster.

```
master
```

conf/slaves (màquina màster)

Figura la llista de hostnames de màquines que actuen com a esclaus.

```
slave1
slave2
...
```

Arrencada

Abans de comprobar que tot funciona correctament cal formatar l'HDFS. Per fer-ho es formata el NameNode amb la següent comanda:

```
$ bin/hadoop namenode -format
```

Per arrencar i aturar Hadoop es fan servir les següents comandes:

```
$ bin/start-all.sh
$ bin/stop-all.sh
```

Amb tot això, Hadoop estarà llest per a l'execució.

Apèndix B

Manual d'ús

Aquest apèndix conforma un petit manual d'ús de les aplicacions desenvolupades a partir d'una distribució disponible per descàrrega des del repositori del projecte. Aquest manual val pels sistemes operatius Linux i Mac OS X.

Prerequisits

La distribució que s'ofereix necessita tenir instal·lats els següents mòduls a la màquina màster del clúster on el vagi a executar:

- **Java JDK** 1.6 o superior.
- **Make**
- **Apache Ant**
- **Apache Hadoop** 1.0.4 o superior. A l'apèndix A s'explica com instal·lar-ho.

En aquest manual es presuposa que tot aquest software està instal·lat correctament i funcionant.

Preparació

La descàrrega de la distribució es conforma d'un fitxer comprimit. S'haurà de moure al directori on es vulgui instal·lar el programa i descomprimir-ho allà:

```
$ tar -xzvf HHCluster-1.0.tar.gz
```

```
HADOOP_HOME   = /opt/hadoop-1.0.4
HADOOP_CORE   = $(HADOOP_HOME)/hadoop-core-1.0.4.jar
HADOOP_EXE    = $(HADOOP_HOME)/bin/hadoop
HADOOP_FORMAT = $(HADOOP_EXE) namenode -format

HDFS_DIR = /opt/HDFS

JAR_DIR = dist/HHCluster-1.0.jar

INDEXER_CLASS     = edu.ub.ahstfg.indexer.Indexer
CLUSTERIZER_CLASS = edu.ub.ahstfg.clusterizer.Clusterizer

INPUT_FILES = ~/Desktop/testWordCount/idx_input/*.arc.gz
```

Figura B.1: Directoris al Makefile per canviar.

Un cop descomprimit el fitxer, l'estructura és com segueix:

```
HHCluster-1.0/
├── lib/
├── src/
├── build.xml
└── Makefile
```

El directori `lib` conté les diferents llibreries externes utilitzades, el directori `src`, els fitxers de codi font del programa i el fitxer `Makefile` actúa com a controlador. `build.xml` és el fitxer de Ant per compilar, però no cal executar-ho explícitament.

Abans de res, caldrà editar el fixer `Makefile` per establir dos directoris que canvien a preferència de l'usuari (figura B.1).

El directori `HADOOP_HOME` és el directori d'instal·lació de Hadoop. El directori `HDFS_DIR` indica el directori on es guarda l'HDFS, configurat als nodes, mentre que el directori `INPUT_FILES` es correspon amb la ruta on són els fitxers arc.gz d'entrada. Tot i que per compilar el codi no cal fer aquests canvis, és recomanable fer-los.

Compilació

Per compilar el codi caldrà situar-se en el directori `HHCluster-1.0` i executar:

```
HHCluster-1.0$ make
```

En aquest punt s'executarà Ant per compilar el codi, generar la documentació Javadoc i generar el fitxer Jar per l'execució.

Control de Hadoop i execució

El fitxer Makefile s'ha escrit amb vistes de poder controlar-ho tot des del directori d'instal·lació. Disposa de targets pel control de Hadoop així com per l'execució dels programes.

A continuació s'ofereix una taula (B.1) amb les comandes i l'explicació de què fan.

Comanda	**Explicació**
$ make h_start	Engega la infraestructura de Hadoop.
$ make h_stop	Atura la infraestructura de Hadoop.
$ make h_format	Formata (esborra) el namenode i, per tant, l'HDFS. Cal que Hadoop estigui aturat.
$ make h_fullstop	Atura Hadoop i formata l'HDFS.
$ make idx_set_input	Copia a l'HDFS les entrades arc.gz per ser processades per l'indexador.
$ make indexing	Engega el procés d'indexació. Cal que les entrades hagin estat copiades a l'HDFS.
$ make clustering ARGS="<K> <keywords_weight> <term_weight> <num_machines> <max_iterations>"	Engega el procés de clústering. Necessita els arguments especificats.
$ make reset	Reinicia Hadoop formatant l'HDFS.

Taula B.1: Comandes del Makefile de control.

Els arguments de la comanda de clústering amb la seva explicació són a la taula B.2.

Visualització dels resultats

A l'introduir les comandes d'execució, a la consola de la màquina màster s'anirà visualitzant el procés d'execució. Tanmateix per veure els resultats es necessita

Argument	Explicació
K	Nombre de clústers desitjats.
keywords_weight	Pes de les paraules clau per computar distàncies.
term_weight	Pes dels termes per computar distàncies.
num_machines	Nombre de màquines que conformen el clúster.
max_iterations	Màxim nombre d'iteracions que realitzarà Hadoop.

Taula B.2: Comandes del Makefile de control.

accedir a l'HDFS.

Hadoop disposa d'unes eines de monitorització per controlar l'execució i l'accés als logs de forma més *user-friendly*. Aquestes eines són conformades per diverses aplicacions web, les quals poden ser vistes des del mateix navegador.

Principalment n'hi han dos portals per accedir a informació útil. El primer és el portal del JobTracker, que monitoritza l'execució dels jobs. Per defecte corre sobre el port 50030 (http://localhost:50030). L'altre monitoritza el NameNode i permet un accés a l'HDFS. Corre sobre el port 50070 (http://localhost:50070).

Per veure els resultats del clústering, doncs, al monitor del Namenode s'accedirà a la opció "Browse de filesystem" i s'accedirà a la ruta `/user/<nom_usuari>/kmeans_output/part-XXXXX`. Aquests són els fitxers de sortida del K-means i com es veu a la figura B.2 hi figura cada URL amb l'etiqueta del clúster al qual pertany.

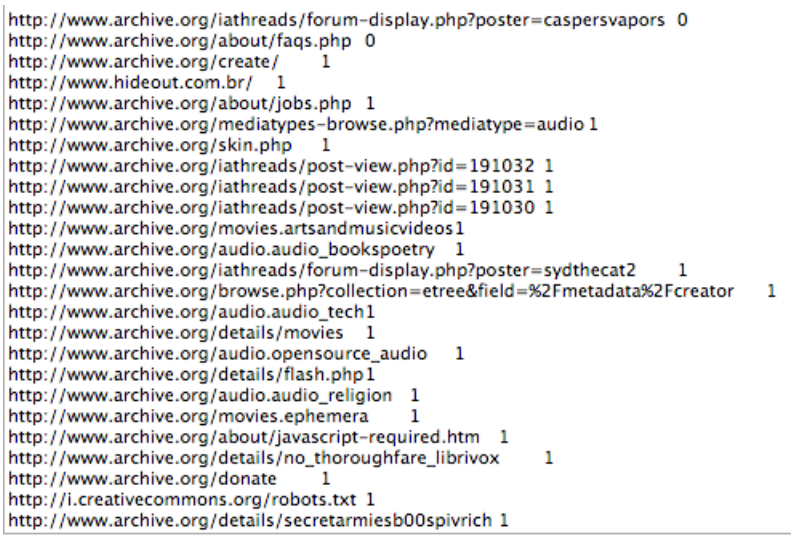

Figura B.2: Monitors d'activitat. A dalt el JobTracker, a baix el NameNode.