Universitat Autònoma
de Barcelona

HADOOP: PROCESSAMENT DISTRIBUÏT DE GRAN VOLUM DE DADES EN EL NÚVOL D'APACHE

Memòria del projecte d'Enginyeria Tècnica en
Informàtica de Sistemes realitzat per

ADRIÀ ARCARONS ARQUÉ

Direcció

DR. JAUME PUJOL CAPDEVILA
PROF. BERNAT GASTÓN BRASÓ

Escola d'Enginyeria

Sabadell, juny de 2013

El sotasignat, **Jaume Pujol Capdevila**,

professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria ha estat realitzat sota la seva direcció per

Adrià Arcarons Arqué

I per a que consti firma la present.

Signat: Jaume Pujol Capdevila

Sabadell, Juny de 2013

El sotasignat, **Bernat Gastón Brasó**,

professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria ha estat realitzat sota la seva direcció per

Adrià Arcarons Arqué

I per a que consti firma la present.

Signat: Bernat Gastón Brasó

Sabadell, Juny de 2013

FULL DE RESUM – PROJECTE FI DE CARRERA DE L'ESCOLA D'ENGINYERIA

Títol del projecte:

Hadoop: processament distribuït de gran volum de dades en el núvol d'Apache

Autor: Adrià Arcarons Arqué | **Data:** 25/06/2013

Tutors: Dr. Jaume Pujol Capdevila; Prof. Bernat Gastón Brasó

Titulació: Enginyeria Tècnica en Informàtica de Sistemes

Paraules clau

Català: Sistemes d'Emmagatzematge Distribuït, Hadoop, HDFS RAID, Codis Correctors d'Errors

Castellano: Sistemas de Almacenamiento Distribuido, Hadoop, HDFS RAID, Códigos Correctores de Errores

English: Distributed Storage Systems, Hadoop, Hadoop, HDFS RAID, Erasure Codes

ABSTRACT

Català

Avui en dia es genera un volum increïble de dades de diferents tipus i que provenen de multitud d'orígens. Els sistemes d'emmagatzematge i processament distribuït són els elements tecnològics que fan possible capturar aquest allau de dades i permeten donar-ne un valor a través d'anàlisis diversos. Hadoop, que integra un sistema d'emmagatzematge i processament distribuïts, s'ha convertit en l'estàndard *de-facto* per a aplicacions que necessiten una gran capacitat d'emmagatzematge, inclús de l'ordre de desenes de PBs. En aquest treball farem un estudi de Hadoop, analitzarem l'eficiència del seu sistema de durabilitat i en proposarem una alternativa.

Castellano

Hoy en día se genera un volumen increíble de datos de diferentes tipos y que proceden de multitud de orígenes. Los sistemas de almacenamiento y procesado distribuidos son los elementos tecnológicos que hacen posible capturar esta avalancha de datos y permiten extraer un valor de ellos a través de diferentes tipos de análisis. Hadoop, que

integra un sistema de almacenaje y procesado distribuidos, se ha convertido en el estándar de-facto para aplicaciones que necesitan una gran capacidad de almacenaje, incluso del orden de decenas de PBs. En el presente trabajo realizaremos un estudio de Hadoop, analizaremos la eficiencia de su sistema de durabilidad, y propondremos una alternativa.

English

Nowadays, the amount of data generated, which comes from various sources, is overwhelming. Distributed storage systems are the technological solution that make possible to capture this avalanche of data and to obtain a value from it. Hadoop, which offers a distributed storage and processing systems, has become the *de-facto* standard for applications that seek for a big storage capacity, even in the order of tens of PBs. In the present work, we'll study Hadoop, we'll analyze its durability system's efficiency and we will propose an alternative to it.

Agraïments

En primer lloc, vull manifestar el meu agraïment al Dr. Jordi Pons, pel seu consell i ajuda a l'hora d'encaminar el present projecte. Ell em va ajudar a donar les primeres passes per aquest intens camí.

En segon lloc, vull agrair sincerament al Dr. Jaume Pujol tota l'atenció i ajuda que m'ha proporcionat durant tot el projecte. La seva proximitat mostrada en les converses que hem mantingut al seu despatx ha contribuït a estimular el meu interès per la matèria en un clima molt agradable. La seva determinació en els moments de dubte o d'estancament m'ha transmès forces per perseverar. Sense ell aquest projecte no hauria estat possible.

En tercer lloc, vull agrair al Professor Bernat Gastón el temps que ha dedicat a orientar-me i a transmetre'm molts coneixements teòrics que han estat necessaris per desenvolupar aquest treball. El seu coneixement sobre la matèria ha fet que algunes reunions amb ell s'hagin convertit en autèntiques classes.

En quart lloc, vull agrair profundament a la meva mare, la Dra. Maite Arqué, tots els moments en els que hem pogut intercanviar emocions i opinions. Les converses disteses que hem mantingut en els moments més quotidians han estat molt inspiradores. Les seves aportacions en forma d'idees i d'ànims han estat d'un gran valor i han deixat una clara empremta en aquest treball.

Tot i que, a vegades, amb un posat distret, sembla que no t'escolti, les teves paraules calen fons en mi. Ets mestre i consellera, sàvia de les emocions, i una excel·lent mare. Moltes gràcies.

En cinquè lloc, vull agrair a Melcior Arcarons, el meu pare, les converses mantingudes durant els nostres dinars al Silvestre i el recolzament que m'ha donat quan l'he necessitat. Malgrat que últimament no hem pogut trobar-nos gaire, ell és una persona a qui admiro, i el seu criteri i les seves opinions són sempre valuoses.

Per últim, vull agrair a totes aquelles persones properes a mi tot el suport i la empenta que m'han donat durant aquest camí.

Taula de continguts

Índex de figures

Índex de taules

Capítol 1: Introducció

1.1 Presentació

El present treball és el resultat del projecte final de carrera necessari per obtenir la diplomatura en Enginyeria Tècnica en Informàtica de Sistemes. Els estudis corresponents en aquesta diplomatura s'han realitzat a la Facultat d'Enginyeria de la Universitat Autònoma de Barcelona, al campus universitari de Sabadell.

L'objectiu principal d'aquestes recerques tracta d'iniciar al futur diplomat en competències pròpies d'una recerca científica: identificació d'algun tema rellevant per la ciència informàtica, plantejament d'hipòtesis de treball, cerca i utilització de fonts d'informació diverses, obtenció de conclusions sobre tot el procés desenvolupat i comunicació de resultats de formes diverses: text escrit i exposició oral.

Durant el procés de decisió del tema de recerca, van sorgir dubtes sobre quin tema triar. M'interessaven els àmbits de les xarxes, l'arquitectura de sistemes d'alt rendiment i de processament paral·lel. Va ser llavors quan fent una prospecció de fonts vaig identificar la transcendència del camp dels sistemes d'emmagatzematge distribuït. Em va semblar un tema molt interessant per la seva actualitat i per la seva repercussió tècnica i social, i perquè relacionava transversalment diversos àmbits del meu interès. Per tant, la meva línia de recerca tractaria sobre l'estudi sobre els diferents sistemes d'emmagatzematge distribuït.

En aquell punt va ser fonamental l'ajut del Dr. Jordi Pons. L'havia tingut com a professor en l'assignatura de xarxes i havia demostrat ser un excel·lent professional i comunicador i vaig posar-me en contacte amb ell per compartir els meus reptes. Vaig expressar-li el meu interès per una recerca sobre els sistemes d'emmagatzematge distribuïts, pensant en ell com a un bon tutor per dirigir-la. A partir d'aleshores el Dr. J. Pons em va suggerir que seria una millor opció posar-me en contacte amb el Dr. Jaume Pujol, del Departament d'Enginyeria de la Informació i de les Comunicacions a la UAB, que havia creat unes línies de recerca que s'assemblaven a la meva proposta.

A l'octubre de 2012, vaig tenir una entrevista personal amb el Dr. J. Pujol. En aquesta, vaig poder expressar-li la meva proposta, i ell va considerar que es podria integrar en una línia de recerca en la qual estava treballant el professor i doctorand Bernat Gastón, ja que hi tenia elements comuns.

A partir d'aquell moment vam establir que la present recerca portaria el títol de "Hadoop: processament distribuït de gran volum de dades en el núvol d'Apache". A continuació explicaré, amb brevetat, el per què aquest és un tema de rellevància i d'actualitat.

1.2 Justificació del tema de la recerca

L'any 2011, un estudi va estimar la quantitat de dades que es creen i es repliquen durant un any en 1.8 Zettabytes[1] (1.8 x 10^21 bytes), quantitat equivalent a més de 250GB per cada ésser humà a la terra. Un estudi de la mateixa companyia[2] va mes enllà i estima que al 2020 el volum de dades creades i replicades anualment serà d'uns 40 Zettabytes. És a dir, que al 2020, la producció de dades serà 22 vegades més gran que les produïdes el 2011.

Aquestes dades que formen l'Univers Digital tenen orígens molt diversos, per exemple: registres de transaccions bancàries, material provinent de sistemes de videovigilància, col·lisions subatòmiques registrades pel Gran Col·lisionador d'Hadrons al CERN[3], arxiu d'historials mèdics, i també informació generada a les xarxes socials, com vídeos emmagatzemats a YouTube, imatges pujades a Facebook, o text escrit al Twitter. En un futur proper, molts dels objectes que utilitzem quotidianament generaran dades: les neveres, els cotxes, les cases i, per què no, també els nostres cossos[4].

El creixement exponencial d'aquestes dades i la necessitat d'emmagatzemar-les i de processar-les de forma que se'n pugui extreure algun valor planteja reptes desafiants als equips d'Enginyeria, que han hagut de dissenyar sistemes d'emmagatzematge d'alta capacitat que siguin capaços d'escalar d'una forma lineal. Aquests sistemes s'anomenen sistemes d'emmagatzematge i processament distribuït, i són els elements tecnològics que fan possible capturar l'allau de dades que existeix avui en dia i permeten donar un valor a la informació oferint la capacitat de realitzar anàlisis diversos en un temps raonable.

Els sistemes d'emmagatzematge distribuït, o *Distributed Storage System* (DSS), permeten emmagatzemar grans volums de dades digitals, distribuïdes entre un conjunt de nodes interconnectats per una xarxa. Un DSS ha de ser altament escalable, ha de garantir la disponibilitat i la durabilitat de les dades, i ha d'oferir un alt rendiment. Aquestes característiques els fan d'utilitat per a un tipus d'aplicació que demanda: (1) una gran capacitat d'emmagatzematge, arribant a l'ordre de desenes de PB i, (2) la possibilitat de fer un processament paral·lel d'aquestes dades. Els DSS són utilitzats en àmbits d'aplicació molt diversos com el de la investigació científica, les finances, la medicina, el comerç electrònic o les xarxes socials.

Tot i que hi ha diversos DSS, en aquest projecte ens centrarem en Hadoop. Per què?. Perquè Hadoop és un software *open-source*, integra un sistema d'emmagatzematge i processament distribuïts, i s'ha convertit en l'estàndard *de-facto* per a aplicacions que necessiten una gran capacitat d'emmagatzematge. Entre els usuaris de Hadoop hi ha grans empreses com IBM, Yahoo i, especialment, la coneguda xarxa social Facebook, que opera el clúster més gran de Hadoop que és coneix, amb una capacitat d'emmagatzematge que supera els 100PB, i on diàriament s'incorporen 500TB de dades noves

Darrera dels sistemes d'emmagatzematge distribuït i de Hadoop s'amaguen molts

[1] Veure [36].
[2] Veure [35].
[3] Veure http://en.wikipedia.org/wiki/Worldwide_LHC_Computing_Grid.
[4] Veure [37].

interrogants apassionants. En aquest projecte intentarem, humilment, aportar significat a alguns d'aquests interrogants, com: quin és el funcionament de Hadoop? Quines característiques tècniques té? De quina manera es garanteix la durabilitat de les dades a Hadoop? És eficient la replicació com a sistema de durabilitat? Quins costos té? Existeixen alguna alternativa millor? És possible arribar a implementar-la a Hadoop?

Tots aquests aspectes fins ara esmentats ens permeten afirmar la importància d'investigar sobre el present i el futur de Hadoop per ser un sistema que està revolucionant el paradigma de l'emmagatzematge i el processament del gran volum de dades que existeix avui en dia.

1.3 Estructura del treball

Aquest treball està organitzat en 9 capítols. A continuació repassarem quin és el tema de cada capítol i què hi conté:

- **Capítol 1: Introducció.** En aquest capítol farem una breu presentació d'aquest treball, exposarem les motivacions que justifiquen l'elecció del tema, donem una estructura del treball i finalment diem quina ha estat la metodologia per fer la recerca.
- **Capítol 2: Planificació del projecte.** En aquest capítol direm quins són els objectius del projecte i presentarem el pla de viabilitat i la planificació temporal.
- **Capítol 3: Estudi de Hadoop.** En aquest capítol farem un estudi de Hadoop i dels seus components: l'HDFS i el MapReduce. Analitzarem l'arquitectura de Hadoop i acabarem posant un exemple d'una aplicació que podria funcionar sobre Hadoop.
- **Capítol 4: Tècniques basades en codis aplicades a l'àmbit dels sistemes d'emmagatzematge distribuït.** En aquest capítol analitzarem l'eficiència del sistema de durabilitat de Hadoop, introduirem la tècnica dels codis correctors d'errors i en presentarem les seves bases teòriques. A continuació identificarem un problema que presenten els codis correctors d'errors –el problema de la reparació– i finalment explicarem una nova família de codis correctors d'errors enfocats a disminuir-lo, els codis QCFMSR.
- **Capítol 5: Anàlisi del sistema.** Analitzarem l'HDFS RAID, una implementació dels codis correctors d'errors a Hadoop feta per Facebook. Això establir les bases del disseny del nostre desenvolupament: la implementació dels codis QCFMSR a l'HDFS RAID.
- **Capítol 6: Disseny de l'aplicació.** Recapitularem sobre el propòsit de la implementació a partir de tot el que hem estudiat fins al moment. Explicarem el disseny de les classes a implementar i presentarem un diagrama de classes.
- **Capítol 7: Implementació i proves.** Explicarem les classes que s'ha implementat i les funcions que contenen. Detallarem el conjunt de proves que hem dissenyat i finalment donarem els resultats de les proves.
- **Capítol 8: Conclusions.** En aquest capítol analitzarem el grau d'assoliment dels objectius plantejats en el pla de viabilitat, comentarem alguns fets succeïts durant el desenvolupament del projecte, proposarem unes línies de futur i acabarem amb una valoració personal
- **Capítol 9: Bibliografia.** Presentem el material bibliogràfic utilitzat en l'elaboració d'aquest projecte.

1.4 Metodologia de la recerca

Per tal de dur a terme la recerca es va realitzar un Pla de viabilitat, que més tard precisarem, i, que ens va ajudar a ordenar el procés de treball, encara que com és habitual, aquest pla s'ha vist modificat en alguns moments per condicionants diversos.

La metodologia té en compte una part de treball dirigit per part dels professors/tutor i, una part de treball autònom. Quant al treball dirigit s'han realitzat unes 15 reunions de seguiment realitzades al principi, durant i al final de la recerca. Aquestes reunions han servit per a acotar els objectius i la dimensió de la recerca, per plantejar i resoldre dubtes, i reorganitzar de nou algunes de les tasques. Val a dir que en aquesta recerca hi participa un doctorand i que en certes ocasions, les reunions les hem tingudes els tres participants –professor tutor-doctorand-jo com a estudiant- o, solament professor tutor-estudiant, o solament docotrand-estudiant.

Quant al treball autònom, destaquem un nombre d'hores important per a comprendre i fixar el Pla de viabilitat i, seguidament, un bon nombre d'hores per desenvolupar les tasques pròpies de la recerca quant a la selecció de fonts d'informació, la comprensió dels fonaments teòrics, la representació i anàlisi de la informació, i, la síntesi final que es materialitza en dos sistemes d'avaluació, un és el present treball escrit, i, un altre, és l'exposició oral davant d'un tribunal d'avaluació.

Capítol 2: Planificació del projecte

2.1 Objectius del projecte

- Analitzar el funcionament de Hadoop.
- Aprofundir en el sistema d'emmagatzematge distribuït HDFS.
- Proposar i implementar una alternativa al sistema de durabilitat.
- Cercar secció del codi font que conté l'algorisme de les replicacions
- Implementar les replicacions mitjançant codis correctors d'errors
- Implementar les replicacions mitjançant codis regeneratius.
- Fer *benchmark* de les dues implementacions.

2.2 Viabilitat

A continuació presentem quin és el temps previst per la consecució dels objectius definits anteriorment:

	Hores/dia	2,5

Objectiu	**Dies previstos**	**Hores previstes**
El·laborar l'estudi de viabilitat	3	7,5
Analitzar el funcionament de Hadoop.	7	17,5
Aprofundir en el sistema d'emmagatzematge distribuït HDFS.	5	12,5
Proposar i implementar una alternativa al sistema de durabilitat.	10	25
Cercar secció del codi font que conté l'algorisme de les replicacions.	30	75
Implementar les replicacions mitjançant codis correctors d'errors.	32	80
Implementar les replicacions mitjançant codis regeneratius.	22	55
Fer benchmark de les dues implementacions.	12	30
Escriure la memòria	16	40
TOTAL	**137**	**342,5**

Taula 2.1: Dedicació prevista als objectius del projecte

La data d'inici del projecte és el 03/12/2012 i la data prevista de finalització és el 25/6/2013 amb una dedicació diària de 2,5h i una dedicació total prevista de 342,5h. A més, cal sumar-hi el temps dedicat a les reunions de seguiment amb els tutors del projecte. S'estima una reunió d'una hora cada dues setmanes entre el període d'inici i el període de finalització del projecte. Són, aproximadament, 30 setmanes entre l'inici i la finalització i, per tant, 15 reunions de control. Així, cal sumar 15h a la dedicació prevista.

En total, cal dedicar al projecte 357,5 hores, que està d'acord amb la dedicació prevista per part de l'alumne.

Per tal d'estimar el cost econòmic del projecte, prenem com a referència la "Tabla salarial y de plus convenio 2009" situada a l'annex 3 del "XVI CONVENIO COLECTIVO ESTATAL DE EMPRESAS DE CONSULTORÍA Y ESTUDIOS DE MERCADOS Y DE LA OPINIÓN PÚBLICA" [1] publicat al BOE el 04/04/2009. Suposem que ens escau una categoria d'Analista de Sistemes i que, per tant, tenim un salari anual brut de 21.969,50. L'estimació dels dies laborables de l'any 2012 va ser de 251 dies laborables[5]. Calculem el preu per hora: 21.969,50/(251*8) = 10,94€/hora. A continuació presentem una taula amb la valoració econòmica del projecte:

€/Hora	10,94
Hores/dia	2,5

Objectiu	Dies previstos	Hores previstes	Cost de l'objectiu
El·laborar l'estudi de viabilitat	3	7,5	82,05 €
Analitzar el funcionament de Hadoop.	7	17,5	191,45 €
Aprofundir en el sistema d'emmagatzematge distribuït HDFS.	5	12,5	136,75 €
Proposar i implementar una alternativa al sistema de durabilitat.	10	25	273,50 €
Cercar secció del codi font que conté l'algorisme de les replicacions.	30	75	820,50 €
Implementar les replicacions mitjançant codis correctors d'errors.	32	80	875,20 €
Implementar les replicacions mitjançant codis regeneratius.	22	55	601,70 €
Fer benchmark de les dues implementacions.	12	30	328,20 €
Escriure la memòria	16	40	437,60 €
TOTAL	**137**	**342,5**	**3.746,95 €**

Taula 2.2: Cost econòmic del projecte

Quant a la viabilitat legal, els drets de Hadoop estan regulats per la llicència *Apache License*, que és una llicència de software lliure. La llicència Apache obliga a conservar l'avís de copyright però no requereix la redistribució del codi font quan se n'elaboren versions modificades. Com qualsevol altre llicència de software lliure, la llicència Apache permet a l'usuari del software la llibertat d'utilitzar-lo per a qualsevol propòsit, de distribuir-lo i de modificar-lo. La llicència de l'HDFS RAID també està regulada per la *Apache License*. D'aquest fet podem concloure que no existeixen conflictes legals que es puguin derivar de la utilització i modificació de Hadoop i de l'HDFS RAID. A més, al tractar-se de software lliure, no es deriven costos econòmics en termes d'adquisició de drets o llicències.

Quant a la viabilitat tècnica, per a poder desenvolupar el projecte i complir amb els objectius definits, cal disposar dels següents elements:

- Software:
  - Apache Hadoop: Com hem dit, Hadoop és un software lliure i, per tant, no suposa un cost econòmic.
  - HDFS RAID: és un software lliure i no suposa un cost econòmic.
  - Eclipse: és un entorn gràfic de desenvolupament Java que no té cost

[5] Segons la web http://www.dias-laborables.es/

econòmic.
    - Microsoft Office Word: per escriure i maquetar la memòria i l'estudi de viabilitat. L'alumne ja disposa d'aquest software.
    - Microsoft Visio: per la elaboració d'esquemes i diagrames. L'alumne ja disposa d'aquest software.
    - Microsoft Project: per a elaborar la planificació temporal i el diagrama de Gantt. L'alumne ja disposa d'aquest software.
- Hardware:
    - Els recursos de Hardware per operar un entorn de proves d'alt rendiment amb Hadoop són de disposar de 4 màquines amb unes especificacions similars a:
        - 1 x 1Gb/s Ethernet, 2 GB de RAM, 1xCPU Dual (Xeon preferiblement), 100 GB HD SATA (NO RAID, SAS/SCSI NO NECESSARI)
        - Ubuntu Server 12.04.1 LTS
        - Java 6 de SUN
        - SSH activat
        - Firewall: tot el tràfic permès entre les màquines del clúster.
        - No s'aconsella la virtualització en entorns de Hadoop
    - També caldrà disposar d'una màquina on fer el desenvolupament i la redacció de la memòria: per a aquesta finalitat, l'alumne disposa d'un ordinador de sobretaula i d'un ordinador portàtil.

Els requisits de software són satisfets ja que no impliquen cap cost econòmic. Els requisits de hardware són proporcionats per la Universitat Autònoma de Barcelona, la qual disposa d'un laboratori per a projectistes amb unes màquines de característiques similars. Per tant, a nivell tècnic, el projecte és viable.

A nivell econòmic, el fet que no sigui necessari realitzar cap mena d'inversió per l'adquisició de software o hardware fa viable el projecte.

2.3 Planificació temporal

A la figura 2.1 es mostra un diagrama de Gantt amb la planificació prevista pel projecte:

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Elaboració de l'estudi de viabilitat**	3 días	lun 03/12/12	mié 05/12/12
2	**Analitzar el funcionament de Hadoop.**	7 días	mié 05/12/12	vie 14/12/12
3	**Aprofundir en el sistema d'emmagatzematge distribuït HDFS.**	5 días	vie 14/12/12	vie 21/12/12
4	**Proposar i implementar una alternativa al sistema de durabilitat.**	10 días	vie 21/12/12	vie 04/01/13
5	**Cercar secció del codi font que conté l'algorisme de les replicacions**	**30 días**	**vie 04/01/13**	**vie 15/02/13**
6	Decidir sobre quina versio de Hadoop treballarem	2 días	vie 04/01/13	mar 08/01/13
7	Localitzar i descarregar	1 día	mar 08/01/13	mié 09/01/13
8	Explorar i analitzar com estan estructurats els diferents fitxers de codi	10 días	mié 09/01/13	mié 23/01/13
9	Localitzar quina o quines seccions del codi estan involucrades en el procés de replicació de blocs	17 días	mié 23/01/13	vie 15/02/13
10	**Disseny**	**10 días**	**vie 15/02/13**	**vie 01/03/13**
11	Comprendre els principis bàsics de l'arquitectura de Hadoop:HDFS, rols dels nodes,...	3 días	vie 15/02/13	mié 20/02/13
12	Fer un breu repàs a la base teòrica que sustenta la modificació a fer	2 días	mié 20/02/13	vie 22/02/13
13	Llegir sobre codis correctors d'errors i codis regeneratius	2 días	vie 22/02/13	mar 26/02/13
14	Com traduïr aquesta solució a un algorisme? Localitzar algorismes d'exemple que utilitzin codis correctors d'errors i codis regeneratius.	3 días	mar 26/02/13	vie 01/03/13
15	Implementar les replicacions mitjançant codis correctors d'errors.	32 días?	vie 01/03/13	mar 16/04/13
16	Implementar les replicacions mitjançant codis regeneratius.	22 días?	mar 16/04/13	jue 16/05/13
17	**Fer benchmark de les dues implementacions**	**12 días**	**jue 16/05/13**	**lun 03/06/13**
18	Determinar quins són els aspectes a evaluar: I/O de disc, RAM, CPU, I/O de xarxa, nº de bits transmesos a la xarxa durant la fallada d'un sol node, nº de nodes	2 días	jue 16/05/13	lun 20/05/13
19	Trobar formes de medir	7 días	lun 20/05/13	mié 29/05/13
20	Fer benchmarks	3 días	mié 29/05/13	lun 03/06/13
21	Escriure la memòria	16 días	lun 03/06/13	mar 25/06/13

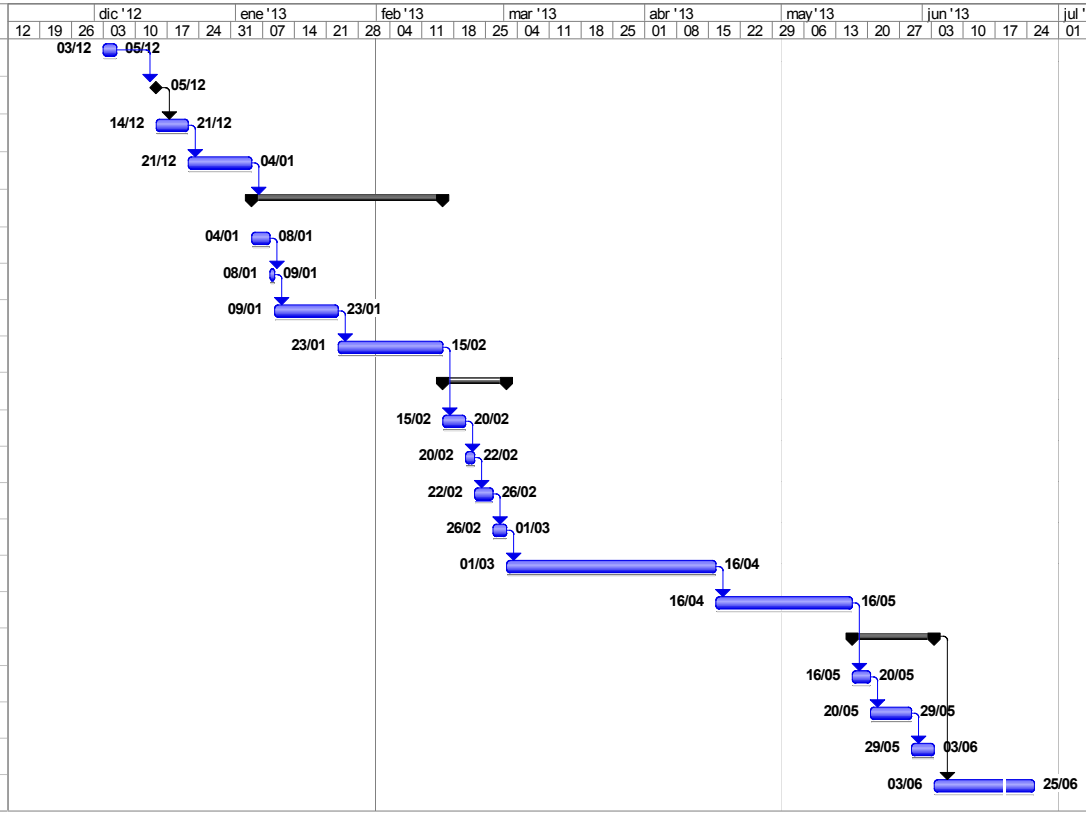

Figura 2.1: Diagrama de Gantt. Elaboració propia.

Capítol 3: Estudi de Hadoop

3.1 Què és, quines característiques té, qui el programa, origen i evolució, utilitats i usuaris del Hadoop

Hadoop és un software *open-source* útil per aplicacions distribuïdes que tenen un flux de dades de gran volum. Ofereix a les aplicacions un sistema d'emmagatzematge de dades d'alta capacitat i la possibilitat de processar grans conjunts de dades de forma distribuïda, sobre un clúster de màquines, fent servir models de programació senzills. Està dissenyat per escalar des d'un sol node fins a milers de nodes, oferint cada un d'ells la capacitat de còmput local i d'emmagatzematge de dades.

En comptes de recolzar-se en hardware car i propietari, Hadoop permet fer processament paral·lel i distribuït sobre servidors *low-cost*, amb especificacions modestes, que poden, al mateix temps, emmagatzemar i processar les dades.

Hadoop està programat en Java i és un projecte liderat per l'Apache Foundation, sota la llicència Apache v2. És un software molt popular i rep contribucions de diverses empreses i programadors d'arreu del món.

El nom "Hadoop" no correspon a un acrònim. És un nom inventat. Doug Cutting, el creador del projecte Hadoop, va explicar amb aquestes paraules com va sorgir:

> *The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term.* [2]

De fet, en consonància amb l'origen del terme, el logotip de Hadoop és un elefant groc.

Figura 3.1: Logotip de Hadoop. Font: http://www.christian-ariza.net/wp-content/uploads/2010/10/hadoop+elephant_rgb.png.

Inicialment, Hadoop va ser inspirat per diversos whitepapers publicats per Google, on descrivien el seu enfocament per gestionar l'allau de dades que tenien. Actualment, s'ha convertit en l'estàndard *de facto* per emmagatzemar, processar i analitzar centenars de terabytes, o inclús petabytes de dades.

La organització del codi de Hadoop és bastant modular i, es poden diferenciar **dos components cabdals**: l'**HDFS** i el **MapReduce**. L'HDFS és el sistema de fitxers que utilitza Hadoop per emmagatzemar les dades. El MapReduce és un paradigma de programació que permet dissenyar aplicacions per fer còmputs sobre les dades

emmagatzemades a l'HDFS. Analitzarem aquests components en profunditat més endavant.

Hadoop ofereix flexibilitat en l'emmagatzematge, ja que no imposa un esquema per les dades [3]. Així, pot admetre qualsevol tipus de dades: estructurades, no estructurades, fotografies, logs, mails, pel·lícules, etc., sigui quin sigui el seu format original. Aquest enfocament *schema-less* no limita la capacitat d'anàlisi de les dades sinó tot al contrari, permet treure'n gran partit, oferint eines per fer anàlisis complexos i exhaustius, més que qualsevol altre sistema de bases de dades pugui oferir.

Hadoop és utilitzat per empreses d'arreu del món en funcionalitats essencials i crítiques del negoci. Algunes d'aquestes són Facebook[6], Twitter, Yahoo, eBay, IBM, Last.fm, LinkedIn o Spotify.

Els àmbits beneficiaris d'utilitat del Hadoop són molt diversos. En destaquem alguns d'importants:

- **Ciència**: diagnòstic mèdic per imatge, recopilació de dades de sensors, seqüenciació del genoma humà, dades climatològiques, informació provinent de satèl·lits.

- **Empresa i gerència**: dades de vendes, *Business Intelligence*, estudi d'hàbits de compra, elaboració d'estudis de mercat, anàlisi de dades de xarxes socials.

- **Dades de sistemes**: sector de la banca, obtenció d'estadístiques a partir de logs, analítiques Web, cerques basades en patrons, elaboració d'índexs.

Cal dir que Hadoop no és una solució idònia per a qualsevol aplicació. Tot i els seus punts forts, Hadoop no està dissenyat per aplicacions que requereixen temps de resposta immediats, sinó més aviat per processos *batch* [2]. A més, l'entorn MapReduce és més complicat que les eines tradicionals de bases de dades com SQL [4].

3.2 Anàlisi del MapReduce

El MapReduce és un model de programació que permet fer processament de gran quantitat de dades. Permet escriure aplicacions que processen conjunts de dades de múltiples terabytes, en paral·lel, dins clústers de milers de nodes, d'una forma confiable i tolerant a fallades.

Hadoop pot executar programes MapReduce escrits en diferents llenguatges: Java, Ruby, Python, C++,... [2]. El més important és que els programes que utilitzen el model MapReduce són inherentment paral·lels. Gràcies a això, qualsevol organització que disposi d'un nombre suficient de nodes obtindrà una capacitat enorme de processament de dades.

El programador només cal que implementi dues funcions, `map` i `reduce`, i queda alliberat d'ocupar-se de totes les particularitats que fan funcionar el sistema de processament paral·lel, ja que l'entorn MapReduce s'encarrega automàticament de l'assignació de recursos, la coordinació de les tasques,... [5].

[6] Veure detalls de l'arquitectura de Hadoop dins de Facebook a [38].

Les dades que processa el MapReduce estan estructurades en parells $< clau, valor >$. La funció *map* realitza un processat en paral·lel sobre un conjunt de parells d'entrada i dona com a sortida una llista intermèdia de parells. Després, una funció fa particions unint la sortida de cada instància de *map*. Hi haurà una partició per cada tasca *reduce* que s'executi[7], a més, cada registre pertanyent a la mateixa $clau$ s'agruparà en la mateixa partició. Cada tasca *reduce* processa una partició i dona com a sortida un parell $< clau, valor >$ que formarà part del resultat final [6]. A continuació presentem un senzill exemple per facilitar la comprensió al lector.

Exemple d'una feina MapReduce[8]: El comptador de paraules

Tenim emmagatzemat a Hadoop un fitxer que conté una llista de les ventes que ha fet una fruiteria durant un dia. Per cada unitat de fruita venuda, s'escriu al fitxer el nom de la fruita. Per tant, si venem 2 kiwis, escriurem al fitxer "kiwi kiwi".

Per una qüestió d'estocs, volem saber, per cada fruita, quin és el número d'unitats venudes. El nostre programador implementa una feina MapReduce que compta cada aparició del nom de la fruita dins el fitxer. Cada aparició significa una unitat venuda.

La tasca seguirà els passos que il·lustra el següent esquema:

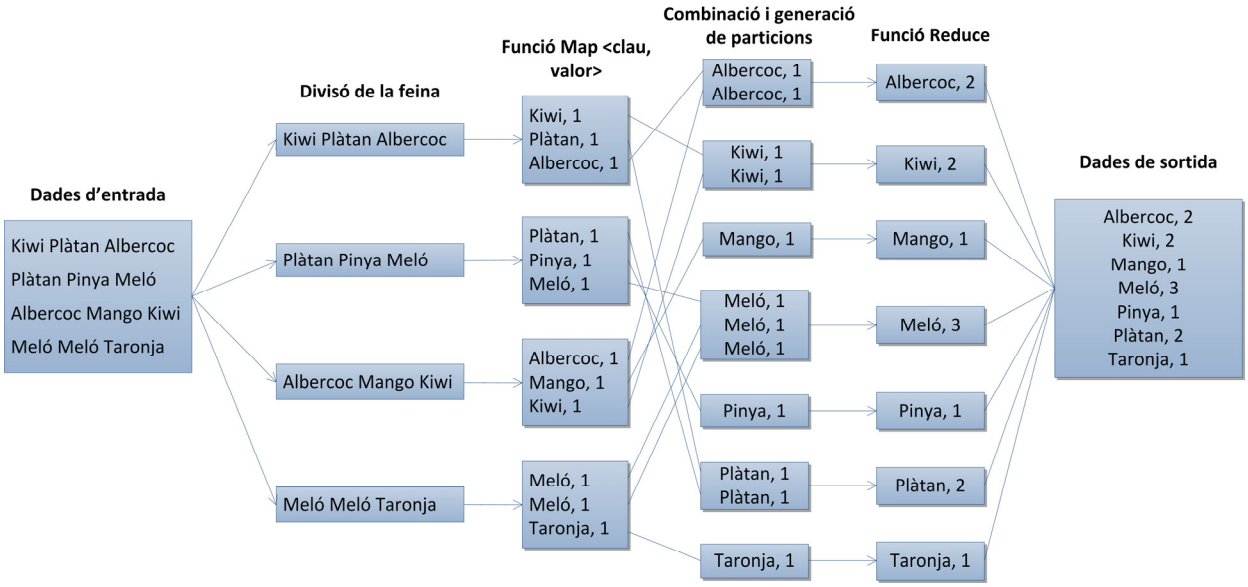

Figura 3.2: Diagrama de feina MapReduce. Elaboració propia. Adaptació de [7].

A Hadoop hi ha dos tipus de nodes que intervenen en el procés d'execució de les feines MapReduce. Hi ha un sol node que pren el rol de JobTracker, i un cert nombre de nodes que prenen el rol de TaskTrackers. El JobTracker és responsable de coordinar totes les feines que s'estan executant en el clúster, i assigna les tasques a fer a cada un dels TaskTrackers. Els TaskTrackers executen les tasques i envien periòdicament informes de progrés al JobTracker, el qual porta un registre del progrés total de cada feina.

[7] Hadoop permet especificar el nombre de tasques `reduce` que s'executaran per a cada feina. [2]
[8] A [39], el lector por trobar un altre exemple de feina MapReduce.

Quan Hadoop rep l'encàrrec de fer una feina MapReduce, divideix el conjunt de dades a processar en petites porcions independents. Els TaskTrackers s'encarreguen de processar cada una de les porcions en les que ha estat dividida la feina. Aquest processament es fa d'una forma completament paral·lela.

El sistema combina les sortides de les funcions Map i genera particions, que es converteixen en l'entrada de la funció Reduce. El JobTracker s'ocupa de programar les tasques, monitoritzar-les i de tornar-les a executar en cas que hi hagi alguna fallada.

Com s'ha dit, una feina MapReduce consta de dos etapes ben diferenciades:

- Primera etapa: ***Map***. El client envia al JobTracker l'encàrrec de fer una feina. Les dades es divideixen en fragments independents. Hadoop crea una tasca Map per cada un dels fragments. El JobTracker assigna una certa quantitat de TaskTrackers perquè executin les tasques Map. El programador ha definit prèviament la funció `map`, que ha estat transmesa a Hadoop pel client. Un cop processades les tasques Map, les sortides intermèdies que s'han generat són emmagatzemades temporalment al disc dur dels TaskTrackers.

- Segona etapa: ***Reduce***. El JobTracker assigna un o més TaskTrackers per executar la funció *Reduce*. Aquests TaskTrackers recopilaran les sortides intermèdies de la funció `map` i executaran `reduce` per obtenir la feina sol·licitada.

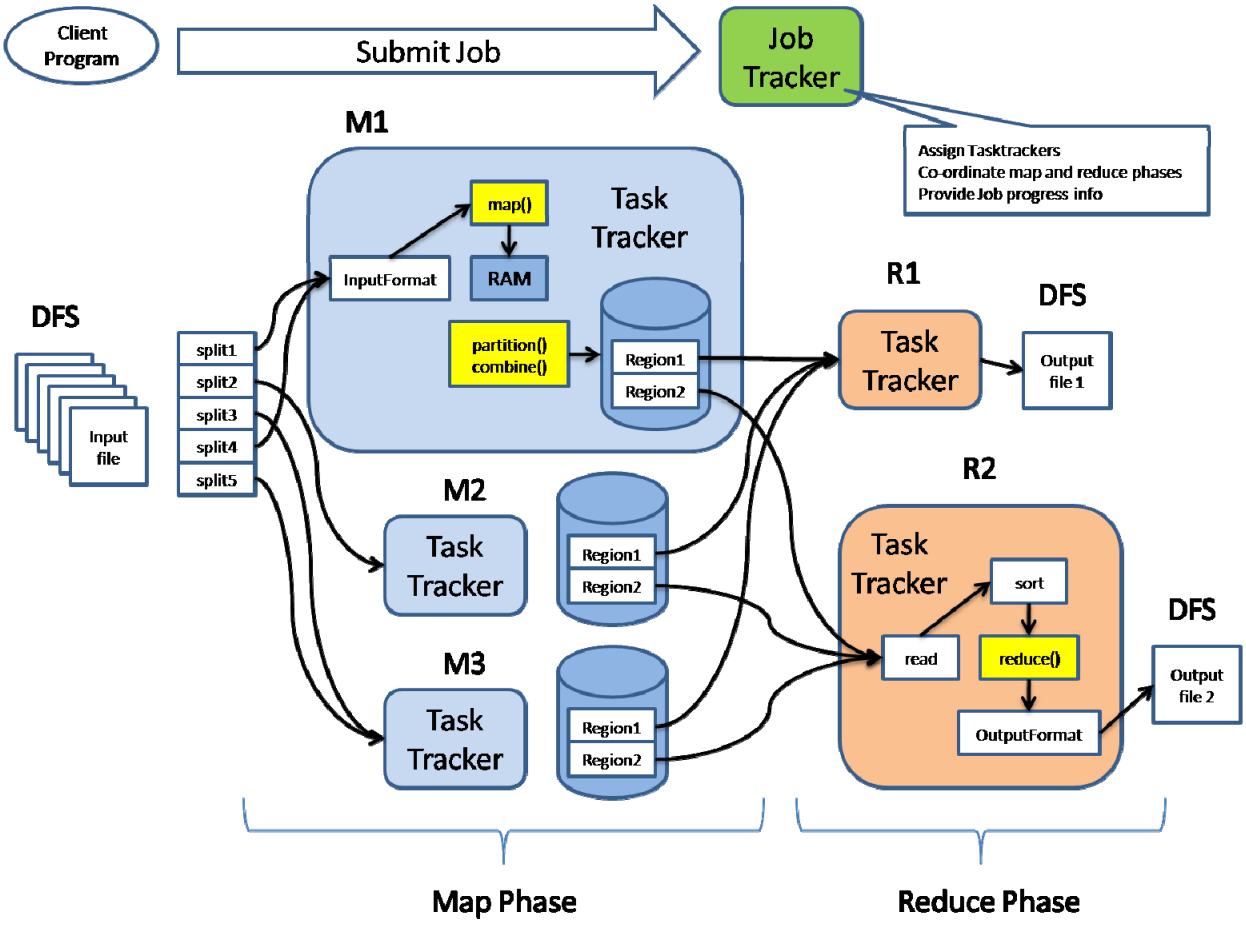

Figura 3.3: Funcionament d'una feina MapReduce. Autor: Richy Ho. Font: http://architects.dzone.com/articles/how-hadoop-mapreduce-works

A la figura 3.3 s'observa el funcionament d'una feina MapReduce. En una feina MapReduce no intervenen únicament el JobTracker i el TaskTracker, sinó que també hi ha el *client*, que sol·licita la feina al JobTracker, i el sistema de fitxers distribuït (DFS), que és utilitzat per compartir els fitxers de la feina entre tots els actors.

Normalment els nodes que fan el còmput també emmagatzemen les dades. Això vol dir que el MapReduce i l'HDFS s'executen en el mateix conjunt de nodes. Hadoop intenta executar les tasques Map en els nodes on es troben les dades, o el més a prop possible dins la topologia de xarxa. Això optimitza l'accés a les dades, ja que com són locals, permet d'evitar o disminuir les transmissions de xarxa. Les tasques Reduce no poden beneficiar-se de la localitat de les dades, ja que les dades d'input de la funció Reduce són els output de les funcions de Map.

3.3 Anàlisi de l'HDFS

El Hadoop Distributed File System (HDFS) és el sistema de fitxers que utilitza Hadoop. És un sistema de fitxers distribuït[9], dissenyat per emmagatzemar fitxers de mida molt gran. Pot ser ubicat sobre *commodity-hardware*, és molt tolerant a fallades i proporciona un accés a les dades en *streaming* d'alt rendiment.

3.3.1 Blocs

La unitat bàsica d'emmagatzematge dins de l'HDFS és el bloc. El bloc de l'HDFS és una unitat molt més gran que el bloc dels sistemes de fitxers tradicionals. Per defecte, tenen una mida de 64MB.

Els fitxers de l'HDFS es parteixen en porcions de la mida d'un bloc. Aquestes porcions s'emmagatzemen com a unitats independents. Tots els blocs d'un fitxer, excepte l'últim, són de la mateixa mida.

3.3.2 Commodity-hardware[10]

Hadoop no requereix un hardware car i extra-redundant per funcionar. De fet, té un funcionament òptim sobre clústers de *commodity-hardware* en els quals la probabilitat de la fallada de nodes en tot el clúster és alta (sobretot en clústers grans).

Això és possible gràcies al seu disseny, ja que Hadoop incorpora mecanismes de redundància propis, que fan que no depengui de la confiabilitat del Hardware. Això ho analitzem en profunditat més endavant, a la secció de "Replicació de dades" (a la pàgina núm 17).

3.3.3 Latència vs. Rendiment

Hadoop està dissenyat amb la idea que la forma més eficient de processar les dades és amb un patró *write-once, read-many-times*. Sovint, un conjunt de dades és emmagatzemat a Hadoop des de l'origen, i després s'hi realitzen diversos anàlisis que

[9] Un sistema de fitxers distribuït és aquell que gestiona l'emmagatzematge a través d'un conjunt de màquines interconnectades per una xarxa.

[10] *Commodity-hardware* fa referència a equips amb unes especificacions de hardware "modestes". És maquinari low-cost que no disposa de sistemes de redundància.

impliquen la lectura del conjunt de dades sencer. Per tant, el temps dedicat a llegir tot el conjunt és més important que no pas la latència en llegir la primera dada (*latency* vs *throughput*).

Aquest model té alguns avantatges: simplifica el problema de la concurrència i la coherència de les dades, i permet un gran rendiment.

Cal dir que una aplicació que necessiti un accés a les dades amb una latència baixa (de l'ordre de desenes de milisegons) no funcionarà bé amb l'HDFS [2].

3.3.4 Aspectes destacats de l'HDFS

- Gran tolerància a fallades i alta disponibilitat: detecta les fallades i les repara de forma ràpida i automàtica.
- Sistema de fitxers distribuït: permet connectar nodes dins un clúster en els quals es distribueixen els fitxers de dades. Hadoop fa transparents les particularitats del sistema distribuït de cara a l'aplicació client.
- Gran rendiment de lectura: les lectures es fan en paral·lel des de múltiples discos.
- Adequat per aplicacions amb un gran volum de dades. La mida dels fitxers emmagatzemats a Hadoop pot ser de l'ordre de GB o PB.
- És escalable i fàcil d'expandir: només cal incorporar més màquines al clúster.
- Permet accedir a les dades en forma d'*streaming*. El tipus d'accés a les dades està orientat al processament *batch*, i no tant a la interactivitat amb l'usuari.
- Funciona bé sobre un clúster heterogeni de *commodity-hardware*.
- Cada fitxer s'emmagatzema com una seqüència de blocs
- Confiable: manté múltiples còpies de les dades
- L'HDFS assegura la detecció de corrupció en les dades mitjançant el codi CRC-32. Per defecte, calcula un *checksum* de 32 bits per cada 512 bytes de dades.

3.4 Arquitectura de Hadoop

3.4.1 NameNode i DataNodes

L'HDFS té una arquitectura mestre-esclau. Un clúster d'HDFS consta d'un node mestre, que s'anomena NameNode. El NameNode gestiona l'espai de noms del sistema de fitxers i regula l'accés als fitxers per part dels clients. També consta d'un cert nombre de DataNodes, que utilitzen el disc local per emmagatzemar les dades.

L'HDFS té un espai de noms i permet que les dades del client siguin emmagatzemades en fitxers. Internament, els fitxers es divideixen en un o més blocs, i aquests blocs s'emmagatzemen en un conjunt de DataNodes. El NameNode executa operacions del sistema de fitxer com obrir, tancar i reanomenar fitxers i directoris. També determina el mapeig de blocs amb DataNodes. És a dir, té la informació de a

quins nodes s'ha emmagatzemat cada bloc.

Els DataNodes són els responsables d'atendre les peticions de lectura i escriptura dels clients del sistema de fitxers. També porten a terme la creació, eliminació i replicació dels blocs, sota la direcció del NameNode.

L'existència d'un sol NameNode en un clúster simplifica notablement l'arquitectura del sistema. El NameNode és l'àrbitre del sistema i on s'hi emmagatzema totes les metadades de l'HDFS. De fet, Hadoop està dissenyat de tal forma que les dades del client mai transcorren a través del NameNode

Tot i així, el NameNode és un *single point of failure*. La fallada d'un NameNode és un problema crític del que Hadoop no es pot recuperar automàticament i requereix intervenció manual. És per això que el maquinari destinat a executar el NameNode és l'únic que requereix sistemes de redundància a nivell de Hardware com RAID o PSU redundades.

L'espai de noms de l'HDFS té una organització tradicional jeràrquica. L'usuari o l'aplicació pot crear directoris i emmagatzemar fitxers dins aquests directoris. La operativa permesa és l'habitual en la majoria de fitxers: es poden crear i eliminar fitxers, reanomenar-los, moure fitxers d'un directori a un altre,...

3.4.2 Lectures i escriptures

Per donar una idea de quins fluxos de dades es produeixen en les operacions de lectura i escriptura, presentem dos esquemes que ens donaran una visió general de la seqüència de passos que es porta a terme.

La interacció entre l'aplicació i l'HDFS es realitza amb les funcions d'una llibreria nativa que proporciona Hadoop.

Operació de lectura

1. L'aplicació obre el fitxer que vol llegir fent una crida a la funció `OPEN` de la instància DistributedFileSystem.

2. Aquest objecte es comunica amb el NameNode, utilitzant RPC, per determinar la ubicació dels primers blocs del fitxer. Per cada bloc, el NameNode retorna l'adreça del DataNode que en conté una còpia. A més, els DataNodes s'ordenen d'acord amb la seva proximitat al client (això es calcula amb la informació de la topologia de xarxa).

3. El DistributedFileSystem retorna un objecte al client perquè pugui fer una lectura en stream de les dades. L'aplicació obté les dades amb la funció `READ` del FSDataInputStream

4. El DFSInputStream es connecta amb el node més proper que conté el primer bloc del fitxer. El DataNode envia les dades al client en forma d'stream, el qual crida la funció de `READ` repetidament.

5. Quan ja s'ha arribat al final del bloc, el DFSInputStream tanca la connexió amb el DataNode, i busca el millor DataNode per servir el proper bloc. Això es fa de

forma transparent a l'aplicació, que l'únic que veu es un stream de dades continu. Aquest pas es repeteix fins a acabar de llegir tots els blocs del fitxer.

6. Quan l'aplicació ha acabat la lectura, crida la funció `CLOSE` per tancar la connexió.

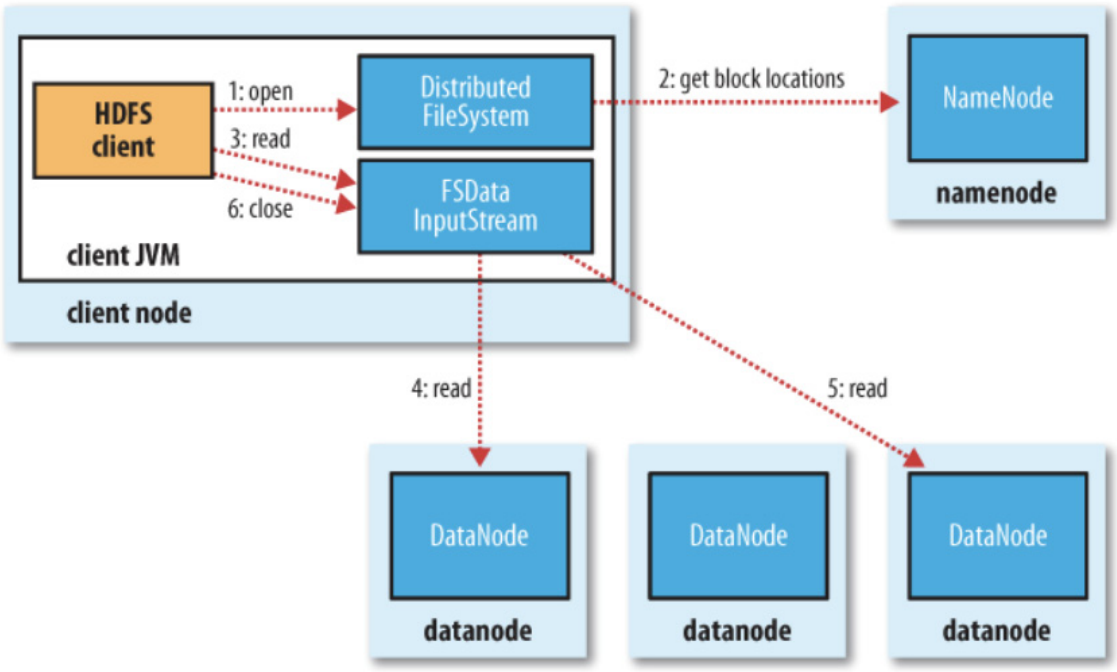

Figura 3.4: Funcionament d'una operació de lectura a l'HDFS. Font: [2].

Operació d'escriptura

1. L'aplicació crea el fitxer fent una crida a `CREATE` de l'objecte DistributedFileSystem

2. El DistributedFileSystem fa una crida al NameNode per RPC per crear el nou fitxer dins l'espai de noms del sistema de fitxers.

3. A mida que l'aplicació va escrivint les dades, el DFSOutputStream les va partint en paquets, i els col·loca dins una cua interna. Aquesta cua es va buidant per un component anomenat "Data Streamer". Aquest component s'encarrega de preguntar al NameNode a quins DataNodes cal emmagatzemar les repliques. El NameNode retorna una llista de 3 nodes (assumint un factor de replicació de 3) que forma una espècie de cadena.

4. El "Data Streamer" envia els paquets al primer DataNode de la cadena, el qual guarda el paquet i el reenvia al segon DataNode de la cadena. El segon DataNode fa el mateix, emmagatzema el paquet, i el reenvia a l'últim DataNode de la cadena.

5. Quan el DFSOutputStream rep les confirmacions (ACK) per part de tots els DataNodes, els marca com a enviats correctament. Després es tanca la cadena que comunica els DataNodes, i si hi ha qualsevol paquet del que no s'ha rebut ack, es torna a posar a la cua d'enviament de dades.

6. Quan la aplicació ha acabat d'escriure les dades, crida a la funció CLOSE per tancar l'stream de dades.

7. Quan s'ha aconseguit enviar tots els paquets, inclòs els que havien fallat (no s'havia rebut l'ACK), el client contacta amb el NameNode per indicar que la transmissió s'ha completat.

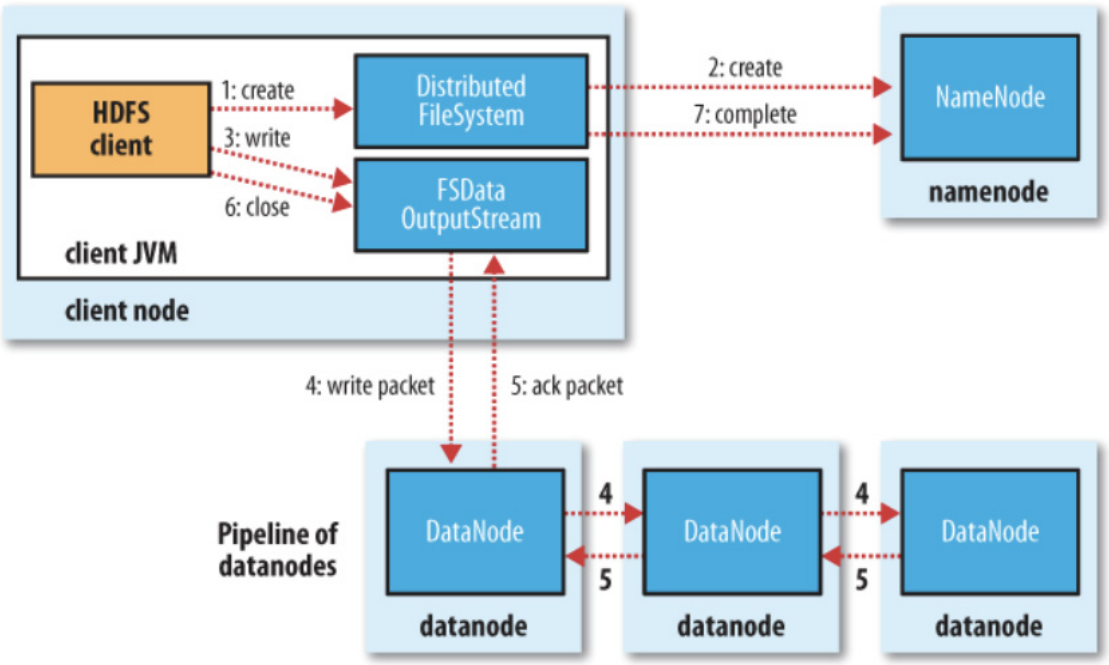

Figura 3.5: Funcionament d'una operació d'escriptura. Font: [2].

La petició d'un client de crear un fitxer no arriba immediatament al NameNode. De fet, inicialment el client HDFS fa un caché de les dades del fitxer dins un fitxer temporal guardat en local. Les escriptures de l'aplicació es redirigeixen temporalment a aquest fitxer temporal. Quan el fitxer acumula la mida d'un bloc HDFS, el client contacta amb el NameNode. El NameNode indica al client amb quin DataNode ha de contactar, i és llavors quan el client comença la transmissió de dades.

Aquest enfocament tenint en compte el tipus d'aplicacions que fan servir l'HDFS. Com s'ha dit, aquestes aplicacions necessiten un accés d'escriptura en *streaming*. Si el client fa escriptures a un fitxer remot directament, sense cap mena de *buffering* local, la velocitat i la congestió de la xarxa tenen un impacte negatiu en el rendiment.

3.4.3 Replicació de dades

En clústers de mida gran, amb centenars o milers de nodes, les fallades de Hardware esdevenen un problema molt freqüent. El número de components susceptibles a fallades en aquest tipus d'entorns és molt elevat i, en conseqüència, sovint es perd la disponibilitat d'alguna instància de l'HDFS. Per tant, la detecció i recuperació de fallades és un objectiu clau de Hadoop.

Per aconseguir aquesta fiabilitat, l'HDFS utilitza la replicació. La replicació és una tècnica que s'utilitza en sistemes informàtics d'emmagatzematge de dades amb l'objectiu de proporcionar tolerància a fallades. Consisteix en emmagatzemar còpies de

la mateixa informació en suports d'emmagatzematge diferents. El factor de replicació -és a dir, quantes vegades es redunda la informació- determina quin és el màxim nombre de fallades que podem tolerar sense perdre les dades.

Hadoop permet parametritzar el factor de replicació de forma individual per cada fitxer, per satisfer necessitats de redundància específiques. Per defecte, Hadoop aplica per defecte un factor de replicació de 3 a tots els fitxers que emmagatzema. En un sistema que tingui un factor de replicació de 3, donat el cas d'una eventual fallada a un dels suports, no hem perdut les dades, ja que seguim tenint 2 còpies de la informació.

El NameNode pren totes les decisions relacionades amb la replicació de blocs. Cada DataNode envia periòdicament al NameNode una senyal de que segueix "viu" (*heartbeat*) i una llista dels blocs que conté.

La fallada d'un enllaç de xarxa pot ocasionar la pèrdua de connectivitat a un conjunt de DataNodes. El NameNode és capaç de detectar aquesta situació per la absència dels *heartbeats*. En aquest cas, el NameNode marca com a "morts" els DataNodes que no han donat senyals de vida recentment, i no els reenvia cap petició d'E/S. La desaparició de DataNodes pot fer que el factor de replicació d'alguns blocs estigui per sota dels nivells fixats. En aquest cas, el NameNode ordenarà la replicació del bloc des de les ubicacions alternatives cap a una altra màquina en bon estat.

La necessitat de tornar a replicar un bloc pot donar-se degut a diverses raons: un DataNode que perd la disponibilitat, una rèplica amb corrupció de dades, un disc d'algun DataNode que ha patit una fallada, ...

En clústers grans les màquines es distribueixen en diversos *racks*, i inclús en diversos datacenters ubicats en països diferents. La comunicació entre dos nodes del mateix *rack* és més ràpida que la comunicació entre nodes ubicats en diferents *racks*. I la comunicació entre dos nodes ubicats dins el mateix datacenter és més ràpida que dos nodes que es troben a datacenters diferents. L'ample de banda és un be escàs, i això és un factor limitant.

La distribució òptima de les rèpliques és crítica pel rendiment de l'HDFS. El NameNode aplica una política de distribució de les rèpliques que té en compte la topologia de la xarxa. L'HDFS no és capaç de detectar la ubicació dels nodes, sinó que necessita que l'administrador del clúster informi de la topologia de la xarxa en els fitxers de configuració.

La política que l'HDFS aplica és: col·locar una rèplica en un node en el *rack* local, una altra rèplica en un node dins un *rack* remot, i la tercera rèplica en un node diferent dins el *rack* remot.

Aquesta política minimitza el tràfic d'escriptura entre *racks*, i donat que la probabilitat de que un *rack* falli és bastant menor que la probabilitat de fallada d'un node, es manté una bona tolerància a fallades.

3.4.4 La visió general

Per concloure el capítol on hem estudiat el funcionament de Hadoop, presentem un esquema que pretén donar una visió general sobre els elements importants de

l'arquitectura de Hadoop.

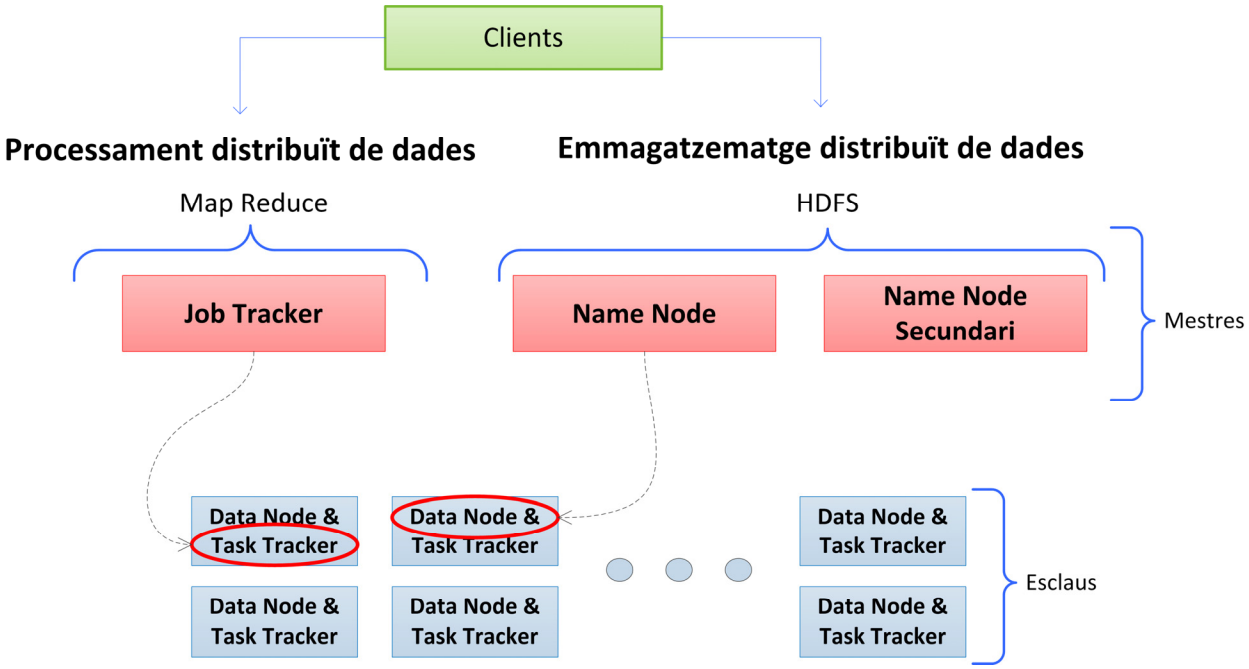

Figura 3.6: Funcionalitats i rols a Hadoop. Elaboració pròpia. Adaptació de [8].

Observem quines relacions s'estableixen entre les funcionalitats i els rols de Hadoop. L'HDFS ofereix a una aplicació "client" un sistema d'emmagatzematge de dades distribuït, i el MapReduce ofereix la capacitat de fer un processament distribuït d'aquestes dades.

L'arquitectura de Hadoop segueix un model jeràrquic mestre-esclau. A l'HDFS, un conjunt de DataNodes emmagatzemen les dades, i són governats pel NameNode, que pren totes les decisions sobre l'emmagatzematge i conté les metadades del sistema de fitxers. Donat que el NameNode és un punt singular de fallada, en clústers grans s'acostuma a tenir un NameNode secundari o de reserva". Al MapReduce, el JobTracker coordina les feines i governa un conjunt de TaskTrackers, que són els que processen les dades [8].

Habitualment, a cada node "esclau" estan presents les instàncies de DataNode i TaskTracker. És a dir, un mateix node ofereix emmagatzematge i processament.

Amb tot el que hem estudiat fins ara, podem enunciar els punts fort de Hadoop:

- És flexible, ja que no imposa un esquema fix per emmagatzemar o processar les dades, que poden ser de diferents tipus, estructurades o no.
- Té una escalabilitat lineal: per afegir capacitat al sistema només cal afegir més nodes.
- És robust i confiable, ja que és capaç de recuperar-se automàticament de les fallades.

- Té una bona relació de cost-efectivitat, ja que funciona bé sobre *commodity-hardware*.
- Ofereix un alt rendiment per operacions d'E/S.
- És capaç de fer anàlisis exhaustius de forma distribuïda sobre grans conjunt de dades, en forma de feines MapReduce. Gràcies al processament paral·lel de les dades, permet completar anàlisis executats sobre centenars de TB en temps raonables. Això maximitza el valor de la informació.

3.5 Exemple de l'arquitectura d'una aplicació afí amb Hadoop: Una biblioteca digital.

A continuació presentem un esquema (Figura 3.7) que hem elaborat per donar un exemple al lector d'un tipus d'aplicació on seria d'utilitat utilitzar Hadoop. En aquest exemple il·lustrem un sistema molt simplificat d'una biblioteca digital, on els llibres de paper es digitalitzen a través d'un escaneig, posteriorment són analitzats per una aplicació que construeix un índex fent un reconeixement de caràcters (OCR) i emmagatzema a Hadoop les pàgines escanejades en un sol document per cada llibre. L'aplicació d'usuari, accessible des de múltiples dispositius, permet fer cerques a l'índex per paraules clau, creant una tasca MapReduce per localitzar els llibres coincidents, i també visualitzar els llibres digitalitzats, emmagatzemats a l'HDFS.

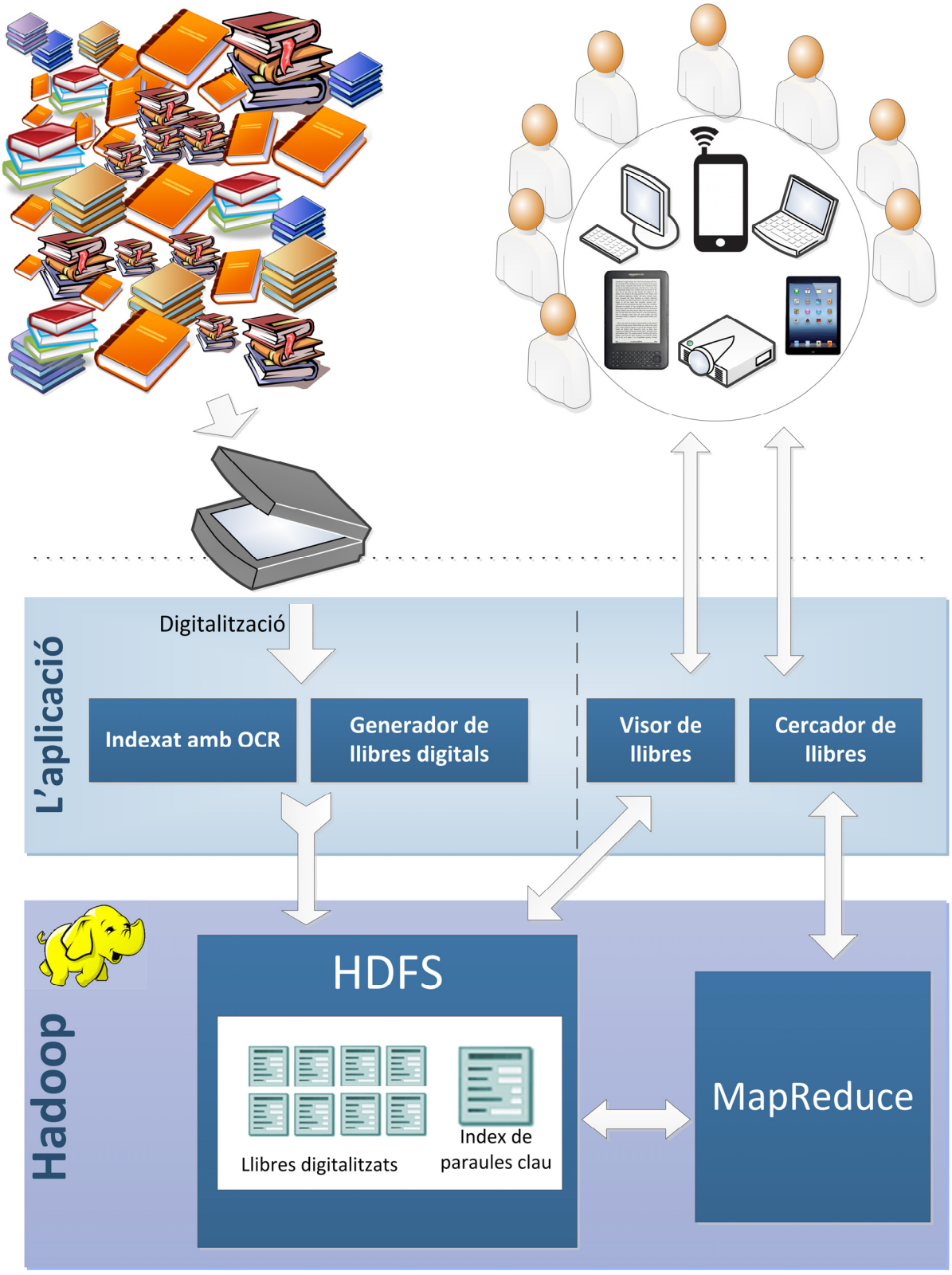

Figura 3.7: Una biblioteca digital. Exemple d'una aplicació afí a Hadoop. Elaboració pròpia.

Amb aquest esquema donem per acabat l'estudi de Hadoop. Al capítol següent identificarem un inconvenient que presenta el sistema de redundància de Hadoop i donarem unes bases teòriques que ens permetran proposar una solució.

Capítol 4: Tècniques basades en codis aplicades a l'àmbit dels sistemes d'emmagatzematge distribuït

En aquest capítol identificarem un problema relacionat amb el sistema de durabilitat de Hadoop: la replicació. Posteriorment introduirem i explicarem els fonaments teòrics sobre una tècnica matemàtica que ens permetrà proposar una alternativa a la replicació. Les fonts bibliogràfiques que s'han consultat per elaborar aquest capítol són: [9] [10] [11] [12] [13] [14] [15] [15] [16].

4.1 Identificació del problema

Els sistemes d'emmagatzematge distribuït sovint són elements molt crítics dins l'arquitectura de l'aplicació que els utilitza i han de satisfer requisits exigents com:

- Durabilitat de les dades en el temps (sovint els sistemes d'emmagatzematge distribuït arxiven dades que es vol preservar durant molt temps).
- Confiabilitat exigent (a vegades perdre un bit implica perdre tot el fitxer).
- Immutabilitat de les dades emmagatzemades.
- Alta disponibilitat (*Service-level agreements* exigents i, en general, requisits de l'aplicació).

Aquests requisits suposen un repte tècnic que cal afrontar en un context en el que els sistemes d'emmagatzematge distribuït creixen ràpidament en capacitat gràcies a la utilització de discs cada cop més grans i a la distribució de les dades per tota la xarxa. Quan es tracten sistemes d'una mida gran, la probabilitat de fallades de components també s'incrementa, per tant les tècniques per protegir les dades esdevenen essencials.

En aquest sentit, L'HDFS és molt robust. Cal tenir en compte, però, que un altre dels requisits importants per aquesta mena de sistemes és aconseguir una eficiència en el cost, que es mantingui o millori en el temps.

És aquí on l'HDFS presenta un inconvenient important, ja que cada bloc d'informació es replica en tres llocs. El cost de mantenir tres còpies de la informació pot semblar assumible tenint en compte la confiabilitat que proporciona i que el cost de l'emmagatzematge en disc durs ha baixat molt darrerament, i segueix baixant considerablement. Però la realitat és que, com veurem, en clústers de mida gran, la replicació és una solució molt ineficient en cost.

Per exemple, considerem que el nostre conjunt de dades ocupa, com a molt, 50TB. Farem el càlcul de quin cost en emmagatzematge de disc dur té un factor de replicació de 3, i un factor de replicació de 2. Considerem que el cost d'emmagatzematge d'1GB d'informació és de 0,1€[11].

[11] Aproximem el preu d'un disc dur en funció dels preus de mercat actual. Un disc dur SATA d'alt rendiment amb una capacitat d'emmagatzematge de 3TB té un cost entre 250-300€.

Factor de replicació de 3	**Factor de replicació de 2**
50TB * 3 copies = 150 TB = 150.000 GB	50TB * 2 copies = 100 TB = 100.000 GB
150.000 GB * 0,1€/GB = 15.000€	100.000 GB * 0,1€/GB = 10.000€
El cost de l'emmagatzematge és de 15k€	El cost de l'emmagatzematge és de 10k€

Taula 4.1: Comparativa del cost de la replicació de 50TB amb un factor de 2 i un factor de 3

L'estalvi en cost d'emmagatzematge que es produeix a l'emmagatzemar 2 còpies en comptes de 3, és de 5.000€. Aquesta estalvi no sembla gaire significatiu per a una empresa mitjana.

Ara, analitzem el cas d'una empresa que té un gran volum de dades i necessita emmagatzemar 30PB d'informació (30 * 10^6 GB)

Factor de replicació de 3:	**Factor de replicació de 2:**
30PB * 3 copies = 90 PB = 90.000.000 GB	30PB * 2 copies = 60PB = 60.000.000 GB
90.000.000 GB * 0,1€/GB = 9.000.000€	60.000.000 GB * 0,1€/GB = 6.000.000€
El cost de l'emmagatzematge és de 9 MEUR	El cost de l'emmagatzematge és de 6 MEUR

Taula 4.2: Comparativa del cost de la replicació de 30PB amb un factor de 2 i un factor de 3

En aquest cas, l'estalvi de cost d'emmagatzematge és de 3 milions d'euros! La taula següent mostra el creixement dels costos d'emmagatzematge en funció de la capacitat d'emmagatzematge del sistema i del factor de replicació utilitzat.

Cost €/GB
0,1

Capacitat d'emmagatzematge					
GB	**TB**	**PB**	**Cost Factor Repl. 3**	**Cost Factor Repl. 2**	**Cost Sense replicació**
1.000	1	0,001	300,00 €	200,00 €	100,00 €
100.000	100	0,1	30.000,00 €	20.000,00 €	10.000,00 €
200.000	200	0,2	60.000,00 €	40.000,00 €	20.000,00 €
500.000	500	0,5	150.000,00 €	100.000,00 €	50.000,00 €
1.000.000	1.000	1	300.000,00 €	200.000,00 €	100.000,00 €
2.000.000	2.000	2	600.000,00 €	400.000,00 €	200.000,00 €
5.000.000	5.000	5	1.500.000,00 €	1.000.000,00 €	500.000,00 €
10.000.000	10.000	10	3.000.000,00 €	2.000.000,00 €	1.000.000,00 €
20.000.000	20.000	20	6.000.000,00 €	4.000.000,00 €	2.000.000,00 €
30.000.000	30.000	30	9.000.000,00 €	6.000.000,00 €	3.000.000,00 €
100.000.000	100.000	100	30.000.000,00 €	20.000.000,00 €	10.000.000,00 €

Taula 4.3: Comparativa dels costos d'emmagatzematge en funció de la capacitat i el factor de replicació.

A més, per estimar més acuradament aquest estalvi caldria tenir en compte els costos de:

- Maquinari destinat a replicació.
- Energia consumida.
- Hores de personal destinat al manteniment d'aquest maquinari.
- Equipament de xarxa.
- Cablejat.
- Generadors d'energia elèctrica.
- Lloguer de l'espai físic, si n'hi ha.
- Costos vinculats amb l'espai físic on s'allotgen les màquines, com: sistemes de refrigeració, renovació d'aire, regulació de la humitat, extinció d'incendis, seguretat, assegurances,...

Com hem dit, la replicació com a solució per garantir la tolerància a fallades és altament efectiva, però poc eficient, ja que és una solució molt costosa en termes econòmics, sobretot en sistemes de molt alta capacitat. Per tant, es fa necessari explorar nous mecanismes per protegir les dades contra les diverses fallades que pateixen els sistemes d'emmagatzematge distribuït.

En base a això, un dels objectius d'aquest projecte serà trobar, analitzar i implementar una solució pel sistema de redundància més eficient que la replicació sense comprometre els nivells de tolerància a fallades.

En aquest punt és quan entren en joc els codis correctors d'errors. Els codis correctors d'errors són un tècnica utilitzada des de fa algunes dècades per fer transmissions de dades sobre canals de comunicació inestables o sorollosos.

La idea és que l'emissor codifica el missatge d'una forma redundant utilitzant un codi corrector d'errors. Les dades del missatge es parteixen en múltiples paquets, on s'hi afegeixen bits addicionals d'informació. Gràcies a aquesta redundància, el receptor de les dades pot reconstruir els paquets encara que s'hagin produït algunes pèrdues o errors durant la fase de transmissió. Això evita haver de sol·licitar la retransmissió de la informació.

Alguns exemples de situacions on s'utilitzen els codis correctors d'errors són:

- Transmissió de dades des de l'espai, on la alta latència fa poc pràctic un esquema de retransmissió.

- Suports digitals òptics, com el CD o el Blu-Ray, els quals utilitzen els codis correctors d'errors per tal de mitigar els efectes de ratllades en la superfície o errors de lectura.

- Aplicacions de comunicació en temps real, com les videoconferències o l'*streaming* multicast de vídeo, on no és possible sol·licitar la retransmissió de les dades.

A banda d'aquests exemples, els sistemes d'emmagatzematge, com el RAID, també s'han beneficiat de les bones propietats dels codis correctors d'errors. A continuació

explicarem quina és la aplicació dels codis correctors d'errors dins l'àmbit dels sistemes d'emmagatzematge distribuït.

4.2 Introducció als codis correctors d'errors aplicats en sistemes d'emmagatzematge distribuïts.

En els sistemes d'emmagatzematge distribuït es veuen involucrats un gran nombre de dispositius (disc durs, fonts d'alimentació, routers, cablejat, sistemes de refrigeració, etc.). És per això que aquests sistemes tendeixen a tenir alguna part dels seus components que no funcionen de la forma esperada, en qualsevol punt en el temps.

En conseqüència, és essencial disposar de mecanismes que assegurin la tolerància a fallades.

Els codis correctors d'errors són una tècnica que permet aconseguir tolerància a fallades transformant un conjunt de dades format per k blocs en un conjunt més gran de n blocs codificats. Aquesta transformació es basa en afegir dades redundants al conjunt de dades original. El procés d'afegir dades redundants s'anomena **codificació**.

El procés de codificació utilitza un algorisme que genera les dades redundants fent combinacions de les dades originals. L'algorisme varia en funció del codi utilitzat.

A diferència de la replicació, els codis correctors d'errors són capaços de recuperar el mateix nombre de fallades sense fer còpies de tota la informació. Això és beneficiós ja que permet reduir l'espai ocupat destinat a la informació redundant i, per tant, estalviar costos d'emmagatzematge.

L'aspecte clau dels codis correctors d'errors és que ens permeten recuperar el conjunt de dades original a partir de qualsevol subconjunt de k blocs codificats. El procés d'obtenció de les dades originals s'anomena **descodificació**.

El nivell de tolerància a fallades el marca la relació que s'estableix entre el número de dades original (k) i la quantitat de redundància afegida ($n - k$).

A continuació presentem un esquema amb l'objectiu de fer més clara la comprensió dels elements claus en els codis correctors d'errors.

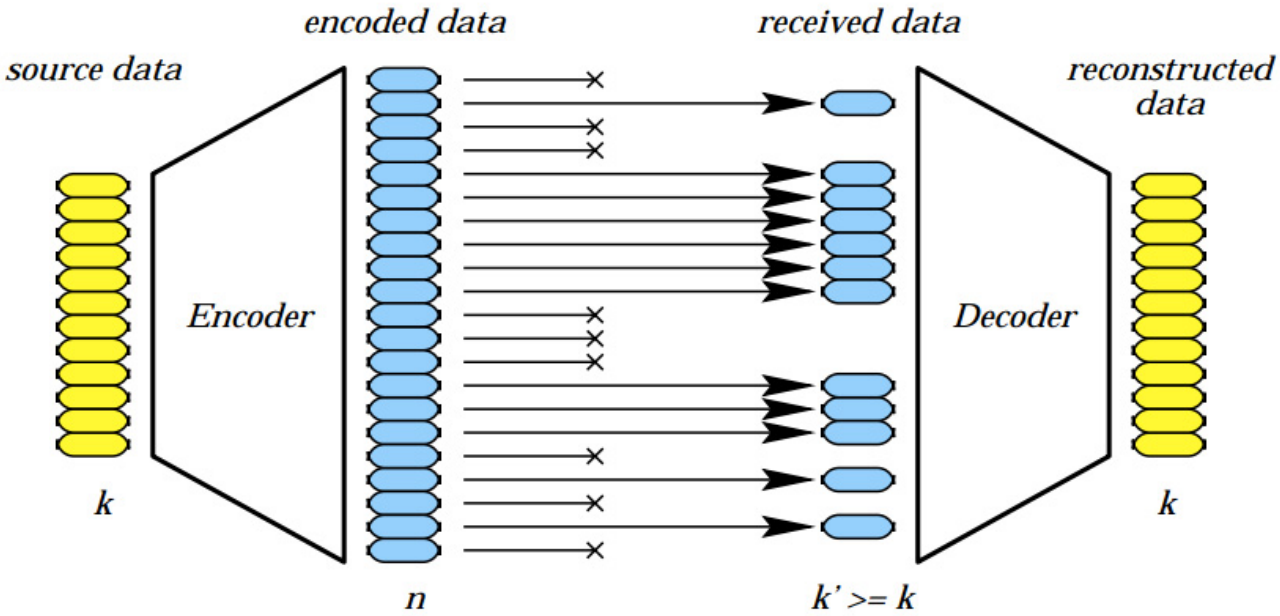

Figura 4.1: Esquema d'aplicació d'un codi corrector d'errors. Autor: Luigi Rizzo. Font: http://info.iet.unipi.it/~luigi/research.html

Hi ha algunes consideracions que es desprenen d'aquest esquema i val la pena destacar:

- Les dades originals es parteixen en k fragments.
- Les dades es codifiquen i es genera un conjunt més gran anomenat n, format pels k fragments originals i $n - k$ fragments d'informació redundant.
- El conjunt de dades codificades s'anomena *stripe*.
- Un codi corrector d'errors s'identifica segons els seus paràmetres n i k com codi (n, k) o codi *k-de-n*.
- La tolerància a fallades del codi és de $n - k$ pèrdues.
- A més informació redundant, més tolerància a fallades.
- Qualsevols k blocs dins l'*stripe* poden recuperar els k blocs originals.

Un codi corrector d'errors (n, k), o simplement codi k-de-n, codifica k blocs de dades en $n > k$ blocs. Fent una descripció informal, un codi k-de-n agafa k blocs de dades i genera $n - k$ blocs redundants, de forma que qualsevol subconjunt de k blocs (ja siguin dades o blocs redundants) pot reconstruir els k blocs de dades originals.

Emmagatzemant cada bloc de l'*stripe* en un node diferent, les dades queden protegides davant la fallada simultània de fins a $n - k$ nodes.

Per exemple, si a i b són blocs de dades, podríem generar dos blocs redundants $a + b$ i $a - b$. Donat un *stripe* que contingui els quatre blocs $\{a,\ b,\ a + b,\ a - b\}$, qualsevol subconjunt de dos blocs podria reconstruir a i b. Donats $a + b$ i b, podem obtenir a restant b de $a + b$. Llavors tenim un codi corrector d'errors 2-de-4, el qual pot tolerar la pèrdua de 2 blocs qualsevols del *stripe*. O el que és el mateix, el sistema té una tolerància a fallades de 2, i una taxa de transmissió de $R = 1/2$.

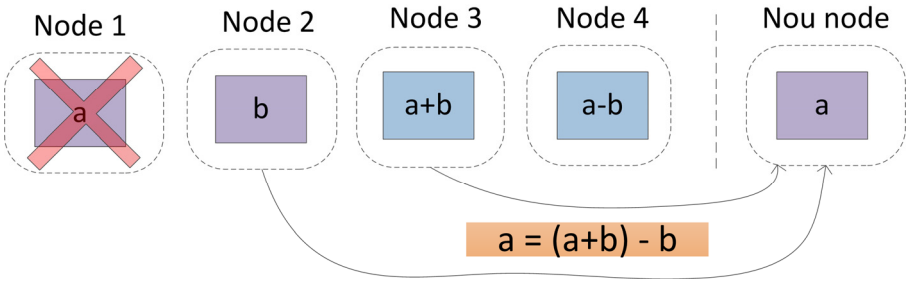

Figura 4.2: Exemple de funcionament de regeneració de dades amb codis correctors d'errors. Elaboració pròpia.

Observem que això és molt més eficaç que la replicació de 2 còpies, amb el mateix espai ocupat. Si simplement repliquem a i b, obtenim $\{a, b, a, b\}$, si més tard perdem les dues rèpliques d'a, ja no podem tornar a reconstruir a. És a dir, tenim un sistema amb una tolerància a fallades de 1 i una taxa de transmissió de $R = 1/2$.

Una classe interessant de codis correctors d'errors són els codis lineals. S'anomenen així perquè poden ser analitzats utilitzant les propietats de l'àlgebra lineal. A l'apartat següent repassarem els fonaments teòrics importants relacionats amb els codis lineals.

4.3 Codis lineals

En aquest apartat repassarem les bases teòriques relacionades amb els codis lineals. La presentació d'aquest contingut és necessària per sustentar la implementació duta a terme en aquest projecte.

La recopilació que presentem a continuació s'ha elaborat utilitzant diversos materials bibliogràfics: [17] [18] [19].

4.3.1 Cos finit o cos de Galois

Un cos finit és una estructura algebraica que conté un nombre finit d'elements. Els cossos finits es classifiquen pel seu ordre, o número d'elements, que ve donat per p^n, on p és un nombre primer, i n és un enter positiu. A més, cada cos finit amb el mateix número d'elements és únic. Això justifica que la notació d'un cos finit només especifiqui el seu ordre. Un cos finit es denota amb $\mathbb{F}_q$ on $q = p^n$.

Exemple: El cos binari

El cos binari està representat per $\mathbb{F}_2$ i és composat pel polinomi base únic de grau 1 amb coeficients dins el conjunt d'enters mòdul 2, que és x. Qualsevol polinomi mòdul x amb coeficients en el conjunt d'enters mòdul 2, dona com a resultat un símbol dins el conjunt {0,1}, per tant, $\mathbb{F}_2$ = {0,1}

Sigui $\mathbb{F}_q$, on $q = p^r$, un cos finit de q elements. Un símbol sobre $\mathbb{F}_q$ pot ser representat com un polinomi de grau més petit que r amb coeficients sobre $\mathbb{F}_p$. Qualsevol polinomi de grau menor que r pot ser representat per un array ordenat d'r coeficients. Si, per exemple, $r = 2$ llavors $x + 2$ pot ser representat per l'array $(1,2)$ o 1 pot ser representat

per $(0,1)$.

Així, el cos $\mathbb{F}_q$ és una extensió del cos $\mathbb{F}_p$ on cada símbol de $\mathbb{F}_q$ és un array de r coeficients sober $\mathbb{F}_p$. Això és aplicable a qualsevol cos. Per exemple, és possible construir $\mathbb{F}_{2^4}$ a partir de $\mathbb{F}_{2^2}$, on cada símbol sobre $\mathbb{F}_{2^4}$ serà un *array* de dos coeficients sobre $\mathbb{F}_{2^2}$.

4.3.2 Teoria de codis lineals

Un codi sobre $\mathbb{F}_q$ és una associació entre $\mathbb{F}_q^k$ i $\mathbb{F}_q^n$ la qual converteix vectors de *k* de coordenades sobre $\mathbb{F}_q$ a un vector de *n* coordenades també sobre $\mathbb{F}_q$.

En aquest projecte, només ens interessarem pels codis lineals, on l'associació és lineal i el conjunt de vectors en $\mathbb{F}_q^n$ forma un subespai de $\mathbb{F}_q^n$.

4.3.2.1 Codis lineals i matrius generadores i de comprovació de paritat

Considerem $v \in \mathbb{F}_q^k$ el vector d'informació a ser codificat. Si el codi és lineal, la paraula codificada $v \in \mathbb{F}_q^n$ corresponent a la codificació de *v*, pot ser obtinguda amb la multiplicació de matriu de vector

$$c = vG,$$

on G és la matriu generadora del codi lineal.

Definició: Matriu generadora

La matriu generadora G d'un codi C sobre $\mathbb{F}_q$, és una matriu $k \times n$ amb coeficients sobre $\mathbb{F}_q$. Les files de G formen la base del subespai lineal C de $v \in \mathbb{F}_q^n$. Els vectors d'aquest subespai lineal s'anomenen "codewords" de C i el conjunt de "codewords" s'anomena el "codebook" de C.

El paràmetre *n* és la mida del codi i el paràmetre *k* és la dimensió del codi C. La taxa del codi és la quantitat d'informació dividida per la quantitat de dades, la qual està composada per la informació més la redundància.

Definició: Taxa de transmissió

La taxa de transmissió d'un codi C de mida n i dimensió k, està definida per $R = k/n$.

Exemple: El codi de repetició

El codi de repetició agafa un símbol d'informació i el repeteix n vegades. Per exemple, un codi de repetició (3,1) on $n = 3$ *i* $k = 1$*, definit com*

$$c = v(1,1,1) = (v, v, v),$$

on $v \in \mathbb{F}_2$*. Notem que* $R = 1/3$*.*

Exemple: El codi de comprovació de paritat única

El codi de comprovació de paritat única de mida $n = 3$ i $k = 2$ consisteix en afegir als símbols d'informació un símbol de paritat redundant. Aquest símbol de paritat és la suma de tots els símbols d'informació.

$$(v1, v2)G \;=\; (v1, v2)\begin{pmatrix}1 & 0 & 1\\ 0 & 1 & 1\end{pmatrix} \;=\; (v1, v2\,, v1 + v2)$$

On $v_1, v_2 \in \mathbb{F}_q$. La Taxa de transmissió del codi és de $R \;=\; 2/3$.

Aquests dos exemples tenen alguna cosa en comú ja que les dues "codewords" produïdes contenen els símbols d'informació original. Això passa quan la matriu generadora té la forma

$$G \;=\; (I_k|A),$$

on A és una matriu $k \;\times\; (\,n \;-\; k\,)$, i I_k és la matriu identitat $k \;\times\; k$:

$$\begin{pmatrix}1 & & \\ & \ddots & \\ & & 1\end{pmatrix}$$

En aquest cas, es diu que el codi és *sistemàtic*. És a dir, existeix una copia no codificada de les dades. Per cada codeword de mida n, la informació és en les primeres k coordenades i la redundància en les últimes $n - k$ coordenades.

Hem vist com hem codificat la informació per crear un codeword. Ara ens plantejarem com fer la pregunta inversa: com és possible determinar, donat un vector sobre $\mathbb{F}_q^n$, si aquest és un codeword d'un codi C? Aquesta pregunta es pot respondre utilitzant la matriu de comprovació de paritat.

Definició: Matriu de comprovació de paritat

La matriu de comprovació de paritat H d'un codi C sobre $\mathbb{F}_q$, és una matriu $(n - k) \;\times\; n$ amb coeficients sobre $\mathbb{F}_q$. Les files d'H formen una base del subespai lineal que conté tots els vectors ortogonals del "codebook" de C. La multiplicació de la matriu de vectors de la matriu de comprovació de paritat amb qualsevol vector del "codebook" de C, dona com a resultat un vector de 0s amb longitud $n - k$.

La matriu de comprovació de paritat de la matriu generadora $G \;=\; (I_k|A)$ és H. Si c és un "codeword" vàlid, llavors $Hc^T = \begin{pmatrix}0\\ \vdots\\ 0\end{pmatrix}$.

Exemple: El codi de Hamming.

La matriu de comprovació de paritat del codi de Hamming de longitud 7 i dimensió 4 amb la matriu generadora

$$G = \begin{pmatrix}1 & 0 & 0 & 0 & 1 & 1 & 0\\ 0 & 1 & 0 & 0 & 0 & 1 & 1\\ 0 & 0 & 1 & 0 & 1 & 0 & 1\\ 0 & 0 & 0 & 1 & 1 & 1 & 1\end{pmatrix},$$

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

La matriu H pot ser utilitzada per detectar errors en un "codeword". El procediment per obtenir el vector d'informació del "codeword" rebut de forma errònia s'anomena descodificació.

4.3.2.2 Conceptes de mesura dels codis

La redundància s'utilitza per poder recuperar porcions de la informació que s'han perdut durant una transmissió per un canal de comunicació entre un emissor i un receptor. En funció de la quantitat d'informació redundant que l'emissor afegeixi, el receptor serà més o menys capaç de recuperar les possibles pèrdues d'informació que s'hagin produït.

A continuació introduirem alguns conceptes per tal de mesurar quina és la tolerància a pèrdues d'informació que un codi permet suportar.

Definició: La distància de Hamming

La distància de Hamming entre dos vectors c_1i c_2 és el nombre de posicions en que c_1 i c_2 són diferents, i ve denotada per $d_H(c_1, c_2)$.

Definició: El pes de Hamming

El pes de Hamming d'un vector c és el número de posicions diferents de 0, i ve denotada per $w_H(c)$.

Definició: Distància mínima de Hamming

La distància mínima de Hamming d'un codi lineal C és la mínima distància entre dos "codewords" $c_1, c_2 \in C$ qualsevols.

$$d(C) = \min_{c_1 \neq c_2 \in C} \big(d(c_1, c_2)\big) = \min_{c_1 \neq c_2 \in C} w_H(c_1 - c_2)$$

De fet, com que C és lineal, no cal comprovar cada parell de "codewords". És suficient amb calcular:

$$d(C) = \min_{c \in C, c \neq 0} w_H(c),$$

ja que el vector resultant de la operació $c_1 - c_2$ és també un "codeword".

Quan es vol donar èmfasi a la distància mínima, es pot escriure explícitament com un paràmetre del codi de la forma $[n, k, d]$. Aquesta notació identifica un codi lineal C de longitud n i dimensió k i distància mínima de Hamming d.

Exemple: *El codi de paritat única a $\mathbb{F}_2$ té el següent "codebook"*

$$\{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}.$$

La distància mínima de Hamming d'aquest codi és $d = 2$*, per tant es un codi* $[3, 1, 2]$.

Donat que cada dos "codewords" són diferents en, almenys, d coordenades, ja podem definir la capacitat de correcció d'esborralls.

Definició: Capacitat correctora d'un codi

Un codi lineal C de paràmetres $[n, k, d]$ és t-corrector si $t = [\frac{d-1}{2}]$. Això significa que podem corregir fins a t errors en cada paraula codi.

Si, a més, el codi té s esborralls, aleshores podrà corregir t errors i s esborralls sempre que $2t + s < d$. En el cas que $t = 0$, és a dir, si tots els errors són esborralls, aleshores el codi podrà corregir fins a un màxim de $d - 1$ esborralls.

4.3.2.3 Codis *Maximum Distance Separable* (MDS)

En aquest punt ja és palesa la relació que s'estableix entre la quantitat de redundància afegida i la distància mínima (i, per tant, la capacitat de correcció) d'un codi. Conceptualment, un codi eficient en termes de capacitat de correcció implica que per a una certa quantitat de redundància, la distància mínima sigui la més gran possible. Podem observar que l'eficiència màxima s'aconsegueix quan, per cada símbol de redundància afegit, la distància mínima s'incrementa en una unitat. Aquest tipus de codis s'anomenen codis Maximum Distance Separable (MDS). Això és perquè els "codewords" tenen la distància màxima possible entre ells.

Teorema:

Donada la matriu de comprovació de paritat H del codi C, la distància mínima és d si, i només si, qualsevol conjunt de $d - 1$ columnes d'H són linealment independents i algun conjunt de d columnes són linealment dependents.

Teorema: Singleton Bound

Donat un codi C amb les característiques $[n, k, d]$, llavors $d \leq n - k + 1$.

Definició: Codi *Maximum Distance Separable* (MDS)

Un codi que compleix el *Singleton Bound* es considera un codi *Maximum Distance Separable* i pot tolerar fins a $n - k$ pèrdues.

Es coneixen alguns codis MDS que són utilitzats per multitud d'aplicacions. Un codi MDS és especialment adequat quan cal minimitzar la quantitat de redundància a afegir. Els codis Reed-Solomon són, possiblement, els codis MDS més utilitzats i estan basats en construccions algebraiques utilitzant polinomis. Els principals inconvenients dels codis Reed-Solomon per la transmissió de dades és que necessiten un conjunt fix de k coordenades abans de codificar o decodificar i que la complexitat del seu algorisme de descodificació és de $O(n^2)$. No obstant, la seva eficiència en termes de taxa de transmissió i el bon funcionament per a multitud de paràmetres k i n, el fan una bona elecció per la majoria de aplicacions basades en codis.

4.3.2.4 Codis d'*array* o *Array Codes*

Com hem explicat anteriorment, és possible crear la extensió d'un cos $\mathbb{F}_{q^t}$ a partir d'un cos base $\mathbb{F}_q$. Això vol dir que qualsevol símbol sobre $\mathbb{F}_{q^t}$ pot ser representat com un *array* de t símbols sobre $\mathbb{F}_q$. Finalment, es pot codificar un vector d'informació $v \in \mathbb{F}_{q^t}^k$ utilitzant un codi $[n, k, d]$ sobre $\mathbb{F}_{q^t}$ i produir un codeword $c \in \mathbb{F}_{q^t}^n$.

Aquest tipus de codis s'anomena codi d'*array*. Els símbols sobre $\mathbb{F}_{q^t}$ d'un codi d'*array* s'anomenen coordenades de l'*array*, mentre que els símbols definits sobre $\mathbb{F}_q$ s'anomenen coordenades.

4.4 El problema de la reparació

L'emmagatzematge de dades mitjançant els codis correctors d'errors es basa en partir les dades en fragments, que son distribuïts en un conjunt de nodes. Aquesta tècnica permet reduir la redundància respecte la replicació simple i al mateix temps garanteix la tolerància a fallades.

Com ja hem apuntat, els sistemes d'emmagatzematge distribuït tenen unes dimensions grans que fan que la recuperació de fallades sigui una part de les operacions habituals, més que una excepció. Donat que la redundància ha de ser restituïda a mesura que els nodes fallen, hi ha un aspecte que esdevé molt important: cal tenir en compte quin és l'ample de banda utilitzat a fi de restituir la pèrdua d'informació.

Tot i que els grans datacenters estan ben equipats amb uns recursos de xarxa d'alta capacitat, la xarxa segueix sent un recurs costós i limitat. Els clústers reben moltes peticions de forma concurrent i han de disposar d'una xarxa no congestionada per oferir un bon temps de resposta. Sorgeix, per tant, una qüestió important: com es poden generar els fragments codificats de forma que es minimitzi la transferència de dades a través de la xarxa?

Considerem, com a exemple, una situació on s'ha generat redundància amb un codi (4,2). El node 1 ha fallat, i un nou node entra en joc per reconstruir les dades. El nou node necessita descarregar qualsevols k blocs per reconstruir a i b. Per exemple, els blocs c, d, a+c i b+d.

Assumim que la mida dels blocs es d'1 GB, s'ha hagut de transferir 4GB a través de la xarxa per reconstruir-ne només 2GB. Aquest és el problema de la reparació [20] [21]. El tràfic de reparació és alt ja que s'ha consumit recursos de xarxa per transmetre més informació de la que s'acabarà generant. A banda del consum intensiu de la xarxa, també s'ha produït gran activitat d'E/S als discs de 3 dels 4 nodes que queden vius [22].

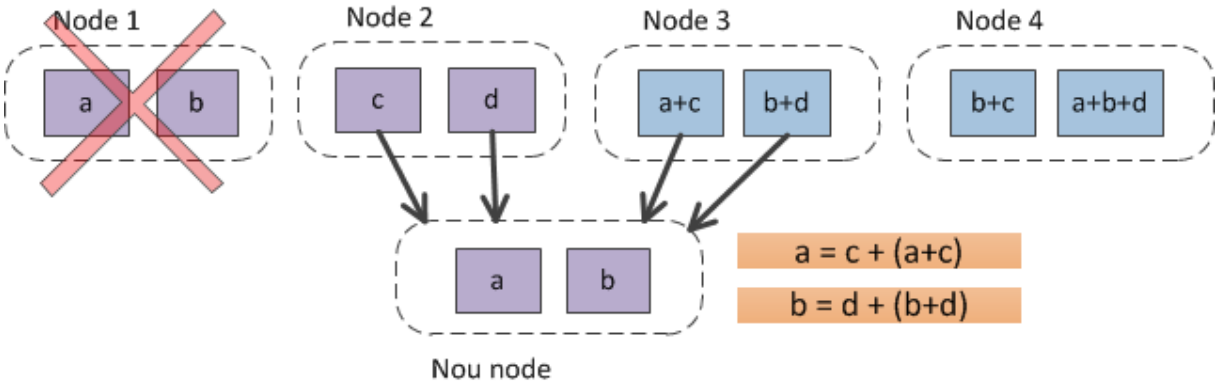

Figura 4.3: Representació del tràfic descrit pel problema de la reparació. Elaboració pròpia.

Observem que en aquest sentit, el sistema tradicional de la replicació és òptim, ja que en cas de fallada la transferència total és exactament la mida dels blocs desapareguts.

Els codis regeneratius són una classe dels codis correctors d'errors lineals que busquen un equilibri òptim entre l'ample de banda necessari per reparar un node avariat i la quantitat de dades emmagatzemades per cada node de la xarxa. Aquest equilibri permet atenuar el problema de la reparació.

A la propera secció presentarem els codis regeneratius quasi-ciclics. Amb aquest apartat completarem els fonaments teòrics necessaris abans de plantejar el desenvolupament a realitzar.

4.5 Els codis regeneratius flexibles quasi-ciclics

Els codis regeneratius flexibles quasi-ciclics, o *Quasi-cyclic Flexible minimum storage regenerating codes* (QCFMSR), són una família de codis regeneratius basada en codis quasi-ciclics.

Els codis QCFMSR són fruit de la recerca realitzada pel doctorand Sr. Bernat Gastón, pel Dr. Jaume Pujol i per la Dra. Mercè Villanueva del Departament d'Enginyeria de la Informació i de les Comunicacions de la Universitat Autònoma de Barcelona.

En aquest apartat ens limitarem a donar les nocions bàsiques d'aquesta mena de codis. Convidem al lector interessat en la matèria a adreçar-se a l'article publicat a la xarxa[12] per tal d'aprofundir en la fonamentació teòrica.

4.5.1 Procés de codificació

Començarem aquest apartat donant una definició general d'un codi d'*array* quasi-ciclic C, sobre el qual basarem una definició concreta d'un codi QCFMSR.

Donat un codi d'*array*[13] C de longitud $n = 2k$ i dimensió k sobre $\mathbb{F}_{q^2}$. Aquest codi C està construït amb els coeficients $\zeta_1, \dots, \zeta_k$ sobre $\mathbb{F}_q$. La codificació amb C es farà en el cos finit base $\mathbb{F}_q$ de la forma següent: un vector d'informació $v \in \mathbb{F}_{q^2}^k$, representat com

[12] B. Gastón, J. Pujol, i M. Villanueva. Quasi-cyclic regenerating codes. Versió del 16 de Maig de 2013. arXiv:1209.3977v2 [cs.IT].
[13] Veure apartat de codis d'*array* a la secció de teoria de codis.

$v = (v_1, \dots, v_n)$ sobre $\mathbb{F}_q$ es codifica en un vector $c \in \mathbb{F}_{q^2}^n$, que serà el "codeword". Les coordenades d'aquest vector codificat c poden ser representades com $c = (c_1, \dots, c_{2*n}) = (v_1, \dots, v_n, \rho_1, \dots, \rho_n)$ sobre $\mathbb{F}_q$. Les coordenades $v_1, \dots, v_n$ són les dades originals. Les coordenades $\rho_1, \dots, \rho_n$ són les posicions de redundància, que són generades amb la següent equació:

$$\rho_i = \sum_{j=i+1}^{k+i} \zeta_{j-i}\, v_j \quad i = 1, \dots, n$$

on $\zeta_l \in \mathbb{F}_q \setminus \{0\}$ per $l = 1, \dots, k$ i $j = i+1, \dots, k+i \bmod n$. La codificació es fa sobre $\mathbb{F}_q$ utilitzant la formula anterior, la qual caracteritza els codis quasi-ciclics. La taxa de transmissió d'aquest codi val $R = 1/2$.

Ara que ja hem definit de forma general un codi d'*array* quasi-ciclic C, presentarem la definició d'un codi QCFMSR. Un codi QCFMSR sobre $\mathbb{F}_{q^2}$ amb els paràmetres $[2k, k, r]$ és un codi regeneratiu construït a partir del codi d'*array* C.

La codificació d'un conjunt de dades d'entrada de mida M amb un codi QCFMSR es fa de la següent manera. Es parteix el conjunt de dades d'entrada en k fragments de mida igual sobre $\mathbb{F}_{q^2}$, o el que és el mateix, en $n = 2k$ fragments sobre $\mathbb{F}_q$. Els fragments es representen com un vector $v = (v_1, \dots, v_n)$.

Per implementar un codi QCFMSR sobre $\mathbb{F}_{q^2}$ amb paràmetres $[2k, k, r]$ cal disposar d'un conjunt de $n = 2k$ nodes d'emmagatzematge, que podem denotar com $\{s_1, \dots, s_n\}$. Cada un dels nodes d'emmagatzematge s_i, on $i = 1, \dots, n$, emmagatzema dos coordenades (v_i, ρ_i) sobre $\mathbb{F}_q$. Cada parell de coordenades (v_i, ρ_i) es pot veure com un *array* de coordenades sobre $\mathbb{F}_{q^2}$. La mida de cada coordenada sobre $\mathbb{F}_q$ és $M/2k$ i la mida de cada *array* de coordenades emmagatzemades a s_i és de $\alpha = M/k$.

$F = (I|Z)$ és una matriu $n \times 2n$, on I és la matriu identitat $n \times n$, i Z és una matriu circulant $n \times n$ definida a partir dels coeficients $\zeta_1, \dots, \zeta_k$ de la forma següent:

$$Z = \begin{pmatrix} 0 & 0 & \dots & 0 & \zeta_k & \zeta_{k-1} & \dots & \zeta_1 \\ \zeta_1 & 0 & \dots & 0 & 0 & \zeta_k & \dots & \zeta_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \zeta_k & \zeta_{k-1} & \dots & \zeta_1 & 0 & 0 & \dots & 0 \\ 0 & \zeta_k & \dots & \zeta_2 & \zeta_1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \zeta_k & \zeta_{k-1} & \zeta_{k-2} & \dots & 0 \end{pmatrix}$$

La matriu F representa el codi d'*array* C, i també el codi QCFMSR construït a partir de C. Cada fila és la codificació d'una coordenada sobre el cos base $\mathbb{F}_q$ i cada node està representat per dues columnes, una de I i l'altra de Z.

Les coordenades d'informació són representades per la matriu identitat I, mentre que les coordenades de redundància són representades per la matriu circulant Z.

4.5.2 Reconstrucció de dades

En cas que es produeixi una pèrdua de dades en el sistema d'emmagatzematge, caldrà fer una reconstrucció. Durant aquest projecte utilitzem la paraula "descodificació" per referir-nos a la reconstrucció de dades.

Quan es realitza el procés de reconstrucció, es contacta amb qualsevols k nodes $\{s_1, \dots, s_k\}$ i es descarrega l'*array* de coordenades (que conté la informació i la redundància associada) $(v_1, \rho_1), \dots, (v_k, \rho_k)$. La tria dels k nodes es pot fer de manera aleatòria ja que qualsevols k nodes contenen la informació necessària per reconstruir el fitxer.

Per trobar les dades perdudes, s'haurà de construir una matriu de dimensions mínimes $m \times k$, on k és fix i representa la quantitat de dades (la dimensió del codi), i m compleix que $m \geq k$. Per tant, és una matriu que té com a mínim el mateix nombre de columnes que de files, o més files que columnes.

Cada fila de la matriu representa una coordenada de dades o de redundància. Un cop construïda la matriu, ja només cal aplicar l'algorisme de Gauss-Jordan per obtenir les dades perdudes.

4.5.3 Exemple d'aplicació dels codis QCFMSR

En aquest projecte implementarem un codi QCFMSR amb paràmetres $[10, 5, 6]$ sobre $\mathbb{F}_{256^2}$. Els coeficients del codi sobre $\mathbb{F}_{256}$ han estat calculats[14] per tal de maximitzar el rang i són $\{1,1,2,8,1\}$. A continuació presentem en una taula les equivalències entre $\mathbb{F}_{256^2}$ i $\mathbb{F}_{256}$.

$\mathbb{F}_{256^2}$	$\mathbb{F}_{256}$
$k_{(\mathbb{F}_{256^2})} = 5$	$k_{(\mathbb{F}_{256})} = 10$
$n_{(\mathbb{F}_{256^2})} = 10$	$n_{(\mathbb{F}_{256})} = 20$

Per tal de fer la codificació, un conjunt de dades originals es fragmentarà en 10 coordenades d'informació $v = (v_1, v_2, \dots, v_{10})$. Cada v_i per $i = 1, \dots, 10$ s'emmagatzemarà en un node $s_i = (v_i, \rho_i)$, juntament amb el seu símbol de redundància corresponent ρ_i, que és calculat utilitzant una matriu quasi-ciclica F de la forma següent. Notem que aquesta matriu es mostra transposada.

[14] No mostrem el procés de construcció d'aquest codi ja que això no entra en l'objectiu del present treball.

$$F = \left(\begin{array}{cccccccccc|cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 2 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 2 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 8 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 8 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 8 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 8 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 8 & 2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 8 & 2 & 1 & 1 & 0 \end{array}\right)$$

Per exemple, per a un vector d'informació sobre $\mathbb{F}_{256}$

$$v = (167, 80, 158, 118, 92, 166, 124, 76, 190, 89)$$

El codeword generat $c = (v_1, \dots, v_n, \rho_1, \dots, \rho_n)$ seria:

$$c = (167, 80, 158, 118, 92, 166, 124, 76, 190, 89, 94, 117, 240, 230, 130, 4, 65, 144, 172, 29)$$

la codificació de ρ_1 seria $\rho_1 = 1 * 80 + 1 * 158 + 2 * 118 + 8 * 92 + 1 * 166 = 94$. Cal dir que les operacions son sobre el cos $\mathbb{F}_{256}$. L'aritmètica en un cos finit és diferent de la aritmètica entera estàndard.

La figura 4.4 mostra el procés de codificació de les dades. En primer, lloc, el fitxer és fragmentat en n símbols sobre $\mathbb{F}_{256}$. Aquests trossos són codificats fent us de la matriu F definida anteriorment, produint $2n$ símbols sobre $\mathbb{F}_{256}$. Per acabar, cada símbol de dades s'emmagatzema amb el seu símbol de paritat corresponent en un node.

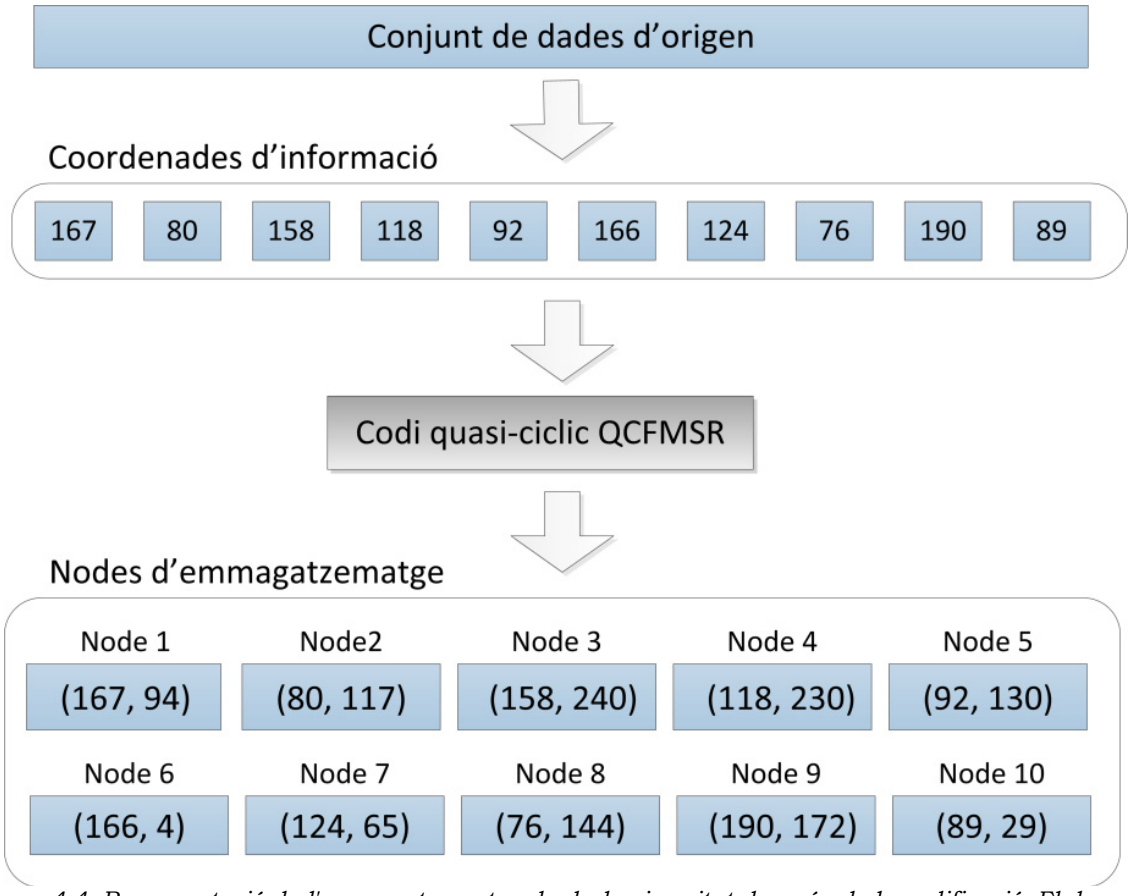

Figura 4.4: Representació de l'emmagatzematge de dades i paritat després de la codificació. Elaboració pròpia.

En el cas de la descodificació, o reconstrucció, tenim un o diversos nodes que han fallat. Caldrà fer una tria aleatòria de k nodes que segueixin vius i muntar una matriu amb les coordenades que aquests emmagatzemen.

En el següent exemple han fallat els nodes 5 i 6. Triem els nodes $\{1,2,3,4,8\}$ i muntem una matriu amb les coordenades que contenen. Reconstruirem les dades originals solucionant el sistema d'equacions amb l'algorisme de Gauss. La redundància no s'obté directament amb Gauss, però una vegada s'obtenen les dades originals, es pot tornar a obtenir amb el procés de codificació.

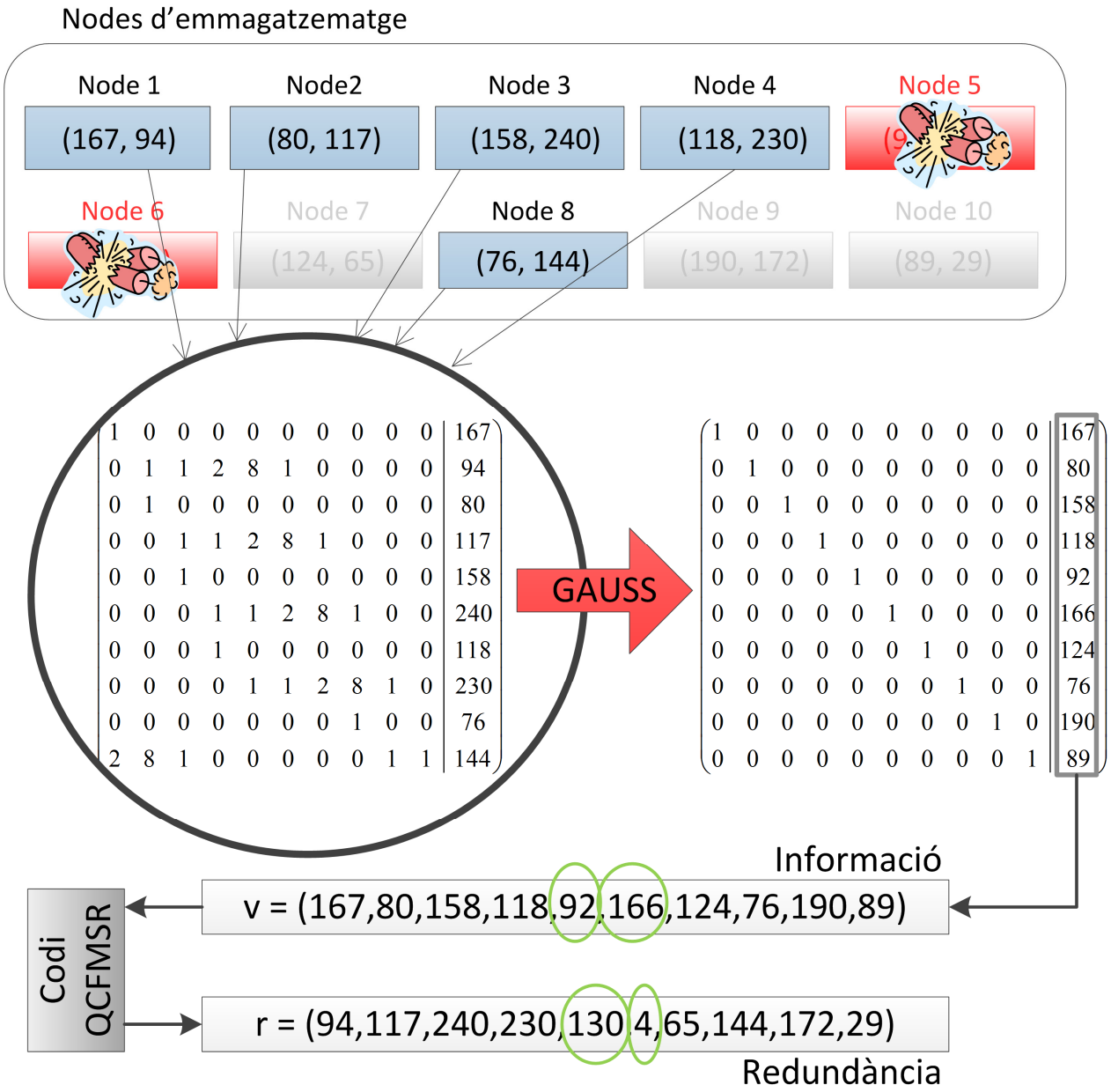

Figura 4.5: Representació del procés de descodificació amb un exemple. Elaboració pròpia.

Resolent aquest sistema d'equacions amb Gauss obtenim les dades originals, i posteriorment utilitzem el codi per tornar a codificar la paritat.

Capítol 5: Anàlisi del sistema

5.1 Anàlisi de l'HDFS RAID: El sistema de redundància implementat a Hadoop per Facebook.

L'HDFS RAID és un software que implementa a Hadoop un sistema de redundància amb codis correctors d'errors [23] [24] [25] [26] [22]. És un projecte desenvolupat integrament per Facebook sota la mateixa llicència de Hadoop (Apache v2).

La motivació de Facebook per portar a terme aquest desenvolupament va sorgir l'any 2009 quan l'equip de dades va predir que el clúster de Hadoop que operaven tindria un creixement exponencial de dades durant els anys posteriors. Aquestes prediccions s'han complert ja que Facebook opera actualment el clúster més gran de Hadoop que es coneix, amb un espai ocupat de dades que supera els 100PB (segons dades de l'any 2012).

Facebook posa a disposició de qualsevol el codi de la versió de Hadoop que utilitzen en producció. És un codi basat en la versió oficial de Hadoop 0.20, i inclou l'HDFS RAID i altres modificacions. La distribució d'aquest software es fa a través d'un repositori ubicat a https://github.com/facebook/hadoop-20.

El disseny de l'HDFS RAID està basat en el DiskReduce[15], una idea del centre de recerca Parallel Data Laboratory, ubicat a la Universitat Carnegie Mellon. Una premissa important a l'hora de desenvolupar l'HDFS RAID va ser elaborar un software modular, per poder mantenir-lo com una capa que funcionés per sobre de l'HDFS. Als enginyers de Facebook no els interessava mesclar el codi de l'HDFS, que ja consideraven prou complex, amb el codi de l'HDFS RAID [26].

L'HDFS RAID implementa dos tipus de codis correctors d'errors: el codi basat en XOR i el codi Reed-Solomon, amb uns paràmetres per defecte de (14,10). Un dels grans beneficis de l'HDFS RAID és que gràcies a això, s'incrementa la protecció davant de la corrupció de dades i permet baixat els nivells de replicació mantenint les mateixes garanties de tolerància a fallades. Això resulta en un estalvi d'espai d'emmagatzematge significatiu, com es pot observar a la figura següent.

[15] Veure [34].

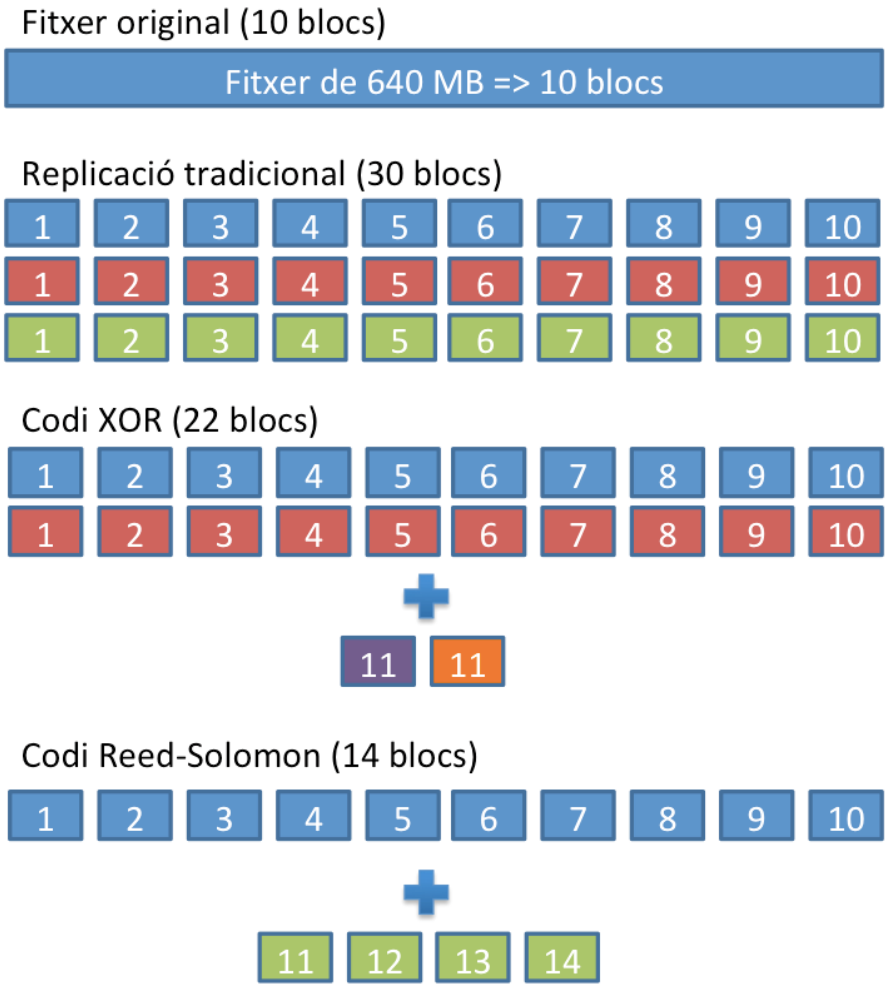

Figura 5.1: Diferents sistemes per garantir la redundància. Representació de la quantitat de blocs de dades i paritat que s'emmagatzemen. Elaboració pròpia.

De cara a la modificació que es planteja fer en aquest projecte, un dels grans handicaps que s'ha trobat en el codi de l'HDFS RAID és que és un codi complex i els programadors no han afegit cap tipus de comentaris per fer mes entenedor el codi. Tampoc existeix documentació oficial per part de Facebook sobre l'estructura de classes ni les funcions que contenen. A més, degut a l'escassetat de documentació i a que és un projecte relativament recent, l'experiència per part de la comunitat global d'usuaris és gairebé nul·la.

5.2 Arquitectura HDFS RAID

El mòdul HDFS RAID ofereix un sistema de fitxers anomenat DistributedRaidFileSystem (DRFS) que és utilitzat conjuntament amb una instància del DistributedFileSystem de Hadoop. Els diferents components de l'arquitectura s'encarreguen de generar la paritat pels fitxers emmagatzemats al DRFS. Els fitxers es divideixen en *stripes* formats per diversos blocs. Cada un dels fitxers dins el DRFS té associat un fitxer de paritat, que conté els blocs amb la redundància.

Com veurem més endavant, l'arquitectura de l'HDFS RAID modifica el flux de comunicació entre l'aplicació i el sistema de fitxers de Hadoop, ja que substitueix el client natiu de Hadoop per un de propi, el qual estableix comunicació amb el

DistributedFileSystem.

A continuació, repassarem els components més destacats de l'HDFS RAID mitjançant un esquema i, posteriorment, analitzarem cada un d'ells amb més detall.

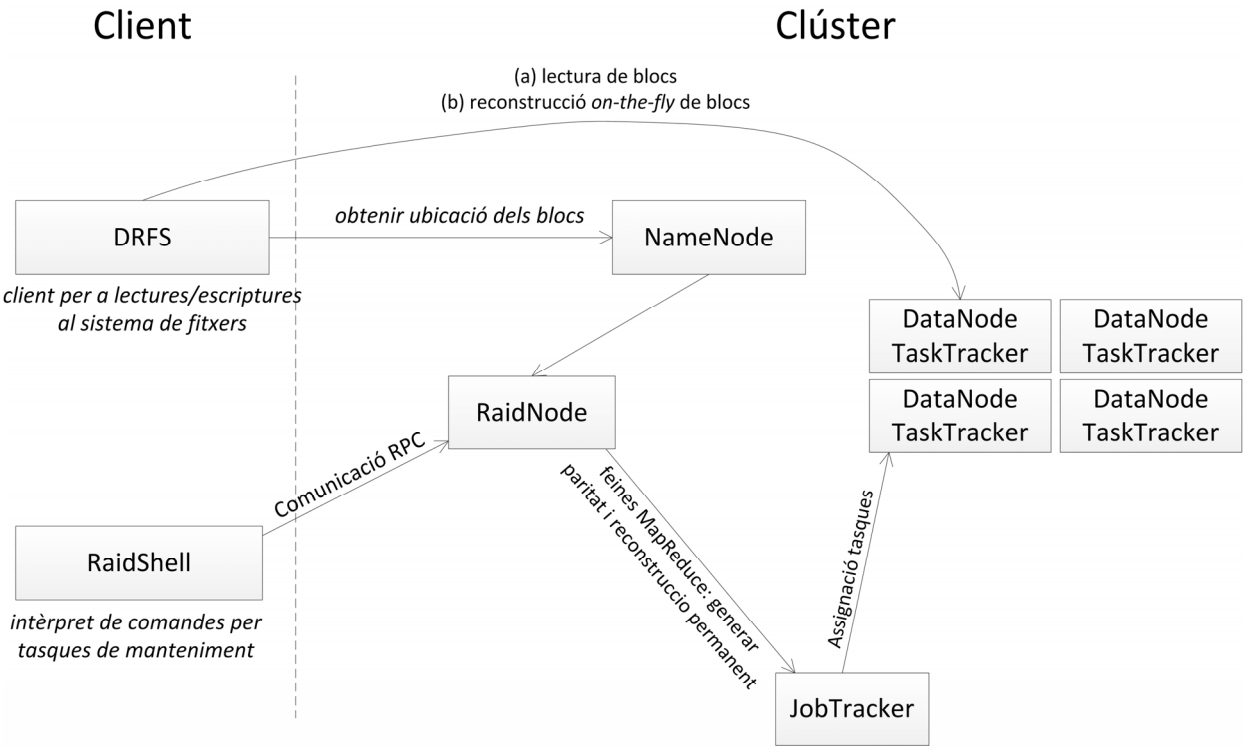

Figura 5.2: Diagrama de l'HDFS RAID. Elaboració pròpia a partir d'una adaptació de [27].

Server-Side

- RaidNode: és una instància de software que s'encarrega de crear i mantenir els fitxers de paritat per totes les dades emmagatzemades dins el DRFS.
- BlockFixer: regenera els blocs amb pèrdua o corrupció de dades.
- Block Placement Policy: s'encarrega de distribuir els blocs de forma que es garanteixi la màxima tolerància a fallades.
- ErasureCode: és la implementació que realitza la codificació i la descodificació de bytes en els blocs emmagatzemats al DRFS.

Client-Side

- El client DRFS: permet a l'aplicació accedir als fitxers emmagatzemats dins el DRFS i recupera, de forma transparent, els blocs amb pèrdua o corrupció de dades trobats durant la lectura d'un fitxer.
- La utilitat RaidShell: permet a l'administrador activar manualment la regeneració de blocs amb pèrdua o corrupció de dades així com el monitoratge de fitxers que han esdevingut irrecuperables.

5.2.1 RaidNode

El RaidNode escaneja periòdicament totes les rutes que s'han configurat per ser emmagatzemades dins el DRFS. Per cada ruta, inspecciona de forma recursiva tots els fitxers i selecciona aquells que tenen més de 2 blocs i no han estat modificats recentment (per defecte, el llindar és de 24 hores).

Els fitxers candidats per ser emmagatzemats al DRFS són processats pel RaidNode, el qual crea el nombre apropiat de blocs de paritat per cada *stripe*. Una vegada s'han creat els blocs de paritat, aquests són concatenats tots junts i s'emmagatzemen en un fitxer de paritat associat amb el fitxer original. Una vegada s'ha creat el fitxer de paritat, el factor de replicació del corresponent fitxer es disminueix fins al nivell fixat en el fitxer de configuració.

El RaidNode també s'encarrega de forma periòdica, de fer la purga dels fitxers de paritat antics, en cas que el fitxer de dades originals sigui esborrat.

La generació de fitxers de paritat es du a terme, bé amb un sol procés, o bé programant una tasca MapReduce. De fet, existeixen dues implementacions del RaidNode:

El LocalRaidNode, que computa els blocs de paritat de forma local dins el RaidNode. Donat que la generació de blocs de paritat és un procés que exigeix molt temps de còmput, la escalabilitat d'aquest enfocament és limitada

El DistributedRaidNode, que programa tasques MapReduce per fer la generació dels blocs de paritat.

5.2.2 BlockFixer

El BlockFixer és una instància de software que s'executa al RaidNode i periòdicament inspecciona la "salut" de les rutes configurades pel DRFS. Quan es troba un fitxer amb blocs que tenen pèrdua o corrupció de dades, aquests blocs són reconstruïts i es graven de forma persistent dins el sistema de fitxers.

Per fer-ho, el BlockFixer sol·licita una llista de blocs corruptes al NameNode. Si el bloc corrupte correspon a un bloc de dades, es reconstrueix amb el procés de descodificació. Quan el bloc corrupte és un bloc de paritat, es reconstrueix codificant.

De forma similar al RaidNode, existeixen dos implementacions del BlockFixer, una que reconstrueix els blocs de forma local (LocalBlockFixer) i una altra que programa tasques MapReduce per reconstruir els blocs de forma distribuïda (DistBlockFixer)

5.2.3 Mòdul de Block Placement Policy

La utilització dels codis correctors planteja un problema: es crea una dependència entre els blocs de les dades originals i els blocs de paritat. La distribució dels blocs per defecte no és favorable per garantir una bona redundància de dades, ja que els blocs de dades o de paritat podrien quedar emmagatzemats en el mateix node, o en el mateix *rack*, reduint consegüentment la capacitat de tolerància a fallades del codi.

El mòdul de BlockPlacementPolicy s'encarrega de distribuir els blocs de tal forma que

cada un dels blocs d'un *stripe* s'emmagatzema en un node diferent [28].

5.2.4 ErasureCode

La implementació dels codis correctors d'errors dins l'arquitectura de l'HDFS RAID s'articula mitjançant la classe ErasureCode.

L'ErasureCode és el component subjacent utilitzat pel BlockFixer i el RaidNode per generar els fitxers de paritat i per arreglar els blocs amb errors. L'ErasureCode és una classe abstracta que implementa els mètodes genèrics de codificació i descodificació.

Quan cal fer la codificació de les dades, l'ErasureCode agafa diversos bytes de dades i genera alguns bytes de paritat. Quan es fa la descodificació, l'ErasureCode genera els bytes perduts, ja siguin de dades o de paritat, utilitzant els blocs restants.

Actualment, hi ha dos tipus de codis correctors d'errors implementats a l'HDFS RAID. Aquests son l'XOR i el Reed-Solomon. La diferència entre ells és que l'XOR només permet crear un byte de paritat mentre que el Reed-Solomon permet especificar quina quantitat de bytes de paritat s'han de crear.

La classe ErasureCode és una classe abstracta que fa possible afegir implementacions de codis correctors d'errors diferents, a banda dels que ja estan implementats.

5.2.5 Client DistributedRaidFileSystem (DRFS)

El client DistributedRaidFileSystem (DRFS) està implementat com una capa que funciona per sobre del DistributedFileSystem (DFS), el sistema de fitxers natiu de Hadoop. El DRFS intercepta i processa totes les crides fetes al DFS i les passa al client subjacent.

El DFS llança una excepció ChecksumException o BlockMissingException quan, durant la lectura d'un fitxer, detecta que s'ha produït alguna pèrdua o corrupció dels blocs. El DRFS captura aquestes excepcions, localitza el fitxer de paritat associat amb el fitxer original i reconstrueix els blocs amb errors abans d'entregar les dades a l'aplicació.

Cal tenir en compte que tot i que el DRFS reconstrueix els blocs amb errors durant la lectura d'un fitxer, aquests blocs no són arreglats en el sistema de fitxers. Per contra, els blocs reconstruits durant la lectura del fitxer, són descartats una vegada que s'ha satisfet la petició de l'aplicació. Cal utilitzar el BlockFixer o el RaidShell per arreglar de forma persistent els blocs amb errors.

5.2.6 Utilitat RaidShell

La utilitat RaidShell permet a l'administrador fer el manteniment i inspeccionar el DRFS. Suporta comandes per activar manualment la regeneració de blocs amb errors i també permet a l'administrador visualitzar una llista de fitxers irrecuperables (degut a que s'han perdut massa blocs de dades o de paritat).

5.3 Requisits de l'aplicació

5.3.1 Requisits funcionals

- El programa ha d'implementar una funció de codificació de dades que segueixi l'algorisme de codificació del codi corrector d'errors QCFMSR (veure apartat "Procés de codificació" dels codis QCFMSR a la pàgina 34).
- El programa ha d'implementar una funció de descodificació de dades que segueixi l'algorisme de descodificació del codi corrector d'errors QCFMSR (veure apartat "Reconstrucció de dades" a la pàgina 36 i "Exemple d'aplicació dels codis QCFMSR" a la pàgina 36).
- El procediment de descodificació ha d'estar programat de forma que es faci de forma paral·lela, amb un factor de processament paral·lel que sigui parametritzable des dels fitxers de configuració de l'HDFS RAID.

5.3.2 Requisits no funcionals

- Cal fer un exhaustiu joc de proves que ha d'incloure proves sobre dades aleatòries, amb errors generats aleatòriament i seguint tot el circuit: generar dades aleatòries, codificar-les i, tot seguit, descodificar-les, i comparar el resultat amb les dades inicials.
- Per treure profit de la capacitat de fer la descodificació de forma paral·lela, el processador del node que realitza la descodificació ha de tenir dos o més nuclis.

Capítol 6: Disseny de l'aplicació

En el capítol de Disseny de l'aplicació presentem informació específica sobre el desenvolupament que hem portat a terme. En les properes pàgines el lector trobarà diversos apartats amb el contingut següent:

- Recapitularem breument sobre el propòsit de la implementació bastant-nos en tot el contingut que hem estudiat fins ara.
- Analitzarem les característiques de les classes abstractes de l'HDFS RAID, les quals ens marcaran el disseny a seguir en la implementació.
- Presentarem un diagrama de classes de la implementació.

El codi produït en aquest projecte es pot trobar en el material annex a aquesta memòria.

6.1 El propòsit de la implementació: connectant totes les peces.

Fins ara hem vist que Hadoop és un software que ofereix un servei d'emmagatzematge distribuït. Hadoop manté la tolerància a fallades fent us de la replicació, cosa que el fa molt confiable. No obstant, un dels principals inconvenients que té la replicació és que ocupa molt espai d'emmagatzematge i això genera uns costos molt alts, sobretot en clústers grans.

Aquest handicap és ben conegut i existeix multitud de recerca enfocada a pal·liar-lo. La tècnica dels codis correctors d'errors és una de les solucions que existeixen, ja que millora l'eficiència respecte a la replicació en termes de cost d'emmagatzematge. Com hem apuntat en capítols anteriors, els codis correctors d'errors permeten mantenir o incrementar el nivell de tolerància a fallades disminuint l'espai ocupat per la redundància.

La millora que suposa la utilització dels codis correctors d'errors és tan significativa que ha portat a grans empreses tecnològiques a implementar-los en els seus sistemes d'emmagatzematge. Recentment, Facebook, la xarxa social líder i la empresa que opera el clúster de Hadoop més gran (>100PB), ha fet pública una implementació dels codis correctors d'errors XOR i Reed-Solomon a Hadoop, anomenada HDFS RAID.

És cert que aquests codis permeten optimitzar l'espai d'emmagatzematge ocupat, però tenen un impacte important sobre l'ús de la xarxa que es converteix en un problema[16]. Resulta que per tal de recuperar unes dades perdudes necessiten transferir per la xarxa una quantitat molt més elevada de dades que la replicació, i donat que en clústers grans la probabilitat de fallada d'algun component és alta, aquestes transferències són molt freqüents.

Així que per una banda els codis correctors d'errors ajuden a estalviar costos en emmagatzematge, però d'altra banda, provoquen un augment en el consum de la xarxa. Això obliga a fer una despesa per augmentar els recursos de connectivitat a fi d'evitar una degradació del nivell de servei ocasionat pels efectes de la saturació de la xarxa. A més, anul·la la bona escalabilitat de Hadoop.

[16] Veure problema de la reparació (Pàg. 32).

Per tant, l'objectiu és trobar una solució per la tolerància a fallades que tingui una bona eficiència en el cost d'emmagatzematge i també en la quantitat d'informació necessària per recuperar les dades. És aquí on entren en joc els *Quasi-cyclic Flexible minimum storage regenerating codes* (QCFMSR).

Els codis QCFMSR són una nova família de codis regeneratius [19], que busquen un equilibri òptim entre l'ample de banda necessari per reparar un node avariat i la quantitat de redundància emmagatzemada. Aquest equilibri permet atenuar el problema de la reparació.

En aquest projecte implementarem un codi corrector d'errors quasi-ciclic QCFMSR amb paràmetres $[10, 5, 6]$ sobre $\mathbb{F}_{256^2}$. Els coeficients del codi sobre $\mathbb{F}_{256}$ han estat calculats[17] per tal de maximitzar el rang i són $\{1,1,2,8,1\}$. La implementació la farem aprofitant la estructura que ens ofereix l'HDFS RAID de Facebook, que ja hem estudiat prèviament. A [29] i [30], el lector podrà trobar el detall d'una altra implementació de codis LRC sobre HDFS RAID.

Cal dir que treballarem sobre la versió "production" de l'HDFS RAID, que és la versió considerada "estable". No tenim control sobre quan les versions "beta" passen a "production" així que hem descarregat la versió "production" vigent per evitar sorpreses i canvis inesperats en l'estructura del codi.

6.2 Anàlisi de les classes del HDFS RAID involucrades en el desenvolupament

A continuació examinarem i analitzarem les classes abstractes definides per l'HDFS RAID que tenen relació amb la implementació d'un codi corrector d'errors. Aquestes són les classes que caldrà implementar en el nostre projecte. Com ja hem dit anteriorment, el codi font d'aquestes classes abstractes es pot consultar a través del repositori de Hadoop de Facebook (https://github.com/facebook/hadoop-20).

En el material annex a aquesta memòria s'inclou un fitxer comprimit amb la versió "production" vigent el Juny de 2013. La ruta dins el fitxer comprimit on està ubicat tot el codi de l'HDFS RAID és:

hadoop-20-production\src\contrib\raid\src\java\org\apache\hadoop\raid

Així mateix, el codi implementat també es troba al material annex.

Les classes que examinarem estan dins aquesta ruta i són:

- Classe ErasureCode.java: aquesta classe conté els mètodes de codificació i descodificació propis del codi corrector d'errors implementat.
- Classe Encoder.java: prepararà les dades per tal que siguin codificades i farà una crida a la funció encode() de la classe ErasureCode.
- Classe Decoder.java: prepararà les dades per tal que siguin descodificades i farà una crida a la funció decode() de la classe ErasureCode.

[17] No mostrem el procés de construcció d'aquest codi ja que això no entra en l'objectiu del present treball.

6.2.1 Classe ErasureCode.java

La classe ErasureCode és una classe abstracta que defineix la interfície a utilitzar en la implementació del codi corrector d'errors. Aquí es defineixen les funcions encode i decode.

6.2.1.1 Funció encode

El mètode encode és pròpiament el que codifica les dades i genera la paritat. Aquí cal implementar l'algorisme propi del codi corrector d'errors que estiguem utilitzant.

Capçalera: `public void Encode (int[] message, int[] parity)`
Paràmetres d'entrada

- **`int[] message`**: És un array que conté les dades del missatge que cal codificar. Cada posició de l'array conté un símbol (un byte) de dades de cada un dels blocs a codificar. La mida d'aquest array serà igual al nombre de blocs de dades que cal agafar per generar la paritat. Les dades estan presents en els bits menys significatius de cada enter. El nombre de bits que ocupa cada símbol el podem obtenir amb symbolSize(). El número d'elements de cada missatge ha de ser stripeSize().

  Com que, per la funció encode, volem treballar amb el valor numèric dels bits de cada byte, serà necessari fer una conversió a enter abans de passar les dades d'entrada a Encode. Això és degut a que el tipus Byte de Java s'emmagatzema amb signe (signed byte) [31]. Aquesta conversió s'ha de fer de la forma següent:cal aplicar una màscara 0xFF amb la funció AND, i emmagatzemar el resultat en una variable de tipus enter.

Paràmetres de sortida

- **`int[] parity`**: És el paràmetre de sortida amb les dades de paritat generades per la funció de codificació. El número de bits de paritat és symbolSize(). El nombre de posicions d'aquest array ha de ser paritySize().

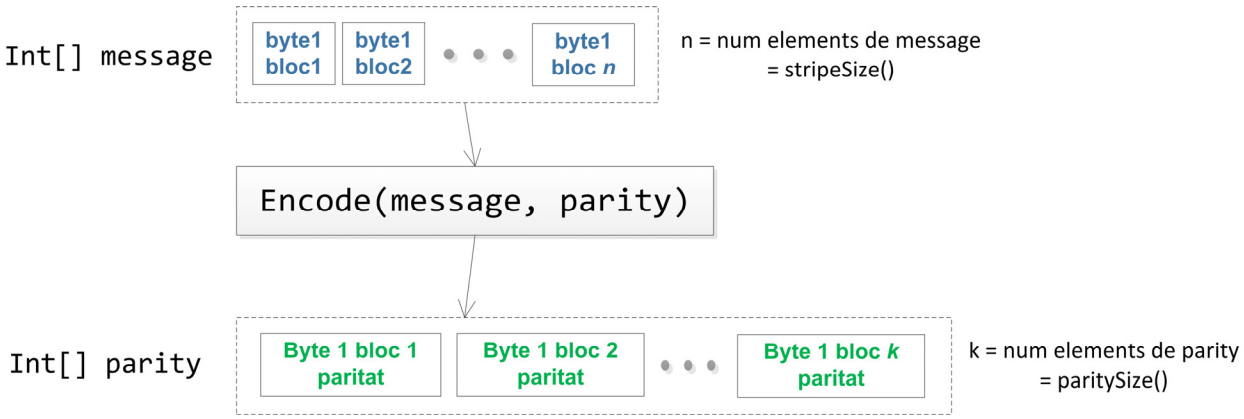

Figura 6.1: Diagrama de funcionament de la classe Encode. Elaboració pròpia.

Esquema de l'algorisme del codi QCFMSR implementat a la funció Encode

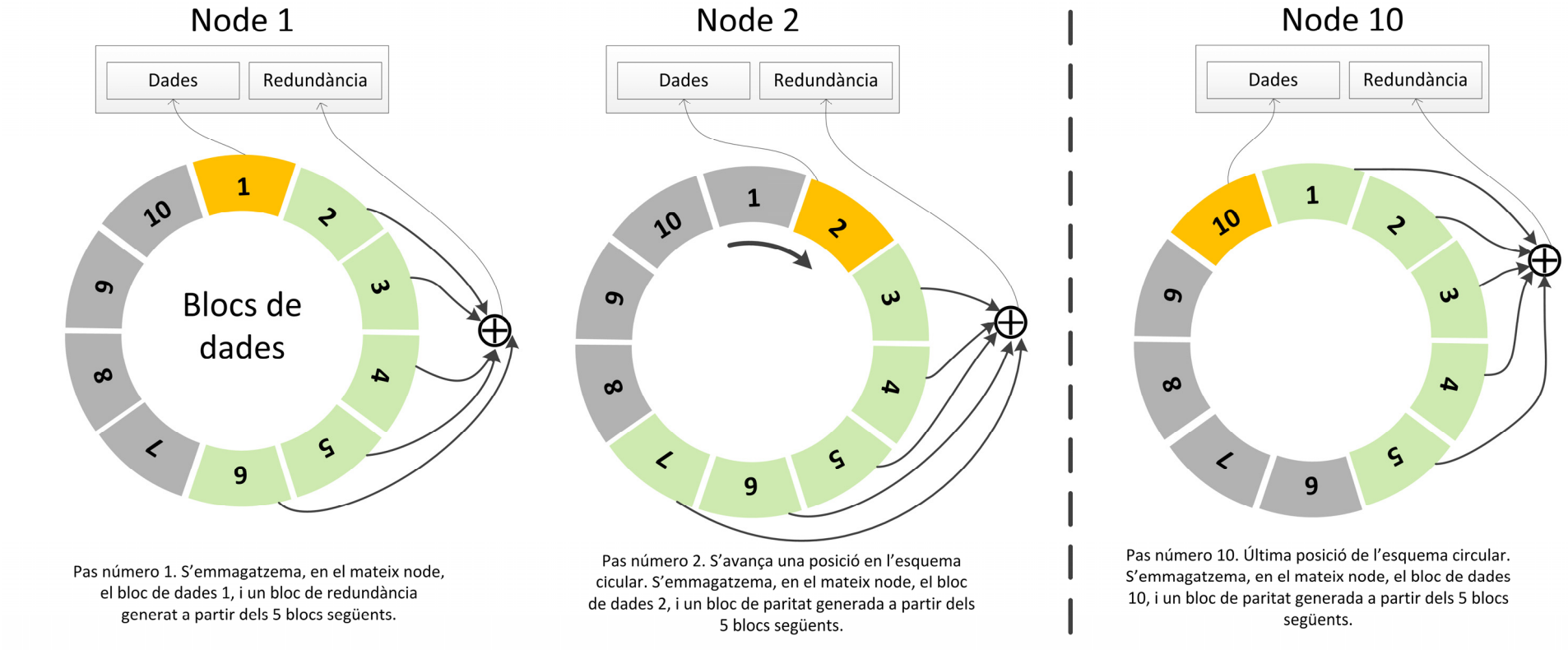

Figura 6.2: Esquema de l'algorisme del codi QCFMSR implementat a la funció Encode. Elaboració pròpia.

6.2.1.2 Funció decode

La funció decode és la funció que recupera parts de les dades perdudes.

Capçalera:

```
public void decode (int[] data, int[] erasedLocations, int[] erasedValues)
```

Paràmetres d'entrada

- int[] data: Aquest array conté les dades i la paritat. La paritat s'ha de situar en les primeres posicions de l'array. A cada un dels enters, la porció de dades rellevant està situada en els bits menys significatius. El número d'elements de l'array data és stripeSize() + paritySize()
- int[] erasedLocations: Les posicions de l'array data on se situen les dades perdudes. El nombre d'elements en aquest array serà igual a la quantitat de dades desaparegudes.

Paràmetres de sortida

- int[] erasedValues: Els valors descodificats corresponents amb erasedLocations.

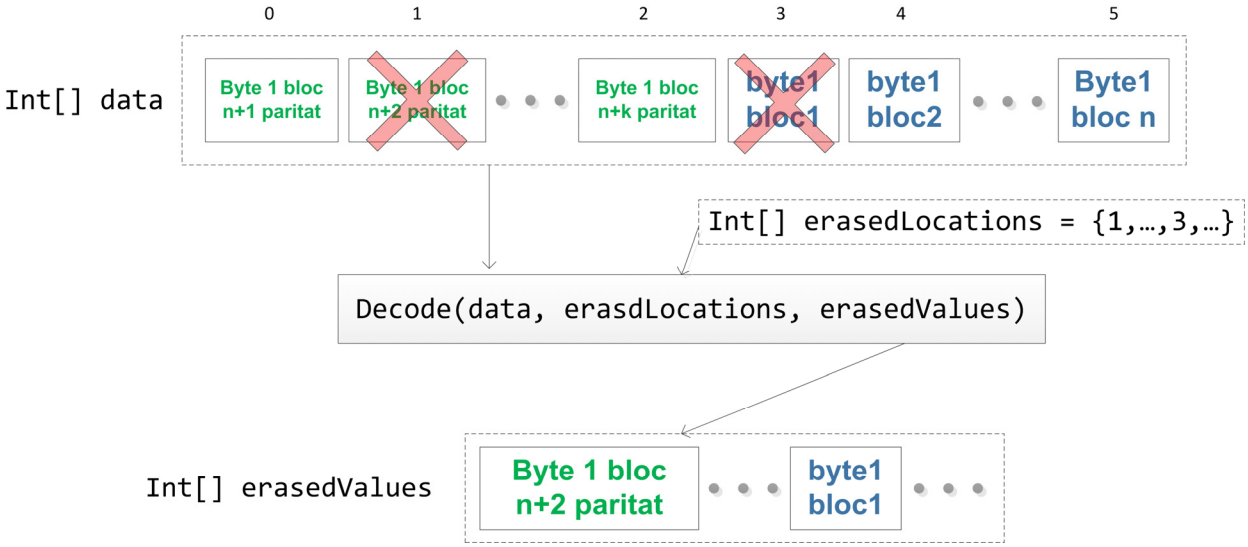

Figura 6.3: Diagrama de funcionament de la classe Decode. Elaboració pròpia.

6.2.2 Classe Encoder.java

El procés de codificació implica la transferència de les dades dels blocs a codificar cap al node que realitzarà el còmput de la paritat amb l'algorisme del codi corrector d'errors. Les operacions de lectura dels blocs, la transferència per la xarxa de les dades i el còmput de la paritat funcionen a velocitats diferents, per tant, es fa necessària la utilització de buffers.

Les dades dels blocs no es llegeixen de cop, sinó que es van llegint a mida que es van consumint les dades del buffer. Això s'articula obrint una connexió a cada un dels blocs

amb una estructura InputStream amb buffer[18]. D'aquesta operació no ens n'hem de preocupar ja que se n'ocupa la classe ParallelStreamReader que s'encarrega de llegir les dades dels InputStreams de forma paral·lela i les col·loca en un buffer anomenat readBufs.

La funció que cal implementar a la classe Encoder del nostre codi és la encodeStripeImpl.

6.2.2.1 Funció encodeStripeImpl

Des d'aquí cridarem a la funció encode de la classe ErasureCode. La funció encode anirà consumint les dades del readBufs fins que ja no en quedin. Llavors, des de la encodeStripeImpl, tornarem a cridar a ParallelStreamReader perquè ompli de nou el readBufs amb noves dades dels InputStreams. El procés de codificació finalitzarà quan s'hagin codificat totes les dades.

Capçalera:

```
protected abstract void encodeStripeImpl(
   InputStream[] blocks,
   long stripeStartOffset,
   long blockSize,
   OutputStream[] outs,
   Progressable reporter) throws IOException;
```

Paràmetres d'entrada

- InputStream[] blocks: és un array d'objectes InputStream, que representen la connexió amb els diferents blocs a codificar.
- long stripeStartOffset: indica en quina posició del fitxer (quin byte) comença el primer bloc de dades a codificar.
- long blockSize: indica la mida en bytes d'un bloc.

Paràmetres de sortida

- OutputStream[] outs: és un array d'objectes OutputStream, que representen la connexió amb els blocs de paritat que volem generar.
- Progressable reporter: permet indicar el progrés de la operació.

A continuació presentem dos esquemes per il·lustrar quin és el flux de dades en el procés de codificació.

[18] Hadoop implementa la classe FSDataInputStream per fer-ho. Veure: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/fs/FSDataInputStream.html

Visió general del procés de codificació

S'estableix una connexió amb els blocs a través d'un *InputStream*. El parallelStreamReader realitza les lectures dels diferents blocs a través dels *InputStream*. Quan s'ha llegit una determinada quantitat de dades, s'omple el *buffer* readBufs. La funció encode codifica les dades del readBufs. Quan s'han codificat totes les dades del readBufs, es torna a cridar a ParallelStreamReader perquè col·loqui més dades a readBufs. La redundància produïda per la funció encode es col·loca dins un *buffer* d'escriptura anomenat writeBufs i és escrita dins els blocs de redundància.

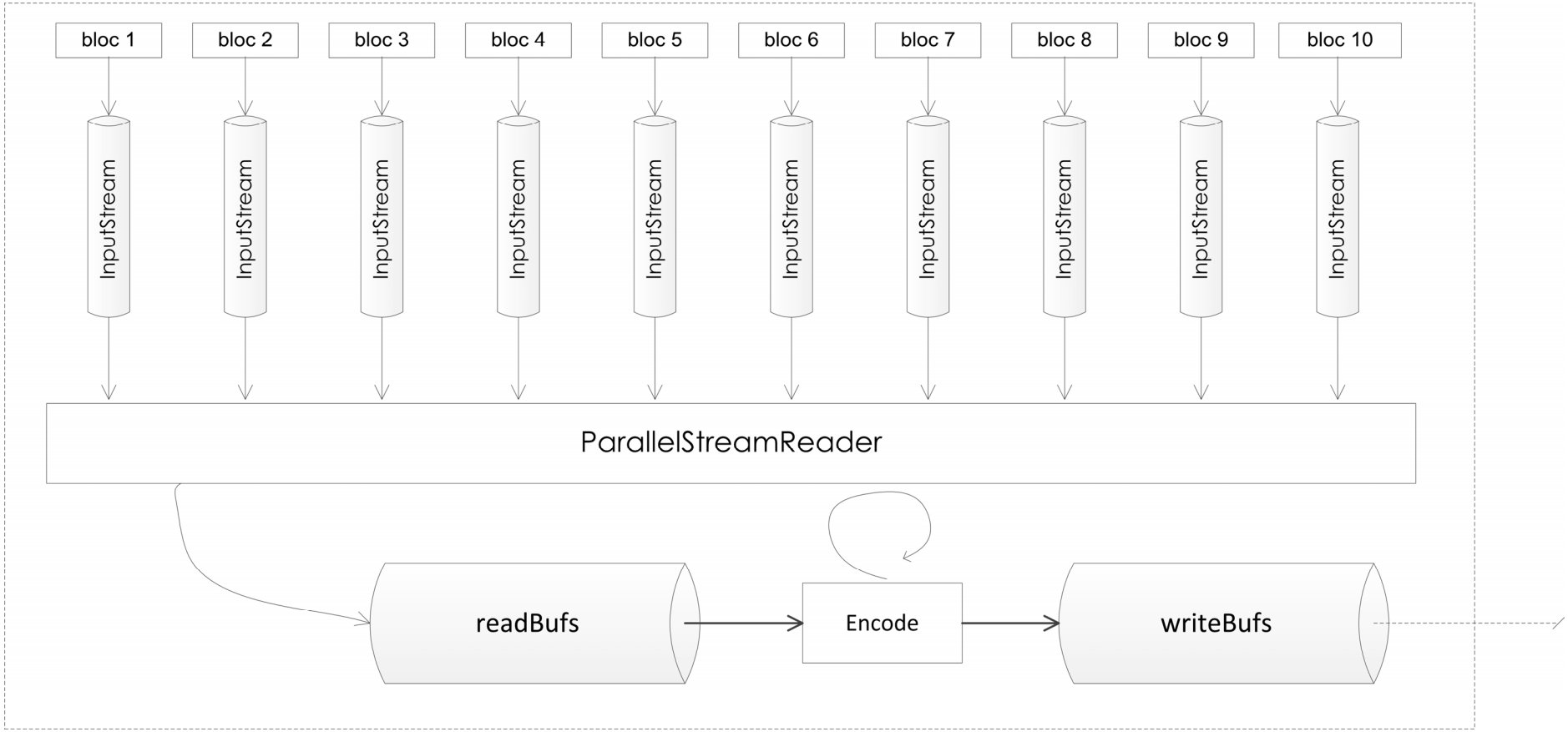

Figura 6.4: Esquema general del flux de la informació en el procés de codificació. Elaboració pròpia

Diagrama del procés de codificació a nivell de bytes i blocs

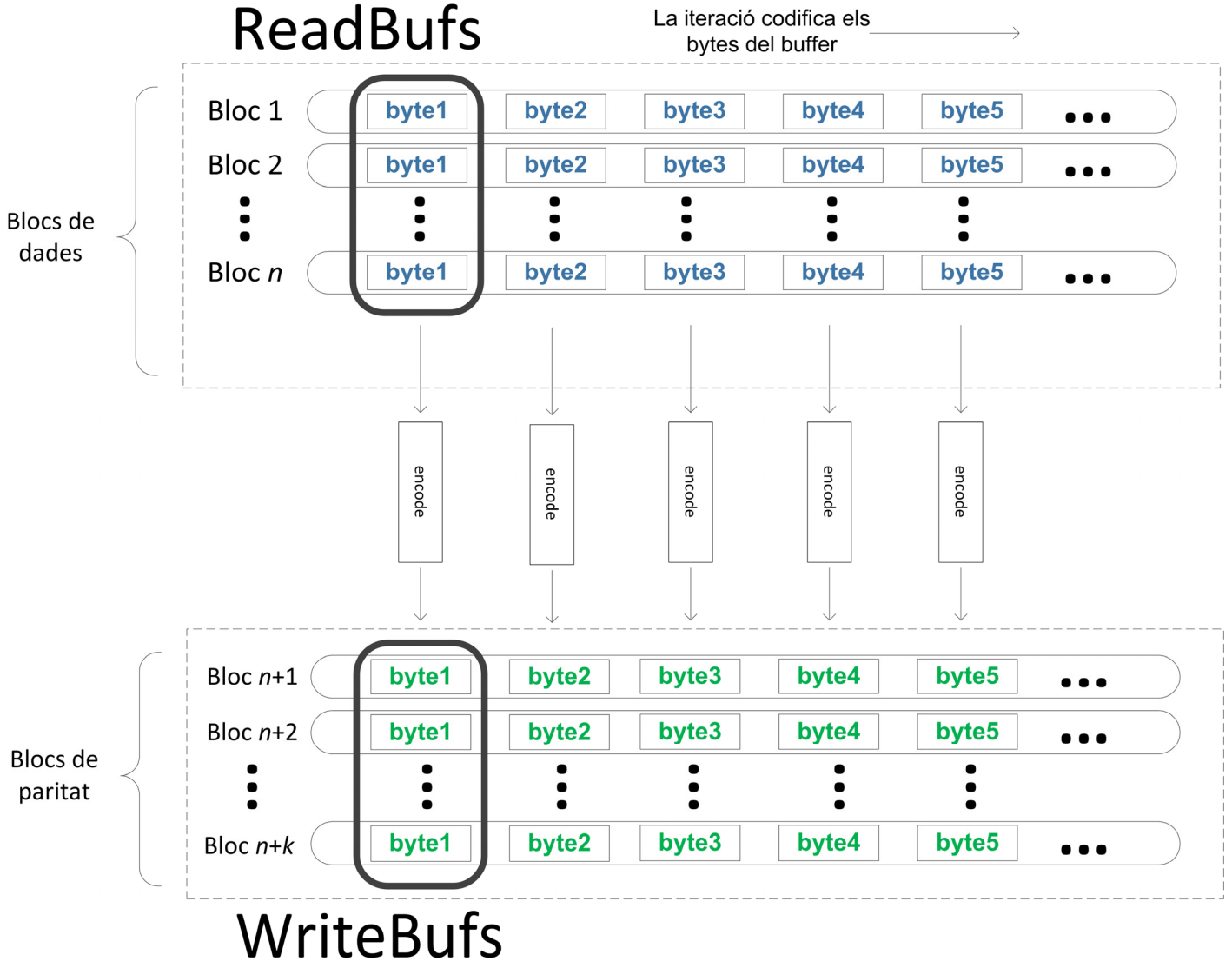

Figura 6.5: Diagrama del procés de codificació a nivell de bytes i blocs. Elaboracio pròpia.

6.2.3 Classe Decoder.java

6.2.3.1 Funció fixErasedBlockImpl

És una funció abstracta que caldrà implementar. La missió d'aquesta funció és recuperar un bloc que ha patit una pèrdua de dades. A diferència de la classe Encoder, observem que a la classe Decoder es dona la llibertat al programador perquè implementi, segons les seves necessitats, la rutina que selecciona els blocs als que vol connectar-se.

Per tant, a la implementació final caldrà implementar una funció que s'ocupi de triar quins blocs necessitem per fer la codificació, i estableixi la connexió cap a ells.

Capçalera:

```
 protected abstract void fixErasedBlockImpl(
     FileSystem fs, Path srcFile, FileSystem parityFs, Path
parityFile,
     long blockSize, long errorOffset, long limit,
     OutputStream out, Progressable reporter) throws IOException;
```

Paràmetres d'entrada

- `FileSystem fs`: és un objecte que representa el sistema de fitxers que conté el fitxer afectat per la pèrdua de dades.
- `Path srcFile`: és la ubicació dins el sistema de fitxers *fs* del fitxer afectat per la pèrdua de dades
- `FileSystem parityFs`: és un objecte que representa el sistema de fitxers que conté el fitxer de paritat associat amb el fitxer de dades. Pot ser diferent de *fs* quan el fitxer de paritat pertany a un arxiu HAR.
- `Path parityFile`: és la ubicació dins el sistema de fitxers *parityFs* del fitxer de paritat
- `long blockSize`: la mida en bytes d'un bloc de dades.
- `long errorOffset`: indica la posició del fitxer (quin byte) on s'ha detectat l'error. Hi poden haver més errors que seran descoberts durant el procés de descodificació.
- long limit: és el número -màxim- de bytes que hem de recuperar.
- OutputStream out: és un objecte de sortida que utilitzarem per escriure les dades del bloc recuperat.
- Progressable reporter:permet indicar el progrés de la operació.

Al capítol següent presentarem les 3 classes que cal crear per a implementar el codi QCFMSR a l'HDFS Raid. Les classes a crear són 3: QuasiCyclicCode.java, QuasiCyclicEncoder.java, QuasiCyclicDecoder.java.

6.3 Diagrama de classes

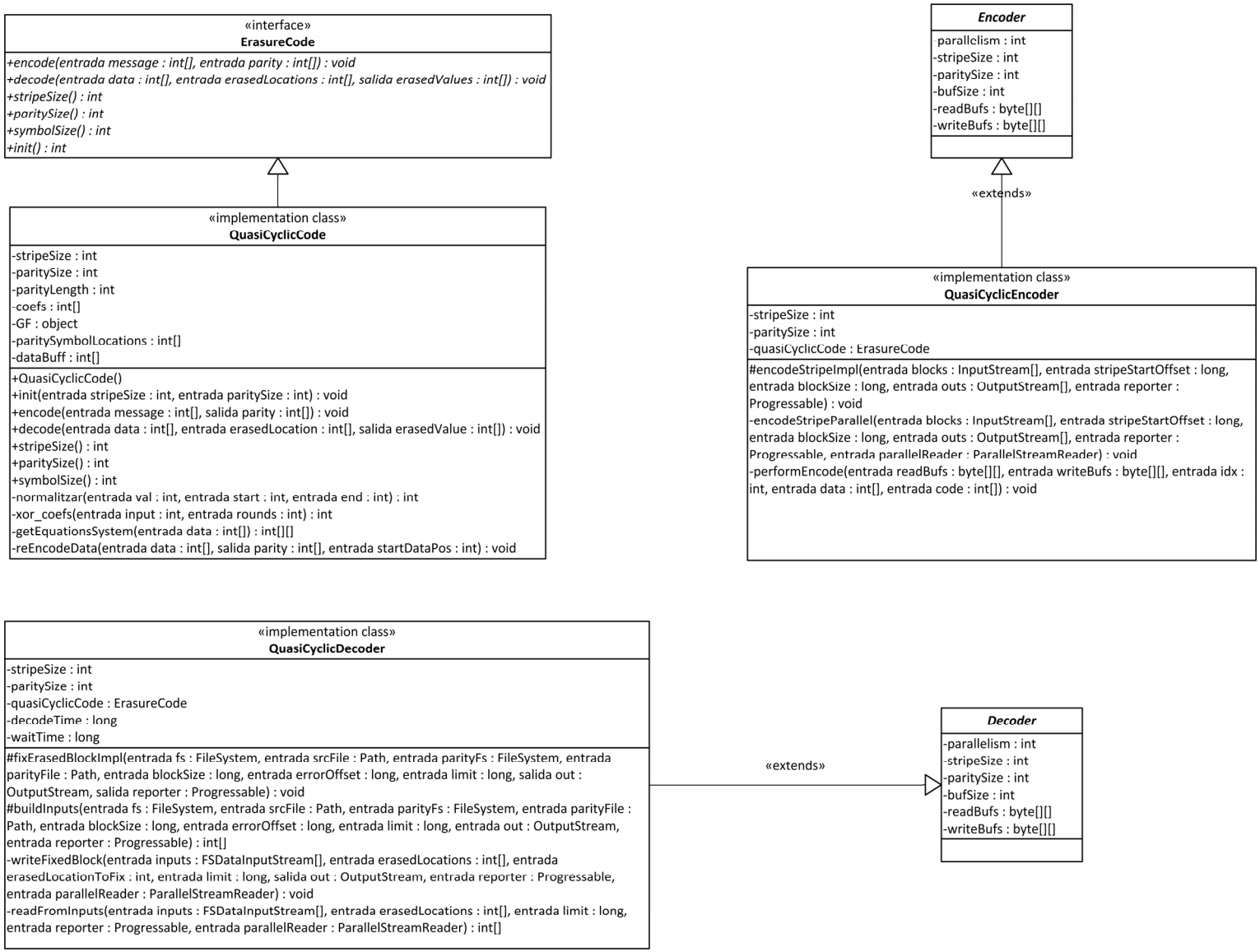

Figura 6.6: Diagrama de classes de la implementació realitzada. Elaboració pròpia.

Capítol 7: Implementació i proves

En aquest capítol enunciarem quines són les classes que hem implementat i explicarem les seves funcions. Posteriorment detallarem el disseny de les proves i en presentarem i comentarem el resultat.

7.1 Classes implementades

Les classes implementades són tres:

- QuasiCyclicCode.java
- QuasiCyclicEncoder.java
- QuasiCyclicDecoder.java

7.1.1 Classe QuasiCyclicCode.java

7.1.1.1 Funció encode

Aquesta és la funció que realitza el procés de codificació de les dades. La funció encode rep un array d'enters *message* amb les dades que cal codificar i retorna un array d'enters *parity* amb la redundància generada. Aquesta funció crida a la funció xor_coefs per formar el byte de paritat de cada grup de blocs. El funcionament d'aquesta classe ja ha estat descrit amb més detall a l'apartat 5.2.

7.1.1.2 Funció decode

Permet recuperar dades codificades amb el codi corrector d'errors QCFMSR. En cas que totes els posicions que s'han de recuperar són de paritat, la funció decode crida a la funció `reEncodeData`. Això es fa per un motiu d'eficiència, ja que el procés de codificació consumeix menys temps de procés que la descodificació.

La descodificació monta una matriu amb les dades d'entrada i obté les dades originals resolent el sistema d'equacions mitjançant Gauss-Jordan.

7.1.1.3 Funció normalitzar

La funció normalitzar converteix un enter qualsevol a un nombre dins una aritmètica modular d'unes dimensions especificades. Això es fa servir per representar l'esquema quasi-ciclic que segueix el codi QCFMSR. Aquesta funció és cridada per la funció encode a l'hora d'aplicar els coeficients del codi a les dades d'entrada.

7.1.1.4 Funció xor_coefs

Aquesta funció és cridada per la funció encode. La seva missió es aplicar els coeficients del codi QCFMSR sobre les dades que formen cada byte de redundància. Per tal de programar aquesta funció ha estat d'ajuda la font bibliogràfica [32].

7.1.1.5 Funció getEquationsSystem

La funció getEquationsSystem retorna un array bidimensional que representa la matriu el codi. Aquesta funció és cridada per la funció decode per tal de obtenir el sistema d'equacions.

7.1.1.6 Funció reEncodeData

Aquesta funció recodifica les dades de paritat en cas de pèrdua. És cridada per la funció decode per tal de preparar les dades provinents de la funció decode perquè puguin ser passades a la funció encode.

7.1.2 Classe QuasiCyclicEncoder.java

Hadoop crida aquesta classe quan s'ha de codificar un arxiu. Aquesta classe s'encarregarà de recuperar els blocs de dades a codificar, col·locarà les dades en les estructures apropiades abans de cridar a la funció encode de QuasiCyclicCode.java.

No cal que ens preocupem de la implementació de les funcions que escriuen la paritat generada als fitxers, ja que això ja està implementat a la funció genèrica Encoder.java.

Aquesta és una classe adaptada als QCFMSR a partir de la implementació del codi Reed-Solomon.

7.1.2.1 Funció encodeStripeImpl

Obre la connexió als blocs de dades que hem de codificar i crida a la funció encodeStripeParallel, que farà que la lectura dels inputs es faci de forma paral·lela.

7.1.2.2 Funció encodeStripeParallel

Llegeix de forma paral·lela els inputs de dades a codificar, i col·loca les dades dins un buffer. Quan el buffer està ple, crida a performEncode per codificar totes les dades llegides. Quan s'han consumit totes les dades del buffer, la funció reomple el buffer amb més dades llegides.

Aquesta funció paral·lelitza la lectura de les dades i la codificació. És a dir, mentre es codifiquen les dades del buffer, el programa segueix llegint els inputs d'entrada. Això augmenta l'eficiència.

7.1.2.3 Funció performEncode

La funció performEncode col·loca un byte de dades de cada bloc a codificar dins l'estructura adequada i crida a la funció Encoder de QuasiCyclicCode.java.

7.1.3 Classe QuasiCyclicDecoder.java

Hadoop crida aquesta classe quan ha trobat un esborrall en alguna part de les dades. Aquesta classe s'encarregarà de fer una sèrie d'operacions prèvies abans d'executar el que és pròpiament l'algorisme de reconstrucció de les dades.

La classe QuasiCyclicDecoder.java és una classe adaptada als QCFMSR a partir de la implementació del codi Reed-Solomon.

El procés de descodificació marcat pel model de la classe abstracta Encoder detallat al capítol anterior, es centra en la funció fixErasedBlockImpl. A la implementació real el

procés de descodificació es divideix en fases i es reparteix en diverses funcions. A continuació repassarem quines funcions són.

7.1.3.1 Funció fixErasedBlockImpl

Hadoop crida aquesta funció quan detecta un error en algun bloc. En els paràmetres d'aquesta funció es passen variables com:

- Quin és el fitxer que conté els blocs de dades (*srcFile*).
- Quin és el fitxer que conté els blocs de paritat (*parityFile*).
- Un *flag* que ens indica si l'esborrall s'ha produït en les dades o en la paritat (fixSource).
- La mida en byte dels blocs (*blockSize*).
- La posició inicial en bytes del bloc on s'han detectat errors, respecte a la posició 0 del fitxer (*errorOffset*).
- La longitud total en bytes del fitxer (*limit*).
- Un OutputStream on escriurem la sortida amb les dades reconstruïdes (*out*).

Aquesta funció segueix aquests passos:

1. Crida a la funció BuildInputs, la qual detallarem més endavant. Resumint, el que fa BuildInputs és obrir la connexió amb els blocs de dades i paritat per tal de fer la reconstrucció. La funció BuildInputs calcula a quin stripe i bloc del fitxer s'han produït els esborralls i retorna a fixErasedBlockImpl un array d'enters amb les posicions on s'han produït esborralls (*erasedLocation*) i un array d'InputStreams amb les connexions als blocs (*inputs*).
2. Crea un objecte ParallelStreamReader per fer una lectura en paral·lel dels blocs apuntats per *inputs*.
3. Crida a writeFixedBlock que seguirà amb el procés de descodificació.

7.1.3.2 Funció buildInputs

La funció buildInputs calcula el número de bloc i stripe afectat per l'error i obre una connexió amb tots els blocs de dades i paritat de l'stripe. Evidentment, el bloc que té l'error no es té en compte –es posa a 0– a l'hora de fer la reconstrucció.

Des d'aquesta funció s'inicialitza un array erasedLocations on s'afegeix el número de bloc de l'stripe que conté errors. Posteriorment, durant les lectures que es faran a la funció readFromInputs, és possible que es trobin més errors a altres blocs. Els nous blocs erronis detectats s'afegiran a posteriori a erasedLocations.

Aquesta funció retorna l'array erasedLocations i un array que conté les connexions a tots els blocs.

7.1.3.3 Funció writeFixedBlock

La funció writeFixedBlock és cridada per la funció fixErasedBlockImpl per seguir amb el procés previ a la reconstrucció.

Les dades a descodificar es troben a readBufs. La operació de descodificació es paral·lelitza creant un número determinat de threads (aquest paràmetre s'especifica en

el fitxer de configuració. La funció writeFixedBlock divideix el readBufs en funció del número de threads. Cada *thread* rep una porció de dades del readBufs i executa la funció performDecode per cada byte del readBufs.

Quan ja s'han consumit totes les dades del buffer de lectura readBufs, la funció writeFixedBlock crida a la funció readFromInputs perquè alimenti el buffer amb les dades llegides en paral·lel per l'objecte ParallelStreamReader creat inicialment a la funció fixErasedBlockImpl.

7.1.3.4 Funció readFromInputs

Utilitza l'objecte ParallelStreamReader per omplir el buffer readBufs a partir de les lectures realitzades de cada bloc. Controla si es detecten errors durant la lectura d'algun bloc, i donat el cas afegeix el número de bloc a erasedLocation.

7.1.3.5 Funció performDecode

Converteix els bytes a enters per un motiu que ja s'ha exposat al tema 5 i els col·loca en una estructura adequada per, posteriorment, cridar a la funció decode de la classe QuasiCyclicCode.java.

7.2 Proves

Amb l'objectiu de provar el funcionament de les classes implementades s'han dut a terme una sèrie de proves que detallarem en aquest apartat.

Ens hem centrat en fer tests unitaris de la classe QuasiCyclicCode.java, ja que aquesta és la que vertaderament conté la implementació dels codis QCFMSR. Totes les proves s'han realitzat amb un codi QCFMSR amb les característiques $[10, 5, 6]$ sobre $\mathbb{F}_{256^2}$, i amb coeficients $\{1,1,2,8,1\}$.

Per tal de fer els tests unitaris s'han tret les referències al *package* de Hadoop i les dependències exclusives de *logging* i instanciació de Hadoop. En el material annex a la present memòria es pot trobar un arxiu amb una versió de la classe QuasiCyclicCode.java sense referències a Hadoop, adequada per fer tests unitaris, i també es podrà trobar un classe que defineix i executa les proves.

7.3 Conjunt de proves generals de codificació

Amb l'objectiu d'avaluar la correcta codificació de la implementació realitzada, hem confeccionat el següent conjunt de proves de codificació de la classe QuasiCyclicCode.java.

- La prova elemental de compilació de la classe QuasiCyclicCode.java.
- La prova elemental d'instanciació de l'objecte QuasiCyclicCode.
- Per tal de comprovar el bon funcionament de la funció de codificació, hem generat aleatòriament un conjunt de parells dades-paritat a través d'una aplicació externa. Hem realitzat una prova del procés de codificació a partir de les dades, i s'ha comparat la paritat obtinguda amb la classe implementada amb la paritat generada per la aplicació externa.

- Per tal de comprovar el bon funcionament de la descodificació, hem realitzat una prova de descodificació sobre les dades mencionades en el punt anterior.

- Aquesta fase de proves ha permès detectar errades en la codificació que han estat resoltes. Finalment, el resultat d'aquestes proves ha estat satisfactori en tots els casos.

7.4 Conjunt de proves sobre el comportament dels codis QCFMSR

- A fi d'avaluar el comportament dels codis QCFMSR, hem dissenyat i implementat peces de software que ens han permès fer proves massives i que tot seguit procedim a definir:

- Hem elaborat un software que genera un nombre arbitrari de conjunts d'informació $v = (v_1, v_2, \dots, v_{10})$ amb v_i aleatoris sobre $\mathbb{F}_{256}$ i en genera la paraula codi corresponent.

- Hem elaborat un software que donat un conjunt de dades d'entrada produeix un nombre arbitrari d'esborralls en les dades. El número d'esborralls i les posicions que tenen esborralls es poden parametritzar definint un llindar.

- La tria de les posicions amb esborralls es fa de forma aleatòria entre un llindar $[minPos,\ maxPos]$. El programa donarà com a sortida un vector d'enters $erasedLocation = \{p_1, \dots, p_j\}$ amb els paràmetres $p_i \in [minPos, maxPos]$ i $j \in [minErrors, maxErrors]$.

- Hem elaborat un software que donat una paraula codi d'entrada i un vector amb posicions amb esborrall crida el procés de descodificació.

- Hem integrat el software de generació de conjunts d'informació aleatoris, el software de generació d'esborralls aleatoris i el software de descodificació per tal de fer proves massives. En primer lloc, el programa genera un conjunt d'informació d'entrada i el codifica, en segon lloc es genera un nombre d'esborralls sobre aquestes dades, en tercer lloc s'inicia el procés de descodificació. Quan el procés de descodificació ha finalitzat, compara la informació inicial amb el resultat obtingut a través de la descodificació i dona un veredicte en funció de si coincideixen o no. Aquest programa ens permet repetir aquestes proves el nombre de vegades que vulguem.

La confecció d'aquests programes d'ajuda ens ha permès dissenyar i executar un conjunt de 4 proves que enumerem a continuació:

1. Generar fins a 5 esborralls en qualsevol posició per comprovar que la reconstrucció funciona sempre.

2. Generar més de 10 esborralls per comprovar que no és possible reconstruir la informació.

3. Generar entre 6 i 10 esborralls per comprovar quin és el percentatge de reconstruccions.

4. Generar la sèrie de 6, 7, 8, 9 i 10 esborralls per comprovar, de forma individual,

quin és el percentatge de reconstruccions, a fi de comprovar quina és la relació entre la quantitat d'errors i el percentatge de reconstruccions satisfactòries.

7.4.1 Resultat de la prova 1

Hem generat entre 1 i 5 esborralls en qualsevol posició per comprovar que la reconstrucció funciona sempre. Aquesta prova s'ha executat 1.000.000 de vegades.

El número de reconstruccions satisfactòries ha estat de 1.000.000 (100%). Per tant, podem dir que entre 1 i 5 esborralls es recupera sempre.

7.4.2 Resultat de la prova 2

Hem generat entre 11 i 20 esborralls en qualsevol posició per comprovar que no es pot reconstruir la informació. Aquesta prova s'ha executat 1.000.000 de vegades.

El nombre de reconstruccions satisfactòries ha estat de 348 (0.0348%). En un nombre de casos de 999652 (99.9652%) no ha estat possible reconstruir la informació. Els casos de reconstrucció en aquesta prova coincideixen amb conjunts de dades que alguna o algunes posicions d'informació valen 0.

Amb aquest resultat concloem que entre 11 i 20 esborralls no es recupera gairebé mai.

7.4.3 Resultat de la prova 3

Aquesta prova consisteix en generar entre 6 i 10 esborralls en qualsevol posició per comprovar quin és el percentatge de reconstruccions. Aquesta prova s'ha executat 1.000.000 de vegades.

El nombre de reconstruccions satisfactòries ha estat de 957975 (95.7975%) i el nombre de reconstruccions fallides Num comb failed: 42025 (4.2025%)

El nombre de reconstruccions satisfactòries ha estat de 957975 (95.7975%). En un nombre de casos de 42025 (4.2025%) no ha estat possible reconstruir la informació.

7.4.4 Resultat de la prova 4

Aquesta prova consisteix en executar una sèrie de proves amb un número fix d'esborralls des de 6 fins a 10 per comprovar individualment quin és el percentatge de recuperacions per a cada número d'esborralls.

Número d'errors	Percentatge de recuperació	Número de recuperacions
6	99,97%	999.741
7	99,77%	997.688
8	98,78%	987.820
9	95,38%	953.800
10	84,93%	849.273
11	0,34%	3.390

Taula 7.1: Resultats de la sèrie de proves amb un número fix d'errors entre 6 i 11.

7.4.5 Gràfic comparatiu de resultats

A continuació presentem una taula (Taula 7.2) i una gràfica (Figura 7.1) que hem confeccionat per mostrar els resultats de totes les proves realitzades:

Número d'errors	Percentatge de recuperació	Número de recuperacions	Número d'errors	Percentatge de recuperació	Número de recuperacions
1	100,00%	1.000.000	11	0,34%	3.390
2	100,00%	1.000.000	12	0,35%	3.505
3	100,00%	1.000.000	13	0,00%	16
4	100,00%	1.000.000	14	0,00%	0
5	100,00%	1.000.000	15	0,00%	0
6	99,97%	999.741	16	0,00%	0
7	99,77%	997.688	17	0,00%	0
8	98,78%	987.820	18	0,00%	0
9	95,38%	953.800	19	0,00%	0
10	84,93%	849.273	20	0,00%	0

Taula 7.2: Percentatge de recuperació entre 1 i 20 errors.

7.4.6 Comentari dels resultats

Els resultats obtinguts a través de les proves del codi confirmen el que s'indica en les bases teòriques del codi QCFMSR:

- Des d'1 fins a 5 esborralls en les dades, sempre es pot realitzar la reconstrucció.
- Entre 6 i 10 esborralls, es pot reconstruir la informació en la majoria dels casos. No obstant, el percentatge de reconstrucció disminueix a mida que augmenta el nombre d'esborralls.
- A partir de 11 esborralls, no és possible reconstruir la informació. En aquest punt, la prova ha donat un resultat diferent a la teoria, ja que en un petit percentatge de casos s'ha aconseguit reconstruir la informació tenint entre 11 i 13 esborralls. A partir de 14 esborralls, mai s'ha reconstruït la informació.

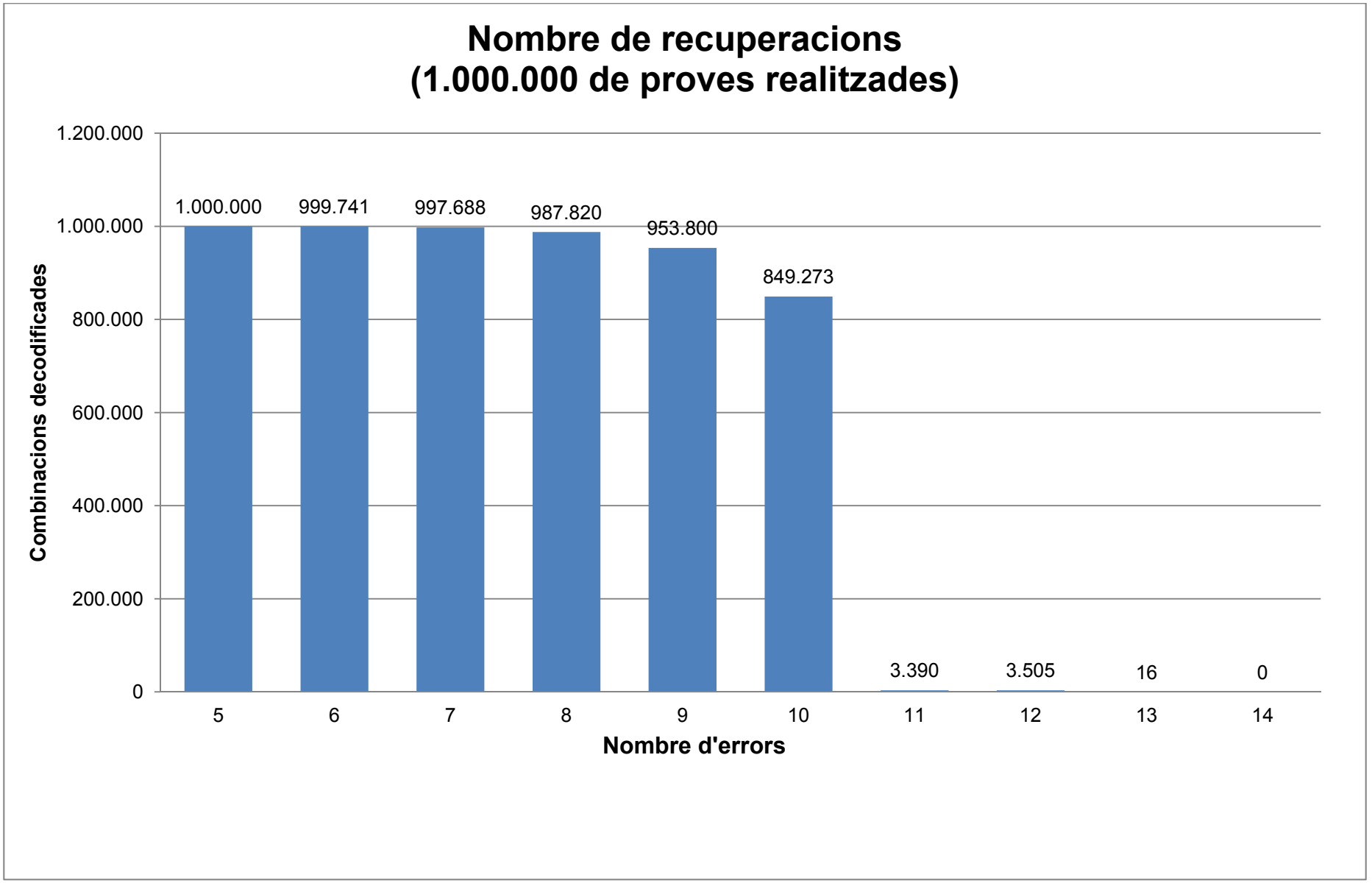

Figura 7.1: Gràfic del percentatge de recuperacions entre 5 i 14 errors.

Capítol 8: Conclusions

En aquest capítol procedim a concloure el present treball en base a un conjunt d'aspectes. En primer lloc, repassarem els objectius marcats inicialment i comentarem els resultats que hem extret fruit del procés de recerca, implementació i proves dutes a terme per analitzar el seu compliment. En segon lloc, exposarem alguns problemes ui solucions succeïdes en el transcurs de la recerca. En tercer lloc, proposarem unes línies de futur i, per acabar, farem una valoració personal del projecte.

8.1 Assoliment dels objectius

Els objectius definits en el pla de viabilitat inicial han estat revisats en les reunions de tutoria que s'han produït durant el transcurs de la recerca. Per tant, en aquest apartat veurem que el treball dut a terme no correspon, en alguns casos, al que inicialment s'havia planificat. Tanmateix, afirmem que tot els objectius redefinits en les reunions s'han complert d'una forma total.

La realitat és que en projectes de recerca i desenvolupament tot sovint sorgeixen situacions o canvis originats per diverses causes que no estaven previstos en el pla inicial. És per això que en l'àmbit de la ciència i l'enginyeria és important desenvolupar la capacitat d'adaptar-se i de reconduir el treball davant situacions inesperades per tal d'obtenir un resultats. satisfactoris per totes les parts interessades.

Tot seguit comentarem el grau d'assoliment dels objectius definits inicialment:

- **Analitzar el funcionament de Hadoop**: hem estudiat en profunditat el software Hadoop donant resposta a qüestions com: què és, quines característiques té, qui el programa, quin és el seu origen i la seva evolució, quines utilitats i usuaris té. Hem identificat els dos components cabdals de Hadoop: el MapReduce i l'HDFS, que posteriorment hem analitzat en profunditat. Per últim hem analitzat l'arquitectura organitzada en rols que presenta Hadoop.

  Per tant podem concloure que el grau d'assoliment d'aquest objectiu ha estat complet.

- **Aprofundir en el sistema d'emmagatzematge distribuït HDFS**: hem estudiat detalladament què es l'HDFS; quines característiques té, destacant-ne les més importants; quina és la seva arquitectura. Hem introduït els conceptes de bloc; *commodity-hardware*, NameNode i DataNode. Hem descrit el funcionament de les operacions de lectura i escriptura del sistema de fitxers. Finalment hem explicat el funcionament del sistema de durabilitat que implementa Hadoop: la replicació.

  Aquest estudi ens ha permès arribar a assolir aquest objectiu.

- **Proposar una alternativa al sistema de durabilitat**: per la consecució d'aquest objectiu, en primer lloc hem identificat i quantificat quin inconvenient presenta el sistema de durabilitat. Seguidament hem fet una introducció a la tècnica dels codis correctors d'errors aplicada a l'àmbit dels sistemes d'emmagatzematge distribuït, on n'hem destacat els seus avantatges. Posteriorment hem presentat les bases teòriques d'aquesta tècnica. Això ens ha permès identificar una desavantatge dels

codis correctors d'errors, que hem senyalat a l'apartat de "Problema de la reparació".

Hem introduït els codis regeneratius, que busquen donar una solució al Problema de la reparació i posteriorment hem donat les bases teòriques dels codis QCFMSR, una nova família de codis regeneratius.

Podem dir que hem assolit l'objectiu plantejat.

- **Cercar secció del codi font que conté l'algorisme de les replicacions**: Una de les descobertes fetes durant la recerca del present projecte és que ja existeix una implementació dels codis correctors d'errors dins de Hadoop, produïda per Facebook i anomenada "HDFS RAID".

  En aquest punt del projecte ens vam replantejar els objectius, ja que no aportava res de novedós fer una implementació del que ja estava fet. A més, amb el temps del que disposàvem, era difícil arribar a un software amb unes característiques millors que l'HDFS RAID.

  La prospectiva inicial va revelar que l'HDFS RAID només implementa els codis XOR i Reed-Solomon, però no codis regeneratius. Per aquest motiu, vam plantejar-nos la possibilitat d'implementar el codi regeneratiu QCFMSR a l'HDFS RAID. Aquesta era només una hipòtesi de treball, la viabilitat de la qual calia confirmar a través d'un estudi profund del codi font ja que no coneixíem, a priori, quina era la complexitat del codi ni si estava preparat per admetre més implementacions d'altres codis correctors d'errors.

  El capítol 5 és el detall d'aquest estudi que confirma la viabilitat de la implementació plantejada i estudia a fons el funcionament de les classes del codi implicades en el procés d'ampliar el programa amb un nou codi corrector d'errors.

  Aquest objectiu no s'ha dut a terme degut al replantejament de la feina, per contra hem definit l'objectiu de "Cercar les classes de l'HDFS RAID on s'implementen els codis correctors d'errors" que si que ha estat assolit.

- **Implementar el sistema de redundància mitjançant codis correctors d'errors**: aquest objectiu no ha estat dut a terme degut al replantejament del treball.

- **Implementar el sistema de redundància mitjançant codis regeneratius**: hem confeccionat les classes QuasiCyclicCode.java, QuasiCyclicEncoder.java i QuasiCyclicDecoder.java, on s'implementa el codi QCFMSR adaptat a l'estructura de l'HDFS RAID. El capítol 5 senta les bases per aquesta implementació, i el capítol 6 en dona els detalls més específics.

  Aquest objectiu ha estat assolit.

- **Fer *benchmark* de les dues implementacions**: s'ha dissenyat i executat un conjunt de proves unitàries per comprovar el bon funcionament dels processos de codificació i descodificació implementats amb els codis QCFMSR, així com per comparar el comportament del codi a la pràctica amb el que especifica la teoria.

Aquest objectiu ha estat assolit i els resultats confirmen que el comportament de la implementació està d'acord amb el que especifiquen les bases teòriques.

8.2 Desenvolupament del projecte

En aquest apartat repassarem algunes qüestions sobre el desenvolupament del projecte que ens han semblat interessants de comentar:

- Una de les dificultats significatives que hem hagut d'afrontar de cara a aquest projecte és l'aprenentatge de les bases teòriques sobre codis correctors d'errors i cossos finits. Això ens ha suposat un repte degut a que l'assignatura de Teoria de la codificació no està present al currículum de l'Enginyeria Tècnica en Informàtica de Sistemes, i el nostre coneixement sobre la matèria era nul.

- La inspecció i estudi de l'HDFS RAID ha estat una tasca de recerca molt dificultosa. El codi de l'HDFS RAID és complex i els programadors no han afegit cap tipus de comentaris per fer-lo mes entenedor. Tampoc existeix cap mena de documentació oficial per part de Facebook sobre l'estructura de classes ni les funcions que contenen. A més, degut a l'escassetat de documentació i a que és un projecte relativament recent, l'experiència per part de la comunitat global d'usuaris és gairebé nul·la.

- Durant la fase de la implementació vam detectar un error, aliè a la nostra implementació, en la funció que aplica mètode de Gauss-Jordan, dins la classe de cossos finits a l'HDFS RAID. Hem solucionat aquest *bug* i adjuntem la classe GaloisField.java corregida en el material annex a aquesta memòria.

- Quant a la planificació temporal cal dir que no s'ha complert del tot degut a que la planificació inicial contemplava uns objectius que no són els mateixos que s'han acabat portant a terme. A més, la part de recerca que es plantejava en aquest projecte era difícil de quantificar en un inici, ja que no es podia estimar a priori el nombre d'hores que requeriria arribar a uns resultats. No obstant, tot i aquestes dificultats, el projecte ha estat finalitzat a temps.

- Alguns dels diagrames presentats en la memòria són d'elaboració pròpia i han estat una eina útil, no només per comunicar i complementar el text escrit dels capítols, sinó també com a recurs d'aprenentatge per comprendre i clarificar els processos.

8.3 Línies de futur

- Per a aconseguir una implementació òptima dels codis QCFMSR dins de Hadoop, cal que cada bloc d'informació s'emmagatzemi en el mateix medi d'emmagatzematge que el bloc de redundància corresponent. És a dir, que el parell bloc-paritat s'haurien d'emmagatzemar en el mateix disc dur. No obstant, Hadoop no permet controlar a quin disc dur físic s'emmagatzemen els blocs. A més, l'HDFS RAID està dissenyat per emmagatzemar blocs de dades i blocs de paritat en nodes diferents, per tal de garantir la màxima tolerància a fallades en

codis Reed-Solomon. Caldria estudiar com implementar una política de distribució de blocs específica per als arxius codificats amb el codi QCFMSR. Veure [28]

- Es podrien estudiar millores en l'eficiència del procés de descodificació fent-lo selectiu a l'hora de recuperar les dades. Si es produeix un esborrall, és possible que amb 10 equacions puguem resoldre el sistema d'equacions, no cal que agafem les dades de tots els blocs. Per fer això caldria modificar la funció buildInputs de la classe QuasiCyclicDecoder.java.

- La implementació dels codis QCFMSR realitzada en el present projecte no ha estat provada en una instal·lació de Hadoop, tot i que s'ha deixat el codi preparat per a fer-ho. Caldria fer proves del rendiment que donen els codis QCFMSR sobre Hadoop.

- Cal investigar als fitxers de paràmetres de Hadoop com indicar que instanciï el codi QCFMSR.

- A l'HDFS RAID podem parametritzar un codi amb stripeSize i paritySize, però no podem personalitzar el parityLength o l'array de coeficients des d'un principi. Ara aquests paràmetres estan *hard-coded* ja que l'HDFS RAID no està preparat per especificar-los.

8.4 Valoració personal

La valoració personal que puc fer sobre aquest projecte de recerca és absolutament positiva. Positiva en diversos aspectes.

Primer, la recerca que he realitzat m'ha permès introduir-me en el món de l'emmagatzematge i el processament distribuït, pel qual ja sentia una curiositat, que havia sorgit no feia gaire temps quan, dins l'àmbit laboral, observava que per a certes aplicacions els RDBMS tradicionals es quedaven curts en el rendiment. Tanmateix, les arquitectures de gran escala sempre m'havien semblat apassionants.

Aquesta mena de sistemes ja són ara elements fonamentals per sustentar processos crítics de les empreses tecnològiques més importants del món, com Facebook, Twitter o Yahoo, entre d'altres. En els propers anys veurem com aquests sistemes són adaptats i comercialitzats a petites i mitjanes empreses en forma de serveis. El tren del Big Data porta a millorar els processos de presa de decisions amb la possibilitat d'incorporar multitud de variables en els anàlisis de negoci, i *ningú* voldrà quedar-s'hi fora. No es tracta només d'emmagatzemar la informació, sinó d'extreure'n un valor a través d'anàlisis diversos.

Segon, l'oportunitat d'haver treballat en contacte amb els professors del Departament d'Enginyeria de la Informació i les Comunicacions ha estat un descobriment que ha eixamplat la meva visió sobre una de les tasques importants universitàries, a banda de la docència, que és la recerca.

Durant els anys que he estat cursant la carrera, veia i vivia la universitat com a una transmissora de coneixements amb l'objectiu de sentar les bases d'una formació global

al voltant de la informàtica. Durant aquests estudis no sempre he trobat un sentit immediat a totes les assignatures realitzades. És amb el temps que vaig observant el sentit i la utilitat d'aquestes assignatures, probablement unes més que d'altres. Al llarg d'aquesta recerca he trobat, una mica més, el sentit de la meva formació universitària. Un dels problemes que jo havia detectat al llarg de la carrera és que, en molts casos, hi ha una manca de relació entre la teoria impartida i la aplicació a la pràctica. I, tanmateix, en el projecte final de carrera se m'ha fet present la connexió que existeix entre ambdós àmbits. Resumint, en aquesta recerca se m'ha fet palesa de com la teoria ens dóna resposta a problemes que es plantegen a la pràctica, i com aquesta pràctica acaba sent útil per a la societat. Aquesta conclusió sobre l'experiència en recerca, no només complementa sinó que augmenta la meva formació com a estudiant i senta unes bases sòlides per encaminar el meu futur.

Tercer, es dedueix del punt anterior que aquest treball m'ha conduit a desenvolupar habilitats requerides per dur a terme una recerca, i que fins ara no havia portat a terme. En aquest sentit, penso que ara disposo dels mètodes i els procediments adequats per abordar nous reptes de recerca.

I, **quart**, i per acabar, observo que el repte que suposa aplicar la teoria per solucionar problemes pràctics de la societat, amb la motivació de generar una utilitat, em resulta engrescador i em manté interessat per continuar, en el futur més immediat, vinculat, d'una manera o altre, a processos d'innovació a través de la recerca. En aquests moments, l'interès que he adquirit pels sistemes d'emmagatzematge distribuït fan que em plantegi encaminar la meva formació, cursant un màster especialitzat en aquesta matèria.

Capítol 9: Bibliografia

[1] Boletín Oficial del Estado, «XVI CONVENIO COLECTIVO ESTATAL DE EMPRESAS DE CONSULTORÍA Y ESTUDIOS,» 4 Abril 2009. [En línea]. Available: http://www.boe.es/boe/dias/2009/04/04/pdfs/BOE-A-2009-5688.pdf.

[2] T. White, Hadoop: The definitive guide, Sebastopol: O'Reilly, 2009.

[3] IBM, «What is Hadoop?,» [En línea]. Available: http://www.ibm.com/software/data/infosphere/hadoop/.

[4] Desconegut, «HADOOP discussion,» [En línea]. Available: http://cloudcamp.pbworks.com/w/page/16039033/Hadoop.

[5] J. F. Díaz Cañizares, «Integración de Hadoop con planificadores Batch,» Bellaterra, 2011.

[6] Vision Master Designs, «Map Reduce for the n00bs,» 2011 Octubre 2011. [En línea]. Available: http://visionmasterdesigns.com/blog/map-reduce-for-the-noobs/.

[7] X. Zhang, «A Simple Example to Demonstrate how does the MapReduce work,» 7 Maig 2013. [En línea]. Available: http://xiaochongzhang.me/blog/?p=338.

[8] B. Hedlund, «Understanding Hadoop Clusters and the Network,» 10 Setembre 2011. [En línea]. Available: http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network.

[9] M. K. Aguilera, R. Janakiraman y L. Xu, «Using Erasure Codes Eficiently for Storage in a Distributed System,» [En línea].

[10] R. B. Osama Khan, W. P. James Plank y C. Huang, «Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads,» [En línea].

[11] Z. Z. X. M. Amey Deshpande y D. N. Eno Thereska, «Does erasure coding have a role to play in my data center?,» [En línea].

[12] D. Floyer, «Erasure Coding and Cloud Storage Eternity,» WikiBon, 17 Octubre 2011. [En línea]. Available: http://wikibon.org/wiki/v/Erasure_Coding_and_Cloud_Storage_Eternity.

[13] Networking and Emerging Optimization, «La corrección de errores,» [En línea]. Available: http://neo.lcc.uma.es/evirtual/cdd/tutorial/fisico/correc.html. [Último acceso: 29 Novembre 2012].

[14] P. Corbett, «Erasing misconceptions around RAID & Erasure Codes,» 11 Desembre 2012. [En línea]. Available: https://communities.netapp.com/community/netapp-blogs/exposed/blog/2012/12/11/erasing-misconceptions-around-raid-erasure-codes.

[15] D. Papailiopoulos, «Erasure Codes for Storage,» [En línea]. Available: http://www-scf.usc.edu/~papailio/Erasure_Codes_for_Storage.html.

[16] A. G. Dimakis, K. Ramchandran, Y. Wu y C. Suh, «A Survey on Network Codes for Distributed Storage,» [En línea]. Available: http://arxiv.org/pdf/1004.4438.pdf (arXiv:1004.4438v1 [cs.IT]).

[17] B. Gastón, «Coding Techniques for Distributed Storage».

[18] F. Oggier y A. Datta, «Coding Techniques for Repairability in Networked Distributed Storage Systems,» 18 Setembre 2012. [En línea].

[19] B. Gastón, J. Pujol y M. Villanueva, «Quasi-cyclic Flexible Regenerating Codes,» 16 Maig 2013. [En línea]. Available: arXiv:1209.3977v2.

[20] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright y K. Ramchandran, «Network Coding for Distributed Storage Systems,» 2007. [En línea]. Available: http://www-bcf.usc.edu/~dimakis/RC_Journal.pdf.

[21] A. G. Dimakis, «The Repair Problem,» [En línea]. Available: http://csi.usc.edu/~dimakis/StorageWiki/doku.php?id=wiki:definitions:repair_problem.

[22] A. Dimakis, M. Sathiamoorthy, D. Papailopoulos, S. Chen, R. Vadali y D. Borthakur, «Erasure Codes for Big Data over Hadoop,» [En línea]. Available: http://msrvideo.vo.msecnd.net/rmcvideos/163611/dl/163611.pdf.

[23] D. Borthakur, «Implement erasure coding as a layer on HDFS,» [En línea]. Available: issues.apache.org/jira/browse/HDFS-503.

[24] Hadoop Wiki, «HDFS RAID,» 02 Novembre 2011. [En línea]. Available: http://wiki.apache.org/hadoop/HDFS-RAID.

[25] D. Borthakur, R. Schmidt, R. Vadali, S. Chen y P. Kiling, «HDFS RAID,» [En línea]. Available: http://www.slideshare.net/ydn/hdfs-raid-facebook.

[26] D. Borthakur, «HDFS and Erasure Codes (HDFS-RAID),» 28 Agost 2009. [En línea]. Available: http://hadoopblog.blogspot.com.es/2009/08/hdfs-and-erasure-codes-hdfs-raid.html. [Último acceso: 29 Novembre 2012].

[27] Jiangbo, «HDFS RAID,» 21 Desembre 2012. [En línea]. Available: http://jiangbo.me/blog/2012/12/21/hdfs-raid/.

[28] D. Borthakur, «HDFS block replica placement in your hands now!,» Setembre 2009. [En línea]. Available: http://hadoopblog.blogspot.com.es/2009/09/hdfs-block-replica-placement-in-your.html.

[29] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. Dimakis, R. Vadali, S. Chen y D. Borthakur, «XORing Elephants: Novel Erasure Codes for Big Data,» [En línea]. Available:

http://anrg.usc.edu/~maheswaran/Xorbas.pdf.

[30] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. Dimakis, R. Vadali, S. Chen y D. Borthakur, «Poster: Novel Codes for Cloud Storage,» [En línea]. Available: https://mhi.usc.edu/files/2012/04/Sathiamoorthy-Maheswaran.pdf.

[31] B. Eckel, Thinking in Java, 4a ed., New Jersey: Prentice Hall, 2006.

[32] Stackexchange (Math), «How to solve system of linear equations of XOR operation?,» [En línea]. Available: http://math.stackexchange.com/questions/169921/how-to-solve-system-of-linear-equations-of-xor-operation.

[33] M. ANTUNOVIĆ, «How To Build Optimal Hadoop Cluster,» [En línea]. Available: http://www.atlantbh.com/how-to-build-optimal-hadoop-cluster/.

[34] B. Fan, W. Tantisiriroj, L. Xiao y G. Gibson, «DiskReduce: Replication as a Prelude to Erasure Coding in Data-Intensive Scalable Computing,» Carnegie Mellon University. Parallel Data Laboratory, 2011.

[35] J. Gantz y D. Reinsel, «THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East,» Desembre 2012. [En línea]. Available: http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf.

[36] J. Gantz y D. Reinsel, «Extracting Value from Chaos,» Juny 2011. [En línea]. Available: http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf.

[37] Institut Universitari Dexeus, «Noves Tecnologies en Diabetis,» [En línea]. Available: http://www.endocrino.cat/page/diabetis/noves-tecnologies-en-diabetis. [Último acceso: 21 Juny 20013].

[38] Facebook, «DataInfrastructure@FB,» [En línea]. Available: https://www.facebook.com/datainfra.

[39] S. Krenzel, «Finding friends with MapReduce,» [En línea]. Available: http://stevekrenzel.com/finding-friends-with-mapreduce.

[40] D. S. Papailiopoulos y A. G. Dimakis, «Repairing Erasure Codes,» [En línea]. Available: http://static.usenix.org/event/fast11/posters_files/Papailiopoulos.pdf.