UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Comparativa entre MapReduce i Spark

Treball Final de Grau

George Bochileanu Parfenie

Especialitat d'Enginyeria del Software

Grau en Enginyeria Informàtica

Director: Oscar Romero Moral

Enginyeria de Serveis i Sistemes d'Informació (ESSI)

Universitat Politècnica de Catalunya (UPC) – BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

29 de juny 2021

Resum

Donada la quantitat de dades disponible actualment, i a causa del seu creixement exponencial, ha sorgit la necessitat d'analitzar aquestes dades per tal que les organitzacions tant públiques com privades puguin extreure decisions més informades. Les eines que més s'utilitzen actualment, per al processament de conjunts de dades fins a centenars de terabytes d'informació, són MapReduce i Spark.

En aquest treball d'investigació ens centrarem a automatitzar el procés d'experimentació sobre les eines de processament de dades MapReduce i Spark i realitzarem una comparativa entre aquests sistemes, on tractarem de determinar quin dels dos és el més adequat segons el cas d'ús. L'objectiu del treball és donar una vista objectiva, rigorosa i transparent sobre quina de les eines destaca avaluant uns indicadors en els experiments que són la mantenibilitat o facilitat de desenvolupar codi, el rendiment, la grandària del volum de dades, l'escalabilitat i el consum de recursos que obtindrem executant un conjunt d'experiments d'una forma automatitzada i repetible.

Paraules clau: Testbed, Datasets Profiler, Frameworks, Spark, MapReduce, Experimentació

Resumen

Dada la cantidad de datos que hay disponible actualmente, y a causa de su crecimiento exponencial, ha surgido la necesidad de analizar estos datos para que las organizaciones tanto públicas como privadas puedan extraer decisiones más informadas. Las herramientas que más se utilizan actualmente, para el procesamiento de conjuntos de datos de hasta centenares de terabytes de información, son MapReduce y Spark.

En este trabajo de investigación nos centraremos en automatizar el proceso de experimentación sobre las herramientas de procesamiento de datos MapReduce y Spark y realizaremos una comparativa entre estos sistemas, donde trataremos de determinar cuál de los dos es el más adecuado según el caso de uso. El objetivo del trabajo es dar una vista objetiva, rigurosa y transparente sobre cuál de las herramientas destaca avaluando unos indicadores en los experimentos que son la mantenibilidad o facilidad de desarrollar código, el rendimiento, el tamaño del volumen de datos, la escalabilidad y el consumo de recursos que obtendremos ejecutando un conjunto de experimentos de una forma automatizada y repetible.

Palabras clave: Testbed, Datasets Profiler, Frameworks, Spark, MapReduce, Experimentación

Abstract

Given the amount of data available today and due to its exponential growth, a need for analysis has emerged so both public and private organizations could make more informed decisions. The most used frameworks currently, used to process up to hundreds of terabytes of data, are MapReduce and Spark.

In this research, we are going to focus on automating the experimentation process on MapReduce and Spark data processing frameworks and we will compare these, where we will try to determine which one is better suited depending on the use case. The objective of this project is to give an objective, rigorous and transparent view on which framework standouts by assessing some indicators, which are maintainability or ease of code development, performance, dataset size, scalability, and resource consumption, that we will obtain by executing a set of experiments in an automated and repeatable fashion.

Keywords: Testbed, Datasets Profiler, Frameworks, Spark, MapReduce, Experimentation

Índex

Índex de figures

Índex de taules

Capítol 1

Contextualització i abast

1.1 Introducció i contextualització

1.1.1 Context

Aquest és un Treball de Final de Grau (TFG) del Grau en Enginyeria Informàtica (GEI) de la Facultat d'Informàtica de Barcelona (FIB). Aquest projecte s'ha realitzat dintre de la modalitat A, és a dir que és un projecte elaborat a la Universitat Politècnica de Catalunya (UPC) i dins del grup de recerca DTIM [1].

Podem destacar dues fites importants en la història d'Internet: la Web 1.0 i 2.0. En la versió 1.0 de la Web, les pàgines web només presentaven la pàgina web als usuaris. Amb el pas dels anys, al voltant del 2005, les pàgines web van començar a permetre interactuar amb elles, permetent als usuaris crear dades a través de pàgines web com Facebook, Youtube entre altres. Aquest canvi va ser el que va constituir la web 2.0. Amb el pas dels anys, la quantitat de dades generada no ha fet més que créixer exponencialment tal com podem observar a [2], fins que les tècniques utilitzades tradicionalment per al processament de dades han deixat de ser efectives [3].

En la Figura 1 podem observar clarament que la dataesfera o quantitat de dades disponible creix de forma exponencial.

La gran quantitat de dades present actualment té un impacte econòmic molt gran. S'estima que la majoria d'activitats econòmiques dependran de les dades en els pròxims anys. El valor de l'economia de dades (*data economy* – és el terme que fa referència al valor que tenen les dades) de la Unió Europea era d'aproximadament 325 milers de milions d'euros l'any 2019, representant un 2,6% del PIB. La mateixa estimació prediu que aquest valor incrementarà a més de 550 milers de milions d'euros l'any 2025, representant així un total del 4% del PIB total de la Unió Europea [5].

Podem veure que actualment, el *Big Data* i la ciència de dades tenen un impacte molt gran i les dades estan movent una gran suma de diners. En conseqüència, les dades s'han començat a veure com un bé actiu i moltes organitzacions, tant públiques com privades, van veure l'oportunitat d'aprofitar aquestes dades per al seu benefici. D'aquesta forma, ha sorgit la necessitat d'analitzar les dades per a poder extreure conclusions més ben informades.

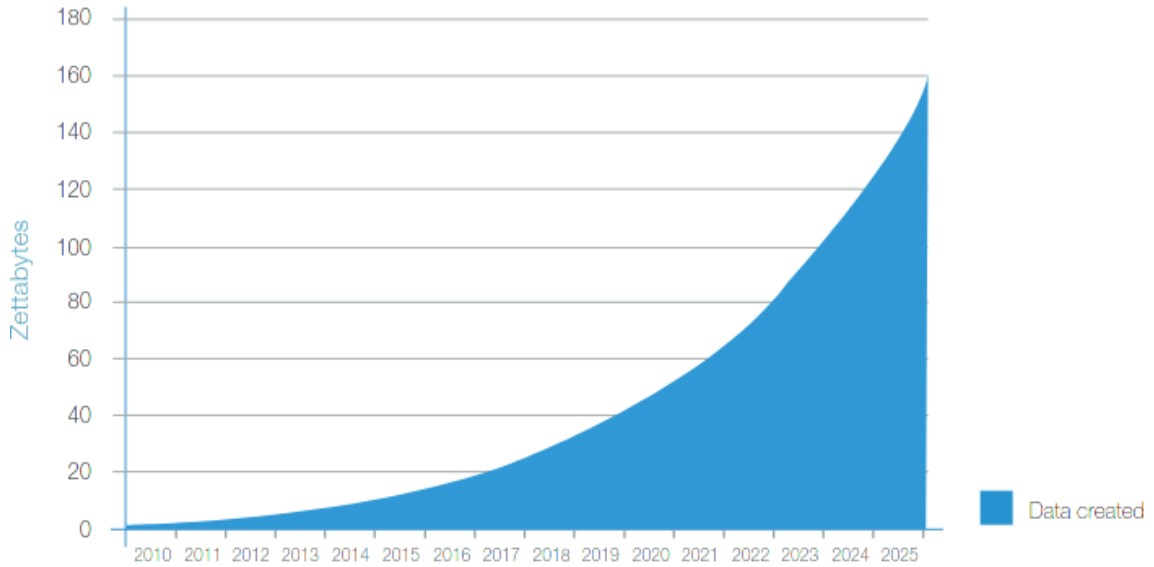


Figura 1: Mida anual de la Dataesfera Global
[Font: [4]]

Per tal de resoldre aquesta necessitat, durant aquest període, van sorgir diverses eines que permeten abstreure als programadors de la gran complexitat que suposa processar moltes dades entre molts ordinadors simultàniament. Aquests tipus d'eina reben el nom de *frameworks* de processament de dades, ja que faciliten poder analitzar les dades. Amb processament, ens referim a l'aplicació d'un conjunt d'operacions sobre un conjunt de dades per a explotar la informació que representen les dades.

Els dos *frameworks* més importants actualment i que han tingut molta rellevància en l'àmbit de l'anàlisi de grans volums de dades han sigut MapReduce i Spark. Cal notar que l'aparició de Spark ha sigut posterior a la de MapReduce, així que sembla que Spark jugui amb un lleuger avantatge enfront de MapReduce; encara que cal remarcar que la idea senzilla darrere de MapReduce i la forma en la qual processa les dades, pot fer que en alguns casos fins i tot sigui més adequat que el nou sistema Spark.

1.1.2 Termes i conceptes

Els conceptes essencials per a entendre el projecte es descriuen a continuació:

- ***Framework* de processament de dades.** És una eina que administra la transformació de les dades en múltiples passos. Generalment, aquests passos o tasques formen un Graf Acíclic Dirigit (o *Directed Acyclic Graph*, DAG, en anglès). Cada tasca consumeix unes entrades i produeix unes sortides. Les entrades poden ser arxius, estructures de dades en memòria i les dades poden ser estructurades o no estructurades [6].

- **Sistema d'arxius.** Component encarregat de controlar com estan emmagatzemades les dades [7]. En el nostre cas, estarem utilitzant HDFS [8], que és un sistema d'arxius distribuït que pot gestionar grans volums de dades en ordinadors barats (*commodity hardware*).

- **Spark.** És un *framework* de processament de dades que permet obtenir un alt rendiment per al processament de dades en *batch* i *streaming* – és a dir, processament de

dades en conjunts o bé un processament continu de dades. Aquest alt rendiment el pot obtenir gràcies al fet que planifica i optimitza la forma en la qual accedeix a les dades tal com es detalla més en profunditat a [9].

- **MapReduce.** És un *framework* de processament de dades caracteritzat per dividir-se en tres fases o passos diferenciats: Map, Shuffle i Reduce. Aquestes fases s'executen de manera distribuïda, en diferents nodes de processament [10]. El programador ha d'especificar el codi per a un Map i per a un Reduce. El *framework* té una fase obligatòria al mig que gestiona la comunicació entre la fase de Map i Reduce. Tal com podem veure a [11], MapReduce està orientat només al processament de dades en *batch*.

1.1.3 Problema a resoldre

Els *frameworks* que analitzen aquestes dades són eines molt complexos i requereixen un estudi detallat per a poder determinar quins serien els casos d'ús més favorables per a una eina o altra per tal de fer una anàlisi més eficient.

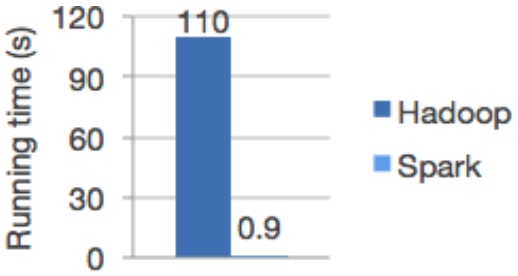


Figura 2: Comparació de Spark i Hadoop MapReduce
[Font: [9]]

En la Figura 2 podem observar les dades que s'anuncien a la pàgina web oficial de Spark [9]. Veiem que Spark millora fins a un centenar vegades el temps d'execució de MapReduce. Cal recalcar que si bé la Figura 2 mostra un rendiment de Spark molt millor, no és transparent a l'hora d'especificar sota quines condicions es va fer la comparativa. Per tant, cal mostrar-nos escèptics respecte a les dades anunciades, ja que es pot tractar purament de màrqueting. Cal destacar que Hadoop és el sistema que integra totes les eines necessàries per a poder executar MapReduce en un clúster. A vegades, es fa referència a MapReduce com a Hadoop MapReduce, fent al·lusió a l'ecosistema que permet executar MapReduce.

En aquest projecte, volem identificar els criteris rellevants per a un *framework* distribuït de processament de dades i posar els fonaments per poder fer comparatives transparents i que cobreixi els criteris que es defineixen per tal de poder entendre amb profunditat quan i en quines circumstàncies cadascun d'ells és preferible.

1.1.4 *Stakeholders*

El terme *stakeholders* vol dir parts interessades i fa referència a aquelles persones o organitzacions afectades per les activitats i decisions preses durant el desenvolupament del treball d'investigació. Les persones que es beneficiaran d'aquesta comparativa de forma directa i són *stakeholders* primaris, són el **director** del TFG i **l'investigador**, és a dir,

jo mateix, ja que estem involucrats en aquest projecte. Els resultats de la comparativa s'utilitzaran per part del director per a millorar els cursos del Grau en Enginyeria Informàtica relacionats amb el *Big Data* i també el Màster en Ciència de Dades.

La intenció d'aquest treball d'investigació és que sigui de domini públic i pugui ser consultat i emprat per qualsevol a internet. Per tant, com a *stakeholders* secundaris, tindríem als **científics i enginyers de dades de les organitzacions públiques i privades**, que es beneficiaran d'una forma directa, podent aprofitar aquest estudi per a fer altres anàlisis en el camp del *Big Data* enfocant-se en un únic sistema, aprofitant aquests resultats per a decidir quin sistema s'adequaria més al seu ús. Com a beneficiaris indirectes hi ha les **organitzacions públiques i privades** i els **seus clients** o **les persones afectades per les decisions de l'organització**, que necessitin tractar grans volums de dades per a uns casos d'ús determinats i no sàpiguen quina alternativa encaixa millor per a ells.

1.2 Justificació

1.2.1 Investigacions prèvies

Hi ha hagut diverses investigacions prèvies tractant de donar una solució quan es tractava de comparar MapReduce i Spark. Una comparativa que podem destacar és ≪Spark wins Daytona Gray Sort 100 TB Benchmark≫ [12]. Aquesta prova tractava de fer una ordenació de 100 TB de dades utilitzant 206 màquines. Ho van aconseguir en 23 minuts, trencant el rècord previ que era de 72 minuts, aconseguit per un clúster de Hadoop utilitzant MapReduce en 2.100 nodes. Aquesta comparativa, però, és molt específica i només fa èmfasi en el problema de l'ordenació de dades.

Podem destacar la investigació que es va dur a terme entre MapReduce i Spark a sobre de discs durs HAMR per Lu Liu [13]; els discs durs HAMR [14] són discs durs que permeten unes capacitats d'emmagatzematge superiors als discs durs actuals, on arriba a conclusions que Spark és molt més ràpid que MapReduce, sobretot en algorismes d'aprenentatge automàtic.

Cal destacar, però l'aportació de Sindhuja Hari [15], on remarca el fet que fins i tot els dos sistemes es poden utilitzar juntament en una empresa: MapReduce per a processar una quantitat molt gran de dades, mentre que es pot utilitzar Spark per a processar dades en temps real.

Els autors Houssam Benbrahim et al. [16] arriben a conclusions similars que els dos sistemes no tenen per què ser mutualment exclusius, encara que Spark és un sistema més nou que MapReduce per al processament de dades.

1.2.2 Justificació

La principal raó de fer aquesta investigació és perquè alguns estudis no són transparents a l'hora d'especificar sota quines condicions es va fer la comparativa. A l'estudi ≪Spark wins Daytona Gray Sort 100 TB Benchmark≫ [12], que està a la pàgina web de Spark, no es menciona el codi que s'utilitza ni tampoc com s'ha fet la comparació. Tampoc s'ha mencionat en cap estudi el desenvolupament d'un *Testbed* que posi cara a cara els dos sistemes

i minimitzi el nombre de variables per a fer la investigació encara més representativa. La falta d'aquest sistema podria ser fonamental en la comparació i podria fins i tot canviar el resultat en alguns casos d'ús.

Podem destacar dos àmbits on és important una investigació com aquesta. Primer de tot, en l'àmbit acadèmic, tot i que els dos sistemes es poden presentar com a continguts d'un curs de bases de dades, l'eina MapReduce és molt més senzilla d'explicar als estudiant com una primera aproximació al món del *Big Data* que Spark. La senzillesa de MapReduce fa que sigui fàcil poder posar exemples il·lustratius del seu funcionament. Spark, en canvi és un sistema molt més complex, on el seu funcionament intern pot fer menys assequible el món del *Big Data* als alumnes. Fer una comparativa per a aquest entorn ens serviria només per a poder tenir material per a ensenyar als alumnes les principals diferències empíriques dels dos sistemes.

Un altre entorn és el món laboral, on aquestes estan a la base de molts sistemes actualment en producció. Veiem a [17] que hi ha al voltant de 45 mil organitzacions que utilitzen Spark, mentre que en [18] veiem que hi ha al voltant de 61 mil empreses utilitzant MapReduce. Veiem que encara moltes organitzacions no han fet el canvi, aleshores la justificació d'aquesta investigació és si val la pena migrar de MapReduce a Spark en aquestes organitzacions, ja que no hi ha cap feina on s'hagi avaluat amb detall l'*overhead* que representa MapReduce, si en representa algun, respecte a Spark.

1.3 Introducció a MapReduce i Spark

En aquesta secció farem una introducció als *frameworks* de processament distribuït, més concretament a les eines Spark i MapReduce.

En general, un *framework* de processament de dades distribuït [19] és una eina que administra la transformació de les dades i ho fa en múltiples passos. Aquests passos, generalment formen en DAG (*Directed Acyclic Graph*). Una *pipeline* és un conjunt de tasques o operacions que s'executa en aquests *frameworks*. Cada operació consumeix una entrada i produeix una sortida i poden ser arxius, estructures en memòria, etc. Les dades poden estar estructurades o sense estructurar.

Si mirem la Figura 3, observarem que un *framework* de processament de dades té tres parts, que els descriurem a continuació:

- ***Master.*** Aquest component s'encarrega de programar les execucions en els *workers* i manté l'estat de l'execució del *pipeline*.
- ***Worker.*** Aquest és el node que realitza l'execució de les operacions programades pel *master*. Utilitza un repositori de dades per a poder llegir, processar i escriure els resultats.
- **Repositori de dades.** És el medi on s'emmagatzemen les dades per a ser processades. Aquest medi és distribuït, això vol dir que tots els *workers* poden treballar en una partició del repositori de dades en paral·lel als altres *workers*.

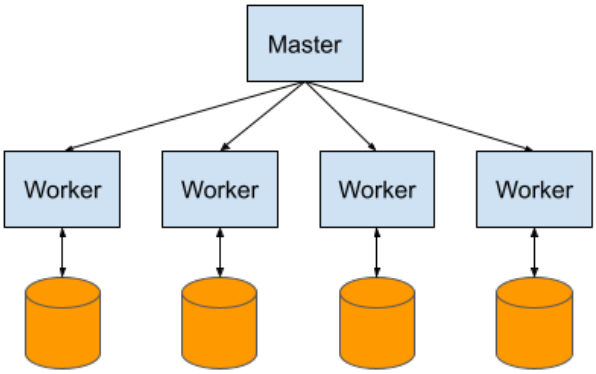


Figura 3: Esquema d'un *framework* de processament de dades
[Font: elaboració pròpia]

Aquests són els conceptes generals, encara que les eines MapReduce i Spark tenen certes especificitats que les fan diferents entre si. Es pot trobar a [20] més en detall aquesta comparació. A continuació farem un resum de les característiques que les fan diferents entre si:

MapReduce

- Utilitza com a repositori de dades de les operacions intermèdies el disc, concretament el sistema de fitxers distribuïts HDFS.
- Hi ha tres fases senzilles: Map, Shuffle i Reduce. La fase de Map i Reduce ha de ser programada pel programador, mentre que la fase de Shuffle a fa automàticament el *framework*.
- Per a executar diferents operacions, si aquestes no es poden fer amb les tres fases descrites anteriorment, aquestes fases s'han de concatenar i s'ha d'utilitzar el disc per a intercanviar les dades.

Spark

- Utilitza com a repositori de dades de les operacions intermèdies la memòria RAM
- Té una API molt senzilla per a comunicar-li les operacions relacionals que volem executar
- Té un optimitzador de les etapes d'execució anomenat DAG Scheduler.

1.4 Abast del projecte

1.4.1 Objectius

Aquest treball té dos objectius, que els detallarem a continuació:

1. L'objectiu principal d'aquest projecte és automatitzar l'experimentació sobre els *frameworks* de processament de dades MapReduce i Spark. Aquest objectiu està subdividit en dues parts:

- Creació d'un sistema per a automatitzar el procés de selecció dels conjunts de dades, que s'anomenarà *Datasets Profiler*. Per a poder fer experiments, és important saber amb quines dades estem treballant. Aquest sistema s'ha de construir perquè hem de caracteritzar els conjunts de dades per a que puguem interpretar correctament els experiments que fem sobre ells. Per exemple, si tenim un conjunt de dades amb una atribut on tots els valors són iguals, aquest no serà bo per a fer seleccions perquè amb una selecció d'un únic element, acabarem seleccionant tot el conjunt de dades.

- Creació d'un sistema per a automatitzar l'execució dels experiments, que s'anomenarà *Testbed*. Aquest sistema ha de ser una eina tal que donada una configuració inicial, que anomenarem *pipeline*, on s'especifiquen les operacions, pugui generar codi Spark i MapReduce, tant en un entorn local com en un entorn distribuït, i també pugui monitorar l'execució del codi i recollir les mètriques necessàries per als experiments.

2. El segon objectiu d'aquest projecte és fer una comparativa ens ajudaria a veure en quins casos MapReduce és millor que Spark i en els casos en els que no és millor, veure quin és el preu a pagar. Aquesta seria realment una primera aproximació per a la comparativa entre MapReduce i Spark però deixarem tot preparat per a que en un futur poguéssim fer comparatives amb un grau de detall més elevat.

1.4.2 Visió general

En aquest apartat mostrarem una visió general del nostre projecte, per a tenir un millor enteniment de com es relacionen totes les parts i com hem estructurat la nostra solució.

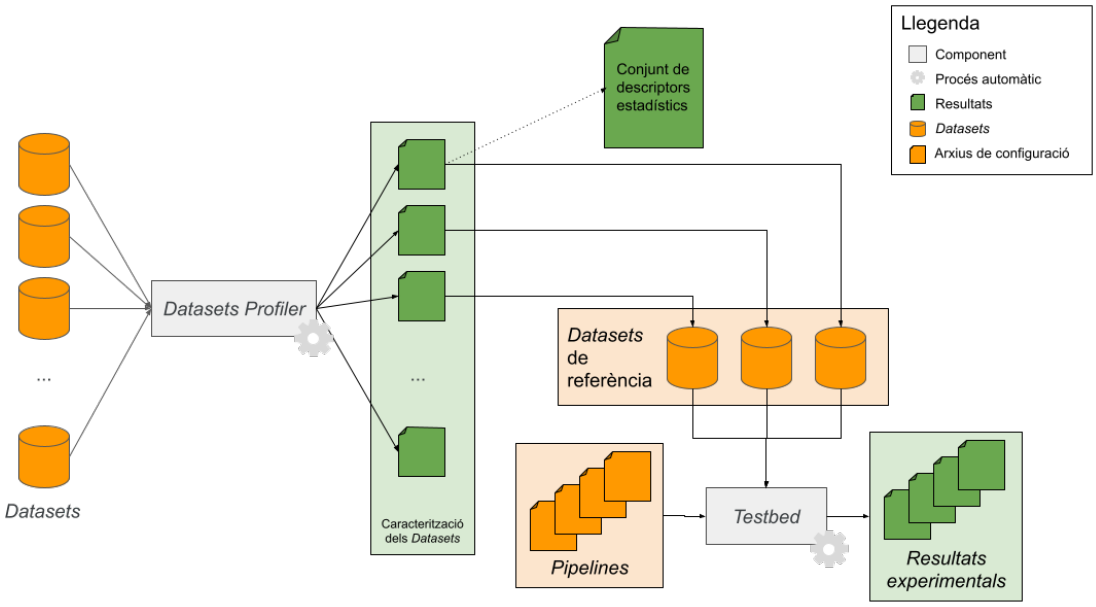


Figura 4: Visió general del projecte
[Font: elaboració pròpia]

Primer, el nostre treball serà col·leccionar una sèrie de conjunts de dades o *Datasets*; aquesta serà una tasca manual. Aquests conjunts de dades s'han d'analitzar un a un per a

trobar una caracterització d'aquests. Aquesta caracterització és imprescindible perquè hem de conèixer quines dades estem tractant. Tal com hem mencionat a la secció 1.4.1, si tenim una operació de selecció i volem seleccionar un atribut que té pocs valors diferents, aleshores no podrem fer uns experiments representatius del món real. Un cop tinguem aquestes caracteritzacions dels conjunts de descriptors estadístics, observem que per a poder tenir suficient informació, haurem de col·leccionar un conjunt determinat de descriptors estadístics. Aquesta tasca pot portar molt temps si volem analitzar un conjunt de *datasets* considerable. Aquesta tasca, si es realitzés manualment, implicaria moltes hores perdudes analitzant dades, implicaria moltes tasques repetitives i no obtindríem una varietat de conjunts de dades per a escollir per als nostres experiments, per tant, els resultats segurament es veurien esbiaixats. Aleshores, per a automatitzar aquesta tasca, hem implementat el sistema *Datasets Profiler*.

Un cop tinguem totes les caracteritzacions, haurem de fer un procés manual de selecció dels conjunts de dades per a decidir quins serien els més interessants i representatius del món real. Els conjunts de dades seleccionats de tots els analitzats reben el nom de *datasets* de referència.

Per a executar els experiments i poder fer una comparativa dels dos *frameworks*, hem de crear *pipelines* amb operacions que simulin escenaris de casos reals. Aquests *pipelines* utilitzaran els *datasets* de referència prèviament seleccionats. Per a poder comparar els les dues eines, durant l'execució del *Testbed*, recollirem mètriques en tots els nodes i aquests formaran els resultats experiments que es troben a la sortida del *Testbed*. Quan vam començar a analitzar el problema a solucionar, vam observar que molt del codi que havíem de crear per a poder executar els experiments es solapava i molts cops donava lloc a resultats poc rigorosos. Per tant, vam decidir crear un sistema genèric, anomenat *Testbed* que ens permet automatitzar l'execució dels experiments i la recollida dels resultats experimentals. Finalment, les conclusions de la comparació sortiran dels resultats experimentals obtinguts anteriorment.

1.4.3 Requeriments

Requeriments funcionals

Hem de contemplar els requeriments funcionals per als dos sistemes que implementarem en aquest projecte. Els requisits funcionals del *Datasets Profiler* es mostren a continuació:

- Ha de poder llegir un conjunt de dades, parsejar el seu contingut i extreure descriptors. Parsejar fa referència a transformar una entrada de text en una estructura de dades.
- Aquesta eina no hauria d'interactuar amb l'usuari. S'hauria d'executar i prendre a l'inici del programa els paràmetres necessaris i, a partir d'aquell moment, fer el perfil estadístic del conjunt de dades.
- Ha de mostrar els descriptors d'una forma que sigui interpretable per a poder classificar els conjunts de dades posteriorment.

Per al *Testbed*, els requeriments funcionals són els següents:

- Ha de poder llegir els conjunts de dades

- Aquest sistema s'ha d'executar diversos cops un darrere d'altre per a poder comprovar que un experiment és correcte.

- Aquesta eina no hauria d'interactuar amb l'usuari. S'hauria d'executar i prendre a l'inici del programa els paràmetres necessaris i, a partir d'aquell moment, executar l'experiment que pertoca.

- Ha de permetre oferir una interfície comuna d'execució tant per a Spark com per a MapReduce per tal de minimitzar el nombre de variables a l'hora d'executar els tests.

- Ha de permetre oferir una interfície amigable per a definir l'ordre d'execució de les operacions a provar en els tests, això és important perquè ens dona més flexibilitat i rapidesa a l'hora d'executar els experiments al clúster.

- Ha de permetre veure la instrumentació de les operacions invocades.

- Ha de permetre mesurar de forma precisa el temps d'invocació de les operacions.

Addicionalment, els requeriments de funcionalitat de la comparativa són l'anàlisi des del punt de vista de la mantenibilitat o de la facilitat de desenvolupar codi, del rendiment, de la grandària del volum de dades, de l'escalabilitat i del consum de recursos entre els dos sistemes a l'hora de realitzar els tests.

Requeriments no funcionals

Com a requeriments no funcionals, per a ambdós programes mencionats anteriorment, són els següents:

- **Matenibilitat.** Tractarem de fer l'arquitectura de tots els components d'una forma modular, seguint els principis SOLID [21] i les bones pràctiques de programació. D'aquesta forma farem els programes molt canviables, ja que recordem que es tracta d'una investigació i és important ser capaç de fer canvis en l'arquitectura i el codi amb suficient facilitat.

- **Escalabilitat.** De forma que pugui ser suficientment ràpid a l'hora d'analitzar des de petits volums de dades, fins a grans volums de dades.

- **Eficiència.** Els dos sistemes han de ser altament eficients, ja que el volum de dades és molt gran i, si els algorismes no segueixen aquest requisit, correm el risc que els sistemes no acabin en un temps raonable.

- **Usabilitat.** Aquest és un requisit no funcional enfocat als nostres *stakeholders*. Cap projecte poc usable i difícil d'entendre pot despertar l'iterés perquè altres persones l'utilitzin o, fins i tot l'estenguin per a provar nous punts de vista.

- **Transparència.** Aquest és un requisit no funcional enfocat a donar visibilitat i exposar públicament tant les dades utilitzades com els procediments emprats per a poder donar una major credibilitat a les conclusions de la comparativa.

1.4.4 Possibles obstacles i riscos

Durant la realització del projecte, hi ha un conjunt de possibles obstacles i riscos a l'hora de realitzar-lo:

- **Data d'entrega fixada.** Aquest risc està generat pel fet que la data per a acabar el TFG té un termini fix i, per al seu naturalesa d'investigació, és probable que hi hagi algunes desviacions respecte a la planificació.
- **Capacitat computacional.** Aquest risc està generat per fet que és possible que no tinguem disponible cap clúster per a realitzar els experiments. El processament de dades molt grans no té cap sentit si no disposem d'un clúster d'ordinadors que ens permeti fer el processament de dades en grans volums.
- **Complexitat dels *frameworks*.** Aquest risc està generat per la complexitat associada a cada un dels *frameworks* de processament de dades que estarem comparant. Aquests sistemes són molt complexos internament i poden requerir una gran quantitat de temps entendre'ls a un nivell de profunditat suficient per a fer la comparativa.
- ***Bugs*.** Aquest risc està generat pel *software*, ja que és molt probable que hi hagi errors en el codi. Aquest risc, podria dificultar la realització del *software* i, conseqüentment, endarrerir l'entrega del TFG.
- **Corrupció de les dades.** Aquest risc està generat per una fallada en el portàtil on es treballa. Això comportaria la pèrdua del codi i documentació redactats, i ens pot fer no arribar a temps per a la data d'entrega del TFG.

Per a tots els riscos anteriorment presentats, jo seré el responsable de mitigar-los prenent les mesures necessàries en cas que algun dels riscos es faci realitat.

1.5 Metodologia i rigor

1.5.1 Metodologia de treball

Per a aquest projecte, seguirem una filosofia *agile*, fent una adaptació de la metodologia de Kanban per al format del Treball Final de Grau. Ja que, recordem que aquesta és una tasca d'una persona que està sent supervisada per una altra. A més a més, és difícil fer una planificació completa de totes les proves a causa de la naturalesa de comparativa, on no se saben amb anterioritat els resultats que s'obtindran.

Kanban [22] és una metodologia inventada al Japó, on el seu enfocament és en crear un tauler amb un símbol visual que s'utilitza per a desencadenar una acció. D'aquesta manera es representen els processos de fluxos de treball.

El nostre tauler tindrà les següents columnes, que al seu torn contindran les accions que passaran d'una columna a l'altra en funció del seu estat:

- ***To-do*:** per a les tasques planificades, però que encara no s'han començat.
- ***In progress*:** per a les tasques que s'han començat, però encara no estan acabades.

- ***Done*:** per a les tasques completades.

Utilitzarem l'eina Trello per a crear el tauler. Aquesta eina destaca per la seva senzillesa i usabilitat. La nostra adaptació de la metodologia serà que es faran reunions setmanals per a presentar les tasques fetes durant la setmana, es trauran conclusions i es prendran decisions per a les següents etapes segons els resultats obtinguts durant la setmana.

1.5.2 Eines de seguiment i validació

Utilitzarem l'eina Git per al control de versions del codi a l'hora de desenvolupar codi des del portàtil, executarem tests d'integritat i només publicarem el codi a GitHub quan els tests hagin passat correctament. La forma en la qual utilitzarem el control de versions és mantenint una única branca anomenada *main*. A l'hora de pujar el codi utilitzarem exclusivament el *rebase* per a aplicar els canvis. *Rebase* [23] és una tècnica que permet reaplicar els canvis de codi al damunt d'una branca; d'aquesta forma permet mantenir una única línia temporal dels canvis introduïts al repositori que hauran succeït en la branca.

Cada setmana farem una comunicació síncrona a través d'una reunió entre el director del TFG i jo mateix a través de Google Meet i es compartiran arxius utilitzant Google Drive. La comunicació de forma asíncrona es farà a través de missatges de correu electrònic en Gmail.

Capítol 2

Identificació de lleis i regulacions

Aquest és un projecte d'investigació centrat en el *Big Data*. Per tant, hem d'esperar que les lleis s'apliquin sobre el processament d'aquestes i depenent també del contingut d'aquestes dades. A l'estat Espanyol, una llei molt important a tenir en compte és la Llei de Protecció de Dades de Caràcter Personal (LOPD) [24]. Aquesta llei té per objectiu assegurar i protegir, respecte al tractament de les dades, les llibertats públiques i els drets essencials de les persones físiques, i en especial del seu honor i intimitat personal i familiar [25].

Aquesta llei no ens afectaria en la realització d'aquest projecte, ja que estem processant dades de caràcter públic, que no posen en compromís aquesta llei pel fet que les dades s'utilitzen només com una entrada i l'objectiu d'aquest treball és mesurar el rendiment del sistema que processa les dades. Aquesta llei, en canvi sí que pot afectar més directament als nostres *stakeholders*, que podran utilitzar el resultat d'aquesta investigació per a escollir entre un sistema o un altre i depenent de l'ús que en facin de les dades, aquest ús estarà subjecte a la llei LOPD.

En l'àmbit europeu, també hi ha el Reglament General de Protecció de Dades (RGPD) [26]. Aquesta llei entrà en vigor el dia 25 de maig de l'any 2021 i substitueix a l'actual llei LOPD de l'estat Espanyol. Aquesta nova llei proporciona una major seguretat als usuaris i sobre quines dades seves té una empresa i com les tracten. Aquesta llei és molt similar a la LOPD i les principals diferències es poden consultar a [27]. De forma similar a la llei LOPD, aquesta llei tampoc ens afectaria de forma directa.

Cal fer èmfasi en el fet que tot i que hi ha lleis existents que protegeixen d'alguna forma l'ús que es fa de les dades personals, actualment estem molt lluny del fet que les empreses no abusin o utilitzin la informació que les persones proporcionen per a favor seu. La forma de protegir, aleshores a les parts afectades, que són les persones físiques, és per mitjà de l'ètica. Aquesta és l'eina que s'utilitza fins que els organismes responsables arribin a treure una llei que permeti protegir els drets de les persones.

Actualment, ens trobem en un període de temps alegal, on com podem observar, les lleis que protegeixen els drets de les persones escassegen, de tal forma que les grans empreses, tenen el dret de poder utilitzar les dades de les persones per al seu propi benefici. Recentment, es va estrenar un documental anomenat "El Dilema de les xarxes socials"[28], que planteja

el tema ètic i com afecta la societat el tractament abusiu de les dades de les grans empreses.

Un tema a destacar és com les grans empreses utilitzen les dades de les persones per a crear un perfil psicològic d'aquestes per tal de poder recomanar-los anuncis cada cop més enfocats a un públic objectiu específic. Aquestes empreses grans tenen un gran poder i una gran responsabilitat, ja que poden arribar a moltes persones i, poden fer canviar el comportament de les persones sense que aquestes siguin conscients.

Les aplicacions es dissenyen perquè siguin addictives, d'aquesta forma, l'usuari es veu en la necessitat d'utilitzar de forma cada cop més freqüent l'aplicació i a desenvolupar una dependència d'aquella aplicació.

En els últims anys, el poder de càlcul va evolucionar de forma exponencial; això ha permès que és poguessin implementar algorismes suficientment complicats perquè s'anomenin intel·ligència. Aleshores, l'ordinador s'entrena segons un objectiu, una definició d'èxit, que les empreses defineixen. Per a obtenir les característiques que més triomfen, es fa ús d'un algorisme anomenat Test A/B. Com a conseqüència, no totes les persones observen el mateix. Aleshores, es genera una falsa sensació que tot el món està a favor teu. En aquest moment, la manipulació és senzilla de fer.

Una dada molt important que es menciona a [28] és que les notícies falses es propaguen uns 6 cops més ràpid que les notícies certes. Cal remarcar que Facebook, per exemple es pot utilitzar com una eina de persuasió que, en mans d'un govern autoritari, poden manipular a la societat.

Finalment, el documental [28] conclou que aquestes empreses no haurien d'estar autoritzades per a prendre les decisions ètiques. Aleshores, l'ús que es faci de les dades, s'hauria de regular, simplement per exemple proposant un incentiu fiscal perquè les empreses no es quedin amb la totalitat de les dades. Però, durant el període en el qual no hi ha lleis, el nostre deure, com a societat és exigir que les xarxes socials es dissenyin d'una forma més humanitària.

Tot i això, la relació del projecte amb les lleis actuals i amb l'ètica de l'ús que es fa de les dades és indirecta, encara que hi és, perquè estem estudiant com ser més eficient a l'hora d'emprar els *frameworks* de processament de dades i, per tant, habilitant a les organitzacions a ser més eficients en les seves anàlisis de dades.

Capítol 3

Datasets Profiler

El procés de creació d'un perfil estadístic consisteix bàsicament en tres passos: identificar el tipus de dada que estem tractant, treure descriptors estadístics en funció d'aquest tipus de dada i, finalment resumir en un format amigable els resultats de l'anàlisi, tal com es pot veure en la secció A.3 dels apèndixs. En aquest capítol ens centrarem en explicar el tipus d'arquitectura utilitzada, veurem l'arquitectura i especificarem els detalls més rellevants de la implementació i finalment, demostrarem la correctesa d'aquest programa.

3.1 Tipus d'arquitectura utilitzada

L'arquitectura emergent [29] és l'arquitectura que té estructures organitzacionals com processos de negoci i tecnologies que són dissenyades de forma incremental. Aquesta és una arquitectura utilitzada en el món del *Agile* per a construir sistemes incrementalment a cada iteració.

Principalment, aquesta arquitectura neix de la necessitat de ser més adaptable i respondre d'una forma més ràpida als canvis [30]. Aquesta necessitat ha reinventat com s'està actualment pensant en el món del desenvolupament de *software* a l'hora de crear la part més important d'un sistema, que és la seva arquitectura.

Aquest tipus d'arquitectura, té els seus punts positius i negatius.

Els punt més positius són que permet crear una arquitectura *ad-hoc* per a cada sistema. El cost de crear un disseny *future-proof* no té massa sentit perquè aquesta serà una eina que només utilitzarem per a crear un perfil estadístic d'un determinat conjunt de *datasets*. Per tant, no té sentit dedicar temps en això. Aquesta decisió fa que aquest tipus d'arquitectures resultin en estructures senzilles i molt fàcils d'entendre. També aquest tipus d'arquitectura ajuda a que hi hagi una forma ràpida de rebre *feedback* sobre un disseny. Aquest és un punt important ja que no volem perdre molt temps dissenyant un sistema que no tingui res a veure amb el que estàvem planificant inicialment.

Com a punts negatius, al veure una arquitectura com en la Figura 5, el disseny emergent pot semblar inicialment un disseny desorganitzat i inconsistent, per la seva natura emergent, o també pot semblar poc robust, sobretot si aquest disseny va evolucionant entre diversos

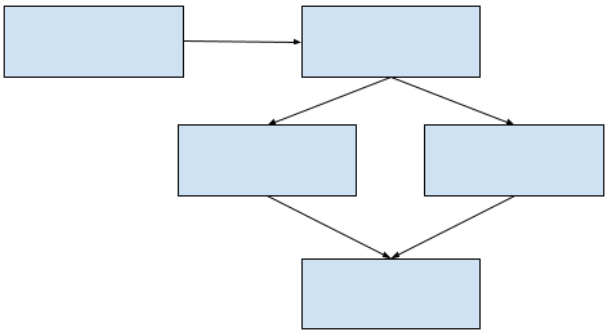


Figura 5: Forma d'una arquitectura emergent
[Font: elaboració pròpia]

integrants d'un equip i a mesura que es van desenvolupant nous components del sistema, si no se segueixen els bons patrons de disseny com per exemple el *Single Responsibility Principle* (SRP) o el *Open-Closed Principle* (OCP) dels principis SOLID, és possible que el desenvolupament es torni ràpidament en poc productiu. Per tant, un punt important i a tenir en compte a l'hora de crear aquest tipus de disseny, és que haurem de seguir aquesta sèrie de principis SOLID per tal de fer més eficient el disseny i evitar fer un disseny de forma ràpida, adquirir deute tècnic i posteriorment estar pagant aquest deute tècnic amb hores de desenvolupament extra.

Els principis SOLID [31] són una sèrie de principis en el desenvolupament del *software*, que tenen els següents objectius:

- Crear un codi eficaç, robust i estable
- Crear un codi nét i fàcil de modificar, és a dir mantenible
- Permetre l'escalabilitat del codi; de forma que si volem afegir noves funcionalitats, que aquestes siguin fàcils d'integrar

Aquests principis ens seran molt importants a l'hora de fer la implementació del nostre sistema per tal de poder reduir la quantitat d'hores totals dedicades a construir aquest sistema i complir un dels requisits que ens vam imposar inicialment, que era que la combinació de temps de desenvolupament dedicada més el temps que s'estigui executant el programa resulti en un temps menor que fer els perfils estadístics manualment.

També reduirem el temps que estarem desenvolupant evitant les solucions més que impliquin crear un sistema menys robust i flexible en el llarg termini. Dit amb altres paraules, seria adquirir un deute tècnic baix al començament del projecte, per tal d'evitar pagar-lo al llarg termini en hores de desenvolupament i evitar així que el nostre projecte ens prengui moltes més hores de les inicialment planejades.

El deute tècnic [32] es defineix com el cost i els interessos a pagar per fer malament les coses. És el sobre-esforç a pagar de mantenir un producte software mal fet. Tot i que sembli que aquest deute l'haguem de pagar en un futur llunyà, o només en els sistemes que s'utilitzen en producció, això no és cert. De fet, és possible que amb sistemes *software* mal fets, aquest deute tècnic es comenci a notar de forma important a partir dels primers components creats. A la llarga, quan el sistema d'hagi d'utilitzar, si hi ha algun canvi que s'hagi de fer (integrar

més descriptors estadístics o bé passar el programa a executar-se al clúster) és important que no prengui massa temps. Per això serà important el fet de refactoritzar el codi i deixar-lo sempre el més nét i senzill possible. També és important fer aquesta neteja del codi a mesura que es va desenvolupant, *commit* a *commit* i no deixar-lo per a final, perquè molts cops no s'acaba fent.

Aquesta filosofia de desenvolupament ens permetrà complir amb els requisits especificats a la secció 1.4.3 i evitar que ens desviem massa de la planificació inicial.

Diferents alternatives a aquesta arquitectura serien utilitzar una Arquitectura de Tres Capes [33], que és una arquitectura molt utilitzada per a sistemes *software*. Podem observar aquesta arquitectura en la Figura 6.

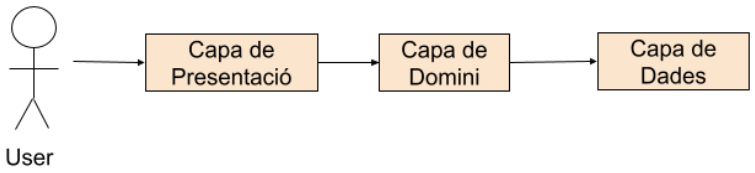


Figura 6: Arquitectura de Tres Capes
[Font: elaboració pròpia]

Com podem veure a la Figura 6, l'arquitectura té tres capes, que les detallarem a continuació:

- **Capa de Presentació.** Aquesta capa és l'interfície d'usuari i comunicacions amb l'exterior on l'usuari final interactua amb l'aplicació.
- **Capa de Domini.** És la capa que conté la lògica de negoci, i és el cor del sistema.
- **Capa de Dades.** Aquesta és la capa que administra les dades: la creació, lectura, modificació i eliminació de les dades.

La desavantatge principal d'aquesta arquitectura, i la fa poc recomanable per al nostre projecte és que sembla ser un disseny *over-engineered* per a l'eina que volem implementar. La capa de presentació, de fet hauria de no fer gairebé res, ja que aquest programa no hauria d'interactuar amb l'usuari al llarg de la seva execució. La capa de dades hauria de ser bàsicament un component que llegeixi el conjunt de dades i un altre que sigui capaç d'escriure el resultat. Si extraiem aquestes dues capes i els considerem simples components i la lògica de negoci la dividim en el parseig i processament de les columnes, aleshores ja tenim un disseny molt més simple i que evita tenir capes molt descompensades en quant a la grandària.

Aquest sistema seria més adequat per a sistemes que utilitzin pròpiament una base dades, i que per exemple, necessitin administrar els usuaris en una base de dades, o bé per a crear una pàgina web que mostri els productes d'una botiga *online*.

Una altra alternativa a proposar que seria potencialment bona a utilitzar seria implementar una arquitectura *EBI (Entity Boundary Interactor)* [34]. Aquesta és una arquitectura moderna utilitzable per a un ampli ventall d'estils d'aplicació. La seva idea més rellevant és

que tracta de produir una arquitectura agnòstica de l'implementació. Aquest tipus d'arquitectura, disposa de tres components importants, que els mostrarem en la Figura 7.

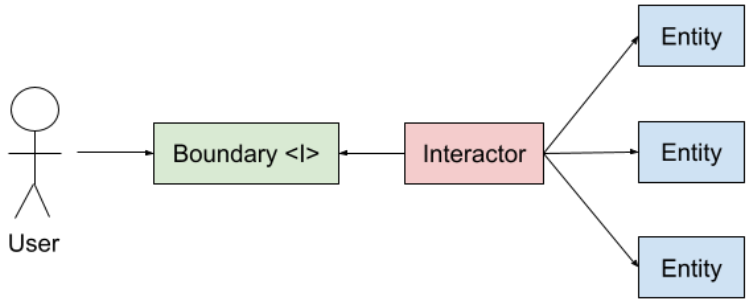


Figura 7: Arquitectura EBI
[Font: elaboració pròpia]

A continuació, explicarem les diferents parts presents en l'arquitectura mostrada en la Figura 7.

- ***Entity.*** Defineix les classes entitats de l'aplicació. Aquests són objectes reutilitzables i que no contenen cap lògica de negoci. Són *POJOs (Plain Old Java Objects)*, que s'utilitzarien en el context de l'aplicació que estem utilitzant.

- ***Boundary.*** Aquesta part del sistema és l'encarregada d'abstraure els diferents *frameworks*, serveis o aplicacions externes que estigui utilitzant el nostre programa. És tracta bàsicament d'una interfície amb el món exterior.

- ***Interactor.*** Són els components que contenen la lògica de negoci. Aquests interactuen amb les *entities* i la *boundary* per a poder implementar els casos d'ús.

Aquesta arquitectura posa un especial èmfasi en el disseny del sistema partint dels casos d'ús que necessitem. Com en aquest sistema va estar previst inicialment només un cas d'ús, no vam contemplar aquesta possibilitat de tenir diversos casos d'ús. Addicionalment, un punt important que ens va fer rebutjar aquesta arquitectura per a aquesta eina és que no hi ha cap necessitat d'una part del sistema del tipus *boundary*. Nosaltres utilitzarem només un únic *framework* i no hi hauria cap problema si aquest *framework* l'utilitzem de forma acoblada al nostre disseny, perquè no és previsible que en un futur haguem de canviar-lo. En aquest vídeo [35] de Robert C. Martin, s'especifica com funcionaria aquesta arquitectura. Un fet important que destaca és que l'arquitectura d'un sistema software, hauria de ser com el plànol d'un edifici, és a dir, que només amb mirar-lo, poguessis ser capaç de veure les diferents parts que té i poder dir, només mirant l'arquitectura que té, quin tipus d'edifici és. Aquesta arquitectura es proposa com una forma d'organitzar els diferents components d'un sistema, de forma que es pugui saber, només mirant l'estructura de carpetes de la implementació, de quin programa s'està tractant. Aquesta arquitectura que Robert C. Martin proposa seria més adequada per a un sistema més gran, que depengués de diferents *frameworks* o sistemes externs. Encara que nosaltres només utilitzarem un, que és Spark, per la seva facilitat d'integrar-se amb Python, per la qual cosa, aquesta característica de l'arquitectura no l'aprofitaríem.

En conclusió, l'arquitectura que ens ha semblat més interessant i apropiada per a aquesta

eina és l'arquitectura emergent, ja que ens permetria crear una arquitectura suficientment senzilla que permetés anar iterant seguint la nostra metodologia *Agile* i poder anar afegint elements a aquesta arquitectura d'una forma més flexible que les altres dues solucions. També les altres arquitectures semblen ser solucions per a problemes més grans, on s'han de gestionar diversos sistemes externs i tenir una estructura clara de tres capes o de *EBI*, facilitaria molt tenir els diferents mòduls organitzats per a tenir una arquitectura més neta. Un altre punt important en la nostra decisió és que aquesta arquitectura només dependria d'un únic *framework*, que no canviarà al llarg del desenvolupament i després d'aquest, que és Spark. Per tant, per a no tenir una solució *over-engineered*, hem decidit utilitzar l'arquitectura emergent.

3.2 Arquitectura

Un cop especificat el tipus d'arquitectura que utilitzarem, a partir dels requisits explicarem l'arquitectura del nostre sistema.

L'arquitectura que hem dissenyat se centraria en dos passos fonamentals: treure descriptors estadístics en funció del tipus de dada i resumir en un format amigable els resultats de l'anàlisi. Aquesta arquitectura es detalla a la Figura 8.

Cal destacar que la lògica que hem seguit per a dissenyat aquesta arquitectura ha començat sent emergent però, al final, aquesta va convergir en una arquitectura semblant a la que tenen els processadors. És a dir, hi ha una unitat de control i una unitat de processament. L'unitat de processament és la que processa pròpiament les dades i és la que apareix com a quadres blaus en la Figura 8. Aquesta es pot invocar per mitjà d'una funció que rep uns paràmetres i retorna un resultat. És com si es tractés d'una funció només. L'unitat de control s'encarrega de gestionar les funcions per a utilitzar-les d'una forma que ens permetin executar el nostre cas d'ús, que és aconseguir un perfil estadístic de la font de dades.

Aquest disseny ens va permetre molta flexibilitat ja que podíem afegir totes les funcions que volguéssim i només les teníem que enllaçar en la unitat de control per a que s'executin en el cas d'ús.

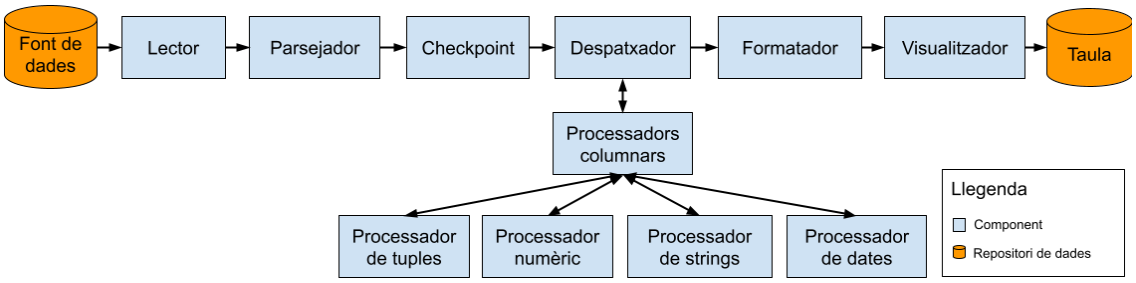


Figura 8: Arquitectura del *Datasets Profiler*
[Font: elaboració pròpia]

A continuació descriurem els components que formen l'arquitectura de la Figura 8.

3.2.1 Font de dades

Són totes les fonts de dades analitzades en el capítol A dels apèndixs.

3.2.2 Lector

Aquest component hauria de ser capaç de llegir la font de dades d'entrada i preparar-la en un format que servirà d'entrada per al parsejador.

3.2.3 Parsejador

Aquest component s'hauria d'encarregar d'agafar tot el conjunt de dades i donar-li una estructura i assignar-li un tipus de dades. Cal destacar que la problemàtica en aquest component estaria que és difícil tenir un parsejador genèric, degut a la gran quantitat de formats en la que es presenten els conjunts de dades. Aleshores, aquest component serà específic per a cada conjunt de dades. L'estructura serà bàsicament una taula per columnes, on cada columna tindria un significat. Molts dels conjunts de dades ja venen per columnes, mentre que per als que no venen d'aquesta forma, els imposarem nosaltres un particionat per columnes.

Depenent del contingut de les columnes, assignarem a cada columna un tipus de dada. Els tipus de dades que s'utilitzen en els conjunts de dades són tres: numèric, cadenes de text i dates o *timestamps*. Aleshores, el parsejador assignarà un d'aquests tipus a cada columna.

3.2.4 *Checkpoint*

El *framework* Spark, per a poder aprofitar el paral·lelisme del clúster, ha de tenir les dades distribuïdes als nodes del clúster. Sinó, Spark no tindria massa sentit perquè les dades haurien de passar pel *driver*. Per tant, a l'hora de parsejar el conjunt de dades, per a poder aprofitar el paral·lelisme del clúster, es necessitarà una forma més eficient de llegir les dades per part del despatxador.

Aquest component és l'encarregat de guardar les dades de forma persistent a HDFS per a que els executors de Spark puguin llegir aquestes dades de forma distribuïda i les puguin processar. Dit d'una altra forma, si el conjunt de dades parsejat no es guarda en format HDFS, aquest no estarà disponible a les diferents màquines. També, els processadors no podran aprofitar de forma eficient el format en el que estiguin les dades. Dit d'una altra forma, el lector llegeix línia a línia l'arxiu de text. Aleshores, el parsejador dona format a la línia que s'està escrivint. Aleshores, els processadors columnars haurien d'esperar a que es processi tot el conjunt de dades, i tenir en memòria tot el conjunt de dades (que molts cops no es pot per la mida dels conjunts de dades), i també s'hauria de re-processar el conjunt de dades passant-lo pel parsejador per a cada processador columnar si no s'implementa cap mesura de caching per com està dissenyat Spark.

Aleshores, hem integrat aquest component per a poder desacoblar el flux de dades de la font de les dades, passant per parsejador i guardant-se en un arxiu. Posteriorment, s'inicia un flux de dades per a tot el clúster i tots els executors en un format a HDFS que és eficient de processar per columnes.

3.2.5 Despatxador

El treball del despatxador és senzill: per a totes les columnes, mira el tipus de columna assignat a la fase prèvia i determina quin processador columnar és l'adequat per a aquella

columna. Per exemple, si una columna té el tipus de dada numèric, aleshores, el despatxador enviarà aquesta columna al processador numèric. El mateix passaria amb els altres dos tipus de dades, que són *strings* i *timestamps*.

3.2.6 Processadors columnars

Els processadors per tipus de dades són els components encarregats de recollir els descriptors estadístics del conjunt de dades. Es poden veure quins descriptors estadístics hauríem de recollir en la secció A.3 dels apèndixs.

Per a una columna determinada del conjunt de dades, aquests recollirien tots els descriptors estadístics rellevants i els enviaria al despatxador al finalitzar. El processador de tuples s'encarregaria de processar tot el conjunt de dades complet, és dir, agafant cada fila, convertint-la en una tupla i processant-la.

3.2.7 Formatador

A l'hora de visualitzar les dades, el format real o lògic i el format de visualització de les dades no té perquè ser igual. Un exemple és el conjunt de dades Android, on els instants de temps no tenen un any específic. Aleshores, s'ha d'utilitzar un any determinat i saber a l'hora de visualitzar les dades que no hi ha any. Un altre exemple és a l'hora de visualitzar els nombres amb coma flotant. Especificar el nombre de decimals que es puguin veure hauria d'estar desacoblat de l'estàndard IEEE754 que codifica la coma flotant als ordinadors.

Aquest component és l'encarregat de poder mantenir una separació entre el format real i el format de visualització. Aquest és relativament senzill, ja que només intercepta la sortida del despatxador i canvia com es visualitzen les dades. Per exemple, per al conjunt de dades Android, s'utilitzaria un formatador que no indiqués l'any.

3.2.8 Visualitzador

Aquest component s'encarregarà de prendre la sortida del despatxador i convertir-la en un format amigable per a ser visualitzat i interpretat posteriorment.

3.2.9 Taula

Aquest arxiu serà la sortida del programa i contindrà el perfil estadístic del conjunt de dades analitzat.

3.3 Detalls d'implementació

En aquesta secció anirem component a component de l'última arquitectura i detallarem la implementació de cada element de l'arquitectura.

3.3.1 Font de dades

Són totes les fonts de dades analitzades en el capítol A dels apèndixs. Les fonts de dades que tractem tenen dos formats que hem de considerar. El format .csv i el format .log. Tots

dos formats es poden llegir línia a línia per a obtenir tot el conjunt de dades complet, i cada línia fa referència a exactament una fila.

3.3.2 Lector

Aquest component llegeix el conjunt de dades indicat com un argument del programa.

Cal destacar que per a evitar haver de crear diferents versions dels conjunts de dades amb e nombre de files limitades, i per a poder mostrejar els conjunts de dades, hem permès que aquest component tingui un paràmetre que indiqui la quantitat de files que es poden llegir des del començament del conjunt de dades i, per tant que es processaran. Això ens permetrà també limitar el processament de les dades.

3.3.3 Parsejador

El parsejador és un component que transforma una fila del conjunt de dades en format text, i li assigna un determinada estructura i un tipus de dades. Les implementacions que havíem considerat eren utilitzar *Machine Learning* per a poder identificar els patrons automàticament i poder parsejar els continguts dels conjunts de dades. La segona alternativa que teníem en ment va ser crear un parsejador per a cada conjunt de dades i especificar-lo com un paràmetre.

L'avantatge que suposava utilitzar *Machine Learning* és que no havíem de crear moltes classes similars per a poder parsejar els continguts. A més, ens estalviàvem haver de provar totes les classes i trobar errors en el parseig. Com a desavantatge important és que el temps necessari per a desenvolupar aquesta alternativa era desconegut, ja que mai abans havia utilitat *Machine Learning* per a un cas d'ús similar. Aleshores, m'hagués hagut de familiaritzar primer, i posteriorment, hauria de crear un determinat model i entrenar-lo amb dades. En aquesta alternativa, també és més difícil controlar com es fa el parseig, i pot ser que no es faci de forma determinista, depenent de l'algorisme utilitzat.

L'opció alternativa, que és crear classes que facin el parseig té l'avantatge que si s'implementa seguint els bons patrons de disseny del *software*, aleshores, no s'hauria de replicar molt codi, només s'haurien d'especificar les columnes, els tipus de dades, si els camps poden tenir valors nuls i com es divideixen els valors en columnes. Aquesta implementació s'estimava que requeria molt poc temps d'implementació, a més de ser completament determinista, ja que sempre s'executa el mateix codi. A més, podem saber amb exactitud el que s'està fent en cada moment.

Per la facilitat de programació i pel temps reduït de desenvolupament requerit, ens vam decantar per la segona opció, que consistia en crear una classe que s'encarregui del parseig par a cada conjunt de dades.

Puntualment, es pot donar el cas en que els logs que es processen no segueixen exactament la mateixa estructura. Podem observar en el Fragment 3.1 un *log* que no seria vàlid.

Encara que el *log* més abundant sigui com el que podem veure en el Fragment 3.2.

En aquests casos, hem deixat en mans de l'usuari, quan es crea el parsejador dels *logs*,

```
2015-12-03 14:37:47,611 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting DataNode
...
************************************************************/
```

Fragment 3.1: Secció invàlida d'un *log*
[Font: elaboració pròpia]

```
2015-12-03 14:37:47,618 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: registered
    UNIX signal handlers for [TERM, HUP, INT]
```

Fragment 3.2: Secció vàlida d'un *log*
[Font: elaboració pròpia]

que es decideixi si aturar el flux de treball per a poder extreure aquestes dades manualment, o bé ignorar-les. En el nostre cas, hem optat per ignorar-les ja que suposen una minoria quan les comparem amb la resta de *logs*.

3.3.4 *Checkpoint*

Aquest component intercepta el *dataframe* de Spark i el fa persistent per a tenir una cache del parseig. La principal raó d'aquest component és que ens permet fer més ràpids els processadors columnars. Al tenir una cache, Spark no ha de reexecutar tot el pipeline per a cada processador columnar.

Per a implementar aquest component, ens hem trobat amb dues alternatives, que es poden observar a la Figura 9.

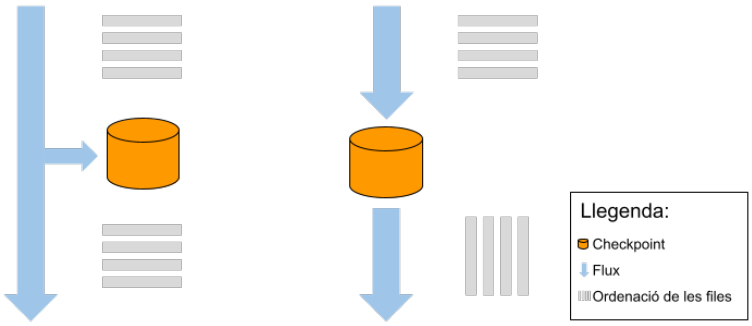


Figura 9: Alternatives de la implementació del *Checkpoint*
[Font: elaboració pròpia]

La primera alternativa, situada a l'esquerra de la Figura 9, consistia en utilitzar la funcionalitat persist() de Spark. Aquesta funció, segons [36], permet escollir el nivell de persistència de l'emmagatzematge. Hi ha diversos nivells a escollir i deferents configuracions. La que s'adequaria millor al nostre cas d'ús és MEMORY_AND_DISK, ja que primer intenta guarda a memòria, si cap, i posteriorment, guarda a disc la part restant. D'aquesta forma fa també més ràpid el *checkpoint* si hi ha suficient memòria RAM. El desavantatge d'aquesta solució és que no podem especificar com s'han de guardar les files. És a dir, si s'han de

guardar per columnes o per files. Aquesta solució només funciona per files.

La segona alternativa que vam tenir es mostra a la dreta de la Figura 9 i es tracta de crear dos fluxos d'execució. El primer flux, s'encarregava de llegir el conjunt de dades cru, parsejar-lo i guardar-lo a HDFS de forma distribuïda. Un segon flux s'encarregava de llegir aquest arxiu i enviar-lo cap als processadors columnars, que cada un llegia una columna. El format més indicat per a guardar i llegir el *dataframe* és en format Parquet, ja que d'aquesta forma, el *dataframe* es guarda per columnes, i no per files, a diferència de la primera solució, fent d'aquesta forma més eficient els processadors columnars ja que només hauran de seleccionar la partició que volen processar i no hauran de llegir informació extra, com passava amb la solució anterior. La desavantatge d'aquesta solució és que és una mica més complicada que la solució anterior, que era només una crida a una funció. En aquesta alternativa, ens hem d'assegurar que realment s'estigui utilitzant HDFS per a guardar el *checkpoint*.

Ens vam decantar per la segona opció en principi, ja que tot i que era una mica més complicada d'implementar, el guany que suposava per als processadors és molt gran, sobretot per als conjunts de dades grans i també per als conjunts de dades de moltes columnes. Tot i així, vam implementar les dues alternatives i vam confirmar que la segona proposta era més eficient que la primera. Per a veure l'anàlisi, vegeu la secció 3.3.10.

3.3.5 Despatxador

Aquest element és l'encarregat de redirigir aquelles columnes cap al tipus de processador columnar correcte en funció del tipus de dada de la columna. Per exemple, per a un columna del tipus *string*, el despatxador entregarà aquella columna al Processador de *string*. Això ho fa per a totes les columnes d'un conjunt de dades. Després de recollir la sortida dels processadors columnars, la envia cap al Formatador.

Tot i que sembli una implementació sofisticada, en realitat és molt simple, ja que hem fet una classe molt modular que s'encarrega de fer el *dispatching* de les columnes en funció dels tipus Aquest és l'únic despatxador específic. Posteriorment, hi ha una altra classe que selecciona columna a columna del *dataframe* d'entrada les columnes i les envia cap al despatxador específic.

També hem creat un *Map* per a mapejar entre el tipus de dada i el processador numèric al que haurà de estar enturada la columna. D'aquesta forma es fa també més extensible aquest component, seguint el *Open Closed Principle*.

3.3.6 Processadors columnars

Els processadors columnars, tal com es va mencionar amb anterioritat, s'encarreguen d'extreure els descriptors estadístics d'una columna.

Processador de tuples

Els descriptors estadístics, per al processador de tuples és el següent:

- **Càlcul de l'entropia.** Aquest càlcul ens permet saber l'entropia de tota la taula, agafant com a tuples totes les columnes i calculant el valor d'aquesta.

Processador numèric

El processador columnar numèric extreu els següents descriptors estadístics:

- **Nombre de files no nul·les.** Aquest descriptor estadístic s'encarrega d'anar element a element d'una columna i veure quantes files tenen valors no nuls.
- **Nombre de files nul·les.** Aquest descriptor estadístic s'encarrega d'anar element a element d'una columna i veure quantes files tenen valors nuls.
- **Nombre de files diferents.** Aquest descriptor estadístic s'encarrega de veure quantes files tenen valors diferents per a aquella columna.
- **Valor mínim.** Aquest descriptor estadístic s'encarrega d'agafar el valor mínim entre tots els valors de la columna.
- **Valor màxim.** Aquest descriptor estadístic s'encarrega d'agafar el valor màxim entre tots els valors de la columna.
- **Mitjana.** Aquest descriptor estadístic s'encarrega d'agafar la mitjana entre tots els valors de la columna.
- **Variança.** Aquest descriptor estadístic s'encarrega d'agafar la variança entre tots els valors de la columna.
- **Desviació estàndard.** Aquest descriptor estadístic s'encarrega d'agafar la desviació estàndard entre tots els valors de la columna.
- **Entropia de la columna.** Aquest descriptor estadístic s'encarrega de calcular l'entropia de la columna seguint la fórmula especificada a l'equació (A.1).

Processador de *strings*

El processador columnar de *strings* extreu els següents descriptors:

- **Longitud mitjana del valor.** Aquest descriptor estadístic s'encarrega d'agafar la mitjana de la longitud de cada valor de la columna.
- **Longitud mínima del valor.** Aquest descriptor estadístic s'encarrega d'agafar el mínim de la longitud de cada valor de la columna.
- **Longitud màxima del valor.** Aquest descriptor estadístic s'encarrega d'agafar el màxim de la longitud de cada valor de la columna.
- **Variança de la longitud del valor.** Aquest descriptor estadístic s'encarrega d'agafar la variança de la longitud de cada valor de la columna.
- **Desviació estàndard de la longitud del valor.** Aquest descriptor estadístic s'encarrega d'agafar desviació estàndard de la longitud de cada valor de la columna.
- **Mitjana d'espais en blanc del valor.** Aquest descriptor estadístic s'encarrega d'agafar la mitjana del nombre d'espais en blanc de cada valor de la columna.

- **Mínim nombre d'espais en blanc del valor.** Aquest descriptor estadístic s'encarrega d'agafar el mínim del nombre d'espais en blanc de cada valor de la columna.
- **Màxim nombre d'espais en blanc del valor.** Aquest descriptor estadístic s'encarrega d'agafar el màxim del nombre d'espais en blanc de cada valor de la columna.
- **Variança del nombre d'espais en blanc del valor.** Aquest descriptor estadístic s'encarrega d'agafar la variança del nombre d'espais en blanc de cada valor de la columna.
- **Desviació estàndard del nombre d'espais en blanc del valor.** Aquest descriptor estadístic s'encarrega d'agafar la desviació estàndard del nombre d'espais en blanc de cada valor de la columna.
- **Mitjana de la longitud de les paraules del valor.** Aquest descriptor estadístic s'encarrega de fer un *flat map* de tots els valors de la columna, i de totes les paraules, agafa la mitjana de la seva longitud.
- **Mínima longitud de les paraules del valor.** Aquest descriptor estadístic s'encarrega de fer un *flat map* de tots els valors de la columna, i de totes les paraules, agafa la longitud més petita.
- **Màxima longitud de les paraules del valor.** Aquest descriptor estadístic s'encarrega de fer un *flat map* de tots els valors de la columna, i de totes les paraules, agafa la longitud més gran.
- **Variança de la longitud de les paraules del valor.** Aquest descriptor estadístic s'encarrega de fer un *flat map* de tots els valors de la columna, i de totes les paraules, agafa la variança de la longitud.
- **Desviació estàndard de la longitud de les paraules del valor.** Aquest descriptor estadístic s'encarrega de fer un *flat map* de tots els valors de la columna, i de totes les paraules, agafa la desviació estàndard de la longitud.
- **Nombre de valors diferents.** Aquest descriptor estadístic s'encarrega de veure la quantitat de valors diferents que hi ha a la columna.
- **Nombre de paraules diferents.** Aquest descriptor estadístic s'encarrega de fer un *flat map* de tots els valors de la columna i, de totes les paraules, mira el nombre de quantes són diferents.
- **Nombre de caràcters diferents.** Aquest descriptor estadístic s'encarrega de fer un *flat map* de tots els valors de la columna i, de tots els caràcters, mira el nombre de quants són diferents.
- **Nombre de files no nul·les.** Aquest descriptor estadístic mira quants dels valors de tota la columna no són nuls.
- **Nombre de files nul·les.** Aquest descriptor estadístic mira quants dels valors de tota la columna són nuls.

- **Entropia de valors.** Aquest descriptor estadístic, mira per a tota la columna, quina és la seva entropia.
- **Entropia de paraules.** Aquest descriptor estadístic fa primer un *flat map* de tots els valor de la columna i calcula l'entropia de totes les paraules.
- **Entropia de caràcters.** Aquest descriptor estadístic fa primer un *flat map* de tots els valor de la columna i calcula l'entropia de totes els caràcters.

Processador de dates

El processador columnar de dates extreu els següents descriptors:

- **Data més vella.** Aquest descriptor estadístic agafa la data més antiga de totes les dates de la columna.
- **Data més nova.** Aquest descriptor estadístic agafa la data més nova de totes les dates de la columna.
- **Mínima diferència entre files consecutives de temps (delta).** Aquest descriptor estadístic primer calcula les delta entre les dates, és a dir la diferència entre els valors de files consecutives a la columna, i agafa la mínima diferència.
- **Màxima diferència entre files consecutives de temps (delta).** Aquest descriptor estadístic primer calcula les delta entre les dates, és a dir la diferència entre els valors de files consecutives a la columna, i agafa la màxima diferència.
- **Mitjana de la diferència entre files consecutives de temps (delta).** Aquest descriptor estadístic primer calcula les delta entre les dates, és a dir la diferència entre els valors de files consecutives a la columna, i agafa la mitjana de totes les diferències.
- **Variança de la diferència entre files consecutives de temps (delta).** Aquest descriptor estadístic primer calcula les delta entre les dates, és a dir la diferència entre els valors de files consecutives a la columna, i agafa la variança de totes les diferències.
- **Desviació estàndard de la diferència entre files consecutives de temps (delta).** Aquest descriptor estadístic primer calcula les delta entre les dates, és a dir la diferència entre els valors de files consecutives a la columna, i agafa la desviació estàndard de totes les diferències.
- **Nombre de files no nul·les.** Aquest descriptor estadístic compta el nombre de valors no nuls que hi ha a la columna.
- **Nombre de files nul·les.** Aquest descriptor estadístic compta el nombre de valors nuls que hi ha a la columna.
- **Nombre de files diferents.** Aquest descriptor estadístic compta el nombre de dates diferents que hi ha a la columna.
- **Entropia de la columna.** Aquest descriptor estadístic calcula l'entropia de la columna.

- **Entropia dels valors delta.** Aquest descriptor estadístic calcula l'entropia de la les diferències de temps entre files consecutives de la columna.

3.3.7 Formatador

El Formatador permet seleccionar per a cada columna, la representació que hi haurà per a un determinat tipus de dades al convertir-lo a cadena de text. Es mostra relativament útil per a determinar per exemple el format de les dates i, en cas que falti algun paràmetre (e.g. l'any), aquest simplement pot permetre determinar per a una determinada columna un format específic, com per exemple el següent: mm-dd HH:MM:SS, on "m" vol dir mes, "d" vol dir dia, "H" vol dir hora, "M" vol dir minut i "S" vol dir segon.

Una altra utilitat pot ser que en un futur es pugui determinar també el format dels nombres amb coma flotant; és a dir, que mostri només dos decimals o tres per als nombres amb coma flotant. Aquest component va sorgir com a necessitat de solucionar un problema de qualitat de dades, on el nostre sistema se inventava l'any quan aquest no existia i apareixia l'any 1900 a l'hora d'analitzar el conjunt de dades Android. Per tant, hem pogut introduir aquest component per tal de separar el valor lògic de les dades del seu format de representació a cadena de text.

3.3.8 Visualitzador

Aquest component és l'encarregat de processar el resultat per tal de poder mostrar-lo d'una forma representable. És a dir, que rep els descriptors calculats i els mostra per pantalla en un format de taula o bé dóna la representació d'una taula en format CSV.

Primer vam fer la implementació de treure per pantalla els resultats, encara que aquesta no era massa eficient a l'hora de convertir les dades a Excel. Per tant, vam fer la implementació del visualitzador que permetia exportar els resultats a un CSV.

Cal destacar que s'han afegit quadres en els resultats que mostra l'eficiència del parsejador; és a dir, el nombre de files que s'han pogut parsejar i el percentatge de files que no s'han pogut parsejar, els resultats columnars, i també la instrumentació de les operacions; és a dir, el temps requerit per a calcular cada descriptor estadístic. Per a la instrumentació de les operacions, hem mostrat per a cada funció la mediana dels temps d'execució, la mitjana, el mínim, el màxim, la desviació estàndard i el nombre de crides. Aquests resultats, els hem utilitzat posteriorment per a fer el tuneig a la secció 3.3.10 i desactivar els càlculs dels descriptors estadístics que més temps d'execució trigaven a la secció 3.3.10.

3.3.9 Taula

Aquest és el resultat de fer el *profiling* del conjunt de dades i serà un arxiu en format CSV que es guardarà a HDFS en el clúster. En el Fragment 3.3 podem veure un exemple del format de la taula.

L'exemple complet es pot trobar a [37].

```
Parser Statistics
;Value
Filtered out rows count;1
Lost rows percentage;3.7653491634636725e-06
Original rows count;26557962

 Statistical profile
;User;DateTime;AdGroupId;PID;NonClk;Clk
NumericResults;******;******;******;******;******;******
Count distinct;1141729;-;846811;-;2;2
Count not null;26557961;-;26557961;-;26557961;26557961
Count null;0;-;0;-;0;0
...
TimestampResults;******;******;******;******;******;******
Count distinct;-;662061;-;-;-;-
Count not null;-;26557961;-;-;-;-
Count null;-;0;-;-;-;-
...
StringResults;******;******;******;******;******;******
Characters entropy;-;-;-;2.95344582469885;-;-
Count blank spaces statistics average;-;-;-;0.0;-;-
Count blank spaces statistics max;-;-;-;0;-;-
Count blank spaces statistics min;-;-;-;0;-;-
...
Execution call tracker
;Value
...
```

Fragment 3.3: Exemple del format de la taula
[Font: elaboració pròpia]

3.3.10 Altres detalls

En aquesta secció veurem altres detalls també importants de la implementació del *Datasets Profiler*.

Execució a *batches*

Per a facilitar haver de tractar tots els conjunts de dades i totes les mostres d'una forma més automatitzada, hem afegit una nova característica al *Datasets Profiler*, que consisteix en declarar un conjunt de paràmetres que es correspondran a diverses execucions, per tal d'automatitzar el procés de càlcul dels descriptors estadístics per a diferents configuracions del *dataset*.

Com a configuracions s'entén la mida d'aquest (limit del conjunt de dades). Això s'utilitza per a fer un *sampling* del *dataset* gran, on s'agafa només una petita porció del dataset per a extreure els descriptors estadístics.

Aquests paràmetres s'especifiquen en un *array* en un arxiu Json i això pot permetre fer múltiples execucions del programa. El format d'exemple que s'utilitzaria seria el que es mostra al Fragment 3.4.

En el Fragment 3.4, podem observar que s'està utilitzant el cas d'ús get_description, que és el que ens permet calcular els descriptors estadístics d'un conjunt de dades. S'especifica com a paràmetre el camí del *dataset* cru, el camí de la sortida dels resultats, en format CSV, el parsejador; en aquest cas és el nombre de la classe en Python, i el límit de files que es llegeixen des del començament del conjunt de dades per a ser processades.

En el segon objecte de Json, que es correspondria a una segona execució del *Datasets Profiler* – la primera execució es correspondria, per tant, al primer objecte en Json – especifica també els formatadors utilitzats. Si no s'especifiquen, com en el primer objecte, per defecte els resultats es converteixen a *strings*. Podem veure com s'especifica el formatador de "no_year_datetime_specific_formatter", que permet que no es visualitzi l'any en els resultats del *Datasets Profiler*, perquè aquestes dades no estan presents en el conjunt de dades cru.

Els dos objectes, estan a dintre d'un *array*, que és el que anomenem *batch*.

Execució Local

Cal destacar el fet que hem creat dos scripts que ens permeten executar el *Datasets Profiler* en mode local o en mode clúster. Aquesta característica ens ha permès estalviar recursos al poder executar el *Datasets Profiler* al portàtil en el que es feia el desenvolupament, sense necessitat de tenir el clúster encès. Tot i així, a l'hora de fer la migració al clúster, ens hem trobat amb una sèrie de problemes, que es detallen en el capítol B dels apèndixs.

IoC Container

Degut a la complexitat creixent de la quantitat de classes a inicialitzar, s'ha optat per l'ús d'un contenidor IoC (contenidor d'Inversió de Control), que permet fer la inicialització

```
[
      {
          "use_case": "get_description",
          "input_path": "/files/datasets_profiler/input/Thunderbird.log",
          "output_path":
              "/files/datasets_profiler/output/Thunderbird_sample_100000000.csv",
          "parser": "thunderbird_log_parser_strategy",
          "limit": 100000000
      },
      {
          "use_case": "get_description",
          "input_path": "/files/datasets_profiler/input/Android.log",
          "output_path": "/files/datasets_profiler/output/Android_sample_100000000.csv",
          "parser": "android_log_parser_strategy",
          "specific_formatters": [
          "no_year_datetime_specific_formatter",
          "string_specific_formatter",
          "string_specific_formatter",
          "string_specific_formatter",
          "string_specific_formatter",
          "string_specific_formatter"
          ],
          "limit": 100000000
      }
]
```

Fragment 3.4: Exemple de fitxer amb paràmetres del Datasets Profiler
[Font: elaboració pròpia]

dels diferents components amb una senzilla i concisa configuració, que s'ocupa de fer la inicialització de tots els components en un ordre adequat.

Aquest component ens ha permès fer la injecció del les dependències d'una forma eficient a mesura que el *Datasets Profiler* anava creixen a nivell de quantitat de codi escrit.

Logs

A part de mostrar els errors a través de la CLI, hem decidit integrar la característica de generar *logs* per tal de poder visualitzar els errors que ocorrin durant l'execució. Cal destacar que durant l'execució del programa per a *datasets* grans, hi ha hagut diversos errors i no s'ha pogut determinar la causa que els provocava fins que es va incloure aquesta característica i es va veure que era degut a la falta de memòria per a processar els *datasets*.

Profiling

Per tal de solucionar els problemes d'eficiència deguts a la quantitat d'operacions efectuades, s'ha realitzat primer un *profiling* utilitzant Pycharm, encara que els resultats no eren massa clars degut a que no es mostraven les classes i funcions que s'estaven executant de forma clara.

Aleshores, vam optar per realitzar una instrumentació en forma de decoradors a les funcions en el programa d'avaluació dels descriptors. L'anàlisi, juntament amb el tuneig dels descriptors estadístics utilitzats es descriuen a la secció 3.3.10.

Selecció de Descriptors Estadístics

També, s'ha introduït la possibilitat de seleccionar quins descriptors estadístics estan activades i quines no a cada processador. Cal destacar, però, que els càlculs d'alguns descriptors estadístics es reaprofiten per a calcular d'altres i, si el valor resultant d'una descriptor estadístic no està present, aquelles que requereixin aquest valor, tampoc es podran calcular, per la qual cosa, pot donar lloc a més valors nuls en els resultats dels inicialment esperats degut a aquestes dependències.

Aquesta configuració es fa de forma estàtica en codi, degut a que acostuma a ser més estàtica que un arxiu de configuració.

Tuneig

Es va observar que el Processador de Dates trigava molt temps quan es comparava tenint tots els descriptors estadístics activats i desactivats per a un conjunt de dades de Test. El conjunt de dades de Test és un conjunt de dades sintètic que vam crear només per a avaluar com funcionava el *Datasets Profiler* amb una quantitat molt reduïda de files (exactament 14 files). Concretament els temps d'execució de la funció "TimestampProcessor.process()" estan a Taula 1.

Cal destacar que les execucions s'han fet a l'ordinador amb les característiques especificades a [38]. Per a cada casella s'han fet 3 execucions i s'ha calculat una mediana d'entre les tres execucions, per tal de considerar un valor real.

Dataset	Nombre d'execució	Temps sense cap descriptor estadístic	Temps calculant tots els descriptors estadístics
Test (14 files)	Execució #1	0,904 segons	3,486 segons
	Execució #2	0,918 segons	3,550 segons
	Execució #3	0,937 segons	3,561 segons
	Mediana	0,918 segons	3,550 segons
Android-sample (1000 files)	Execució #1	0,888 segons	3,563 segons
	Execució #2	0,924 segons	3,543 segons
	Execució #3	0,915 segons	3,523 segons
	Mediana	0,915 segons	3,543 segons
HDFS_1 (11.175.629 files)	Execució #1	1,572 segons	758,164 segons
	Execució #2	1,595 segons	756,742 segons
	Execució #3	1,547 segons	755,197 segons
	Mediana	1,572 segons	756,742 segons

Taula 1: Temps d'execució de la funció "TimestampProcessor.process()"
[Font: elaboració pròpia]

Podem observar clarament que sense cap càlcul dels descriptors estadístics, el processador de dates triga molt de temps (al voltant d'un segon) en no fer cap càlcul. Per tant, es va decidir mirar més a fons la implementació del *lag*, ja que era la única acció que es realitzava independentment de si hi havia algun càlcul de descriptor estadístic activat no. Aquest "*lag*" vol dir que es copia el valor del *timestamp* d'una determinada fila a la següent. Això es fa per tal de poder fer posteriorment una resta entre el valor de la fila i el *lag* i poder determinar la diferència de temps o delta de temps.

Canviant la implementació, s'ha obtingut una millora de l'eficiència en el càlcul d'aquest descriptor estadístic que permet els temps d'execució de la Taula 2.

Dataset	Nombre d'execució	Temps sense cap descriptor estadístic	Temps calculant tots els descriptors estadístics
Test (14 files)	Execució #1	0,483 segons	3,370 segons
	Execució #2	0,518 segons	3,344 segons
	Execució #3	0,497 segons	3,358 segons
	Mediana	0,497 segons	3,358 segons
Android-sample (1000 files)	Execució #1	0,526 segons	3,399 segons
	Execució #2	0,490 segons	3,418 segons
	Execució #3	0,507 segons	3,440 segons
	Mediana	0,507 segons	3,418 segons
HDFS_1 (11.175.629 files)	Execució #1	0,894 segons	56,193 segons
	Execució #2	0,941 segons	57,271 segons
	Execució #3	0,976 segons	57,109 segons
	Mediana	0,941 segons	57,109 segons

Taula 2: Temps d'execució de la funció "TimestampProcessor.process()" després de la millora d'eficiència
[Font: elaboració pròpia]

Per tant, podem observar que s'ha reduït considerablement el temps d'execució per als conjunts de dades, sobretot en el *dataset* amb més files.

Possible millora d'implementació del processador de tuples

Una observació que vam fer a l'hora de processar grans conjunts de dades és que per als *datasets* mitjans i grans, el programa s'aturava quan calculava els descriptors de la taula perquè es quedava sense memòria. La implementació actual és bastant ineficient i, per a processar un *dataset* de 2,7 GiB pot arribar a requerir més de 32 GiB de memòria RAM més uns altres 32 GiB de memòria *swap*. Per tant, la motivació d'aquest apartat és trobar una solució eficient que permeti reduir la càrrega que suposi el processat de les tuples de la taula.

Els detalls de la implementació consisteixen en fer un map de les tuples de la taula i agafar el seu *hash* SHA-512 i emmagatzemar aquest nombre. Per cada columna, tindrem 64 bytes que seran únics. Per tant, dues files iguals, tindran el mateix *hash*. Sempre serà la mateixa

quantitat de bytes, independentment de la grandària de la columna. També cal destacar que aquest mètode no és massa eficient per a *datasets* amb poques files. Aleshores, implementem el mateix algorisme de càlcul de l'entropia, que aplicàvem per a calcular l'entropia de una columna i, trobarem finalment el valor d'aquesta.

Cal notar que aquesta implementació la comentem a efectes de completesa, ja que realment, l'entropia de taula per al tipus de *datasets* analitzats, no aporta massa informació, per tant, tampoc cal realment implementar-la. Això es dedueix pel fet que cada fila té una marca de temps i aquesta és sempre diferent, per tant, no té massa sentit trobar l'entropia de la fila quan té un camp que sempre canvia, perquè aleshores l'entropia serà sempre alta.

La decisió que vam prendre és desactivar el càlcul de l'entropia de la taula ja que ens donava cap informació nova, ja que degut a la naturalesa dels *logs*, totes les taules tenen una columna de DateTime, que va variant sempre i, per tant, fa créixer l'entropia.

Reducció del nombre de descriptors estadístics calculats

Per tal de fer encara més eficient l'execució, s'ha analitzat primer el temps d'execució del càlcul dels descriptors estadístics i després, la seva rellevància. Els descriptors estadístics amb un cost molt elevat, i que no aporten molt valor són els que es mostren a la Taula 3.

Nom de les classes i mètodes que calculen el descriptor estadístic	**Descripció**
string_processor_get_words_length_statistics	Permet obtenir les estadístiques de la longitud de les paraules quan s'analitzen amb el processador de cadenes de text. Les estadístiques volen dir la mitjana, mínim, màxim, variància i la desviació estàndard quan aquestes operacions estan activades.
timestamp_processor_calculate_delta_time_in_seconds_entropy	Permet obtenir l'entropia de la diferència de temps en segons entre files consecutives.

Taula 3: Descriptors estadístics amb un temps elevat
[Font: elaboració pròpia]

Noti que s'ha permès activar i desactivar el càlcul dels descriptors estadístics. Alguns dels quals, si es desactiven, ho faran per a tots els processadors (són transversals). Això vol dir que si tenim dos descriptors, per exemple obtenir la longitud de les paraules i obtenir les estadístiques del delta de temps, i desactivem el càlcul del mínim i màxim, aleshores, aquest càlcul no es realitzarà en cap dels descriptors; aquesta decisió s'ha pres degut al fet que es permet la reutilització del codi i es facilita molt el manteniment d'aquest, al tenir una classe que permet realitzar tots els càlculs en comú (mitjana. mínim, màxim, variància i desviació estàndard), per tant, desactivar algun d'aquests càlculs implica la desactivació en aquesta classe del càlcul, que a la vegada és utilitzada per tots els processadors.

Primer desactivarem una quantitat reduïda de càlculs per tal de veure l'impacte però, primer haurem de tenir un marc de referència, per tant els temps d'execució total dels processadors amb tots els càlculs dels descriptors estadístics activats es mostra a Taula 4.

Després, executant per a diversos *datasets* amb les dues operacions anteriorment mencionades desactivades, s'han obtingut els temps d'execució de l'aplicació que es mostren a la Taula 5.

Tot i així, desactivar només aquestes operacions no suposava una diferència de temps

Dataset	Nombre d'execució	Temps d'execució total dels processadors
Test (14 files)	Execució #1	17,659 segons
	Execució #2	17,384 segons
	Execució #3	17,241 segons
	Mediana	17,384 segons
Android-sample (1000 files)	Execució #1	24,726 segons
	Execució #2	24,726 segons
	Execució #3	24,677 segons
	Mediana	24,726 segons
HDFS_1 (11.175.629 files)	Execució #1	379,414 segons
	Execució #2	378,953 segons
	Execució #3	377,488 segons
	Mediana	378,953 segons

Taula 4: Temps d'execució total dels processadors amb tots els càlculs dels descriptors estadístics activats
[Font: elaboració pròpia]

Dataset	Nombre d'execució	Temps d'execució total dels processadors
Test (14 files)	Execució #1	14,348 segons
	Execució #2	14,372 segons
	Execució #3	14,456 segons
	Mediana	14,372 segons
Android-sample (1000 files)	Execució #1	20,734 segons
	Execució #2	20,806 segons
	Execució #3	20,770 segons
	Mediana	20,770 segons
HDFS_1 (11.175.629 files)	Execució #1	361,654 segons
	Execució #2	363,385 segons
	Execució #3	360,107 segons
	Mediana	361,654 segons

Taula 5: Temps d'execució total dels processadors amb els dos càlculs dels descriptors estadístics desactivats
[Font: elaboració pròpia]

molt gran, per tant, es va procedir a desactivar també altres càlculs de descriptors estadístics menys rellevants que es mostren a la Taula 6.

Un cop descartats els descriptors anteriorment descrits, el rendiment del programa ha sigut el de la Taula 7.

Per tant, podem observar que és molt més eficient ara, és a dir que desactivant les operacions, l'aplicació ara és 4,074 cops més ràpida que abans.

Finalment, hem activat tots els càlculs dels descriptors estadístics, per la qual cosa, apareixeran tots. Això ho hem fet, sobretot perquè per als conjunts de dades petits és més ràpid de calcular per la reduïda quantitat de dades, a més, ens permet tenir una visió més completa tenir quasi tots els descriptors estadístics activats. L'únic descriptor estadístic que hem desactivat per a tots els conjunts de dades és el que calcula l'entropia de taula, ja que no aportava molta informació i el temps que requeria per a calcular aquest valor era molt alt.

Tot i no hem desactivat tots els descriptors estadístics mencionats en aquest apartat, hem deixat per utilitzar aquesta característica per si fos necessari per als *datasets* grans.

Nom de les classes i mètodes que calculen el descriptor estadístic	Descripció
column_statistics_calculate_average_count	Permet obtenir la mitjana i el nombre de columnes que posteriorment s'utilitzaran per a calcular la variància. Aquesta operació no era massa rellevant si tenim el valor màxim, mínim i l'entropia de columna. Hem de tenir en compte que la desviació estàndard no es podrà calcular tampoc.
column_statistics_calculate_standard_deviation	Permet obtenir la desviació estàndar a partir de la variància. Aquest valor no és massa important si disposem de la variància. Aquest descriptor estadístic depèn de la variància.
numeric_processor_calculate_distinct	Permet obtenir el nombre de nombres diferents per a nombres. No és massa rellevant si disposem de l'entropia.
numeric_processor_calculate_not_null_rows_count	Permet obtenir el nombre de files que no són nul·les per a nombres. No és massa rellevant si disposem del nombre de columnes totals a partir del parsejador i el nombre de columnes nul·les.
string_processor_calculate_characters_entropy	Permet obtenir l'entropia dels caràcters de tots els strings. Aquesta operació es fa utilitzant el framework de Spark, al igual que amb les paraules. No és un càlcul que aporti massa informació ja que els missatges tindran un conjunt petit de caràcters que es repeteixin. A més a més, és un càlcul que requereix d'un poder computacional elevat.
string_processor_calculate_not_null_rows_count	Permet obtenir el nombre de files no nul·les per a strings. No és massa rellevant si disposem del nombre de columnes totals a partir del parsejador i el nombre de columnes nul·les.
string_processor_calculate_words_entropy	Permet obtenir l'entropia de les paraules. No aporta massa informació ja que es poden repetir moltes més paraules entre missatges. Per exemple si parlem d'un mateix component descrit per una paraula, aleshores aquesta paraula hi sortirà molts cops, però per a missatges diferents.
string_processor_calculate_distinct_words_count	Permet obtenir el nombre de paraules diferents. No aporta massa informació tal i com s'ha comentat a l'operació de que permet obtenir l'entropia de les paraules.
string_processor_calculate_distinct_characters_count	Permet obtenir el nombre de caràcters diferents. Aquest és un càlcul costós que no aporta massa informació ja que hi haurà un nombre petit de caràcters que s'aniran repetint al llarg de les paraules. És bàsicament la mateixa raó per la qual descartem també el nombre de paraules diferents.
string_processor_get_message_length_statistics	Permet obtenir les estadístiques de la longitud dels strings. Aquest descriptor tampoc aporta massa informació si utilitzem el descriptor de nombre de missatges diferents, ja que aquest últim ens aporta la informació sobre la unicitat dels missatges, i no només quant de grans son els missatges. Dos missatges diferents poden tenir la mateixa longitud. Per tant, per a fer un profiling més representatiu, hem decidit excloure aquest descriptor.
timestamp_processor_calculate_distinct_rows_count	Permet obtenir el nombre de files diferents per a dates. Aquest descriptor no té massa sentit si es tracta de logs, ja que normalment els missatges que apareixen quasi sempre tenen el timestamp diferent i cada cop més gran (l'últim missatges de log és el més nou). Per tant, no ens aporta massa informació.
timestamp_processor_calculate_not_null_rows_count	Permet obtenir el nombre de files no nulles per a dates. No és massa rellevant si disposem del nombre de columnes totals a partir del parsejador i el nombre de columnes nul·les.

Taula 6: Altres descriptors estadístics menys importants descartats
[Font: elaboració pròpia]

Proves empíriques

Per tal d'esbrinar quina alternativa és millor, hem decidit fer unes proves empíriques per a veure el rendiment de cada solució per a diferents *datasets*. Cal destacar que en aquest experiment hem activat tots els càlculs dels descriptors estadístics. Addicionalment, ens hem assegurat que els *checkpoints* generats tenien el format Parquet, tal i com es pot veure a la Figura 10.

Aleshores, els temps d'execució total dels processadors per a les dues configuracions, es detallen a la Taula 8.

Podem observar a la Taula 8 que guardar les dades per columnes i llegir-les subsegüent-

Dataset	Nombre d'execució	Temps d'execució total dels processadors
Test (14 files)	Execució #1	6,974 segons
	Execució #2	6,734 segons
	Execució #3	6,868 segons
	Mediana	6,868 segons
Android-sample (1000 files)	Execució #1	9,553 segons
	Execució #2	9,522 segons
	Execució #3	9,463 segons
	Mediana	9,522 segons
HDFS_1 (11.175.629 files)	Execució #1	88,987 segons
	Execució #2	88,772 segons
	Execució #3	88,325 segons
	Mediana	88,772 segons

Taula 7: Temps d'execució total dels processadors amb els càlculs dels descriptors estadístics menys importants desactivats
[Font: elaboració pròpia]

Dataset	Nombre d'execució	Persist	Dos fluxos diferents
Test (14 files)	Execució #1	17,181 segons	17,987 segons
	Execució #2	17,231 segons	17,986 segons
	Execució #3	17,234 segons	17,171 segons
	Mediana	17,231 segons	17,986 segons
Android-sample (1000 files)	Execució #1	24,867 segons	24,776 segons
	Execució #2	24,887 segons	24,814 segons
	Execució #3	25,317 segons	24,769 segons
	Mediana	24,887 segons	24,776 segons
HDFS_1 (11.175.629 files)	Execució #1	449,310 segons	378,893 segons
	Execució #2	449,320 segons	376,704 segons
	Execució #3	449,137 segons	380,453 segons
	Mediana	449,310 segons	378,893 segons

Taula 8: Temps d'execució total dels processadors les dues alternatives de *checkpointing*
[Font: elaboració pròpia]

ment per columnes no té un rendiment molt millor que utilitzar *persist* per a *datasets* petits, fins i tot és pitjor per al *dataset* de test, encara que aquesta forma de guardar les dades es comença a notar que dóna bons resultats per al dataset de Android-sample, on és 100 mil·lisegons més ràpid. Aquesta diferència es fa encara més evident al utilitzar un *dataset* molt més gran, sent aquesta una mica menys de 100 segons.

També he decidit adjuntar el temps necessari per tal de fer el *checkpointing* a la Taula 9. Això és per a demostrar que no és gratuït fer el *checkpoint*, sobretot amb dos fluxos diferents, que augmenta de forma proporcional a la mida del *dataset* el temps que necessita per a fer el *checkpoint*. Podem observar també que l'operació de *persist* és gairebé constant respecte a la mida del *dataset*. Això és degut a que les dades han cabut a memòria, per tant no ha fet falta enviar res a disc. Tot i així, podem veure que aquesta solució pot tenir problemes d'escalat, ja que si la taula té un nombre considerable de columnes, aleshores, el rendiment empitjorarà degut a que no hi ha particionament vertical, mentre que tot i que la solució dels dos fluxos diferents triga més temps en fer el *checkpointing*, aquesta aconsegueix un rendiment millor en general.

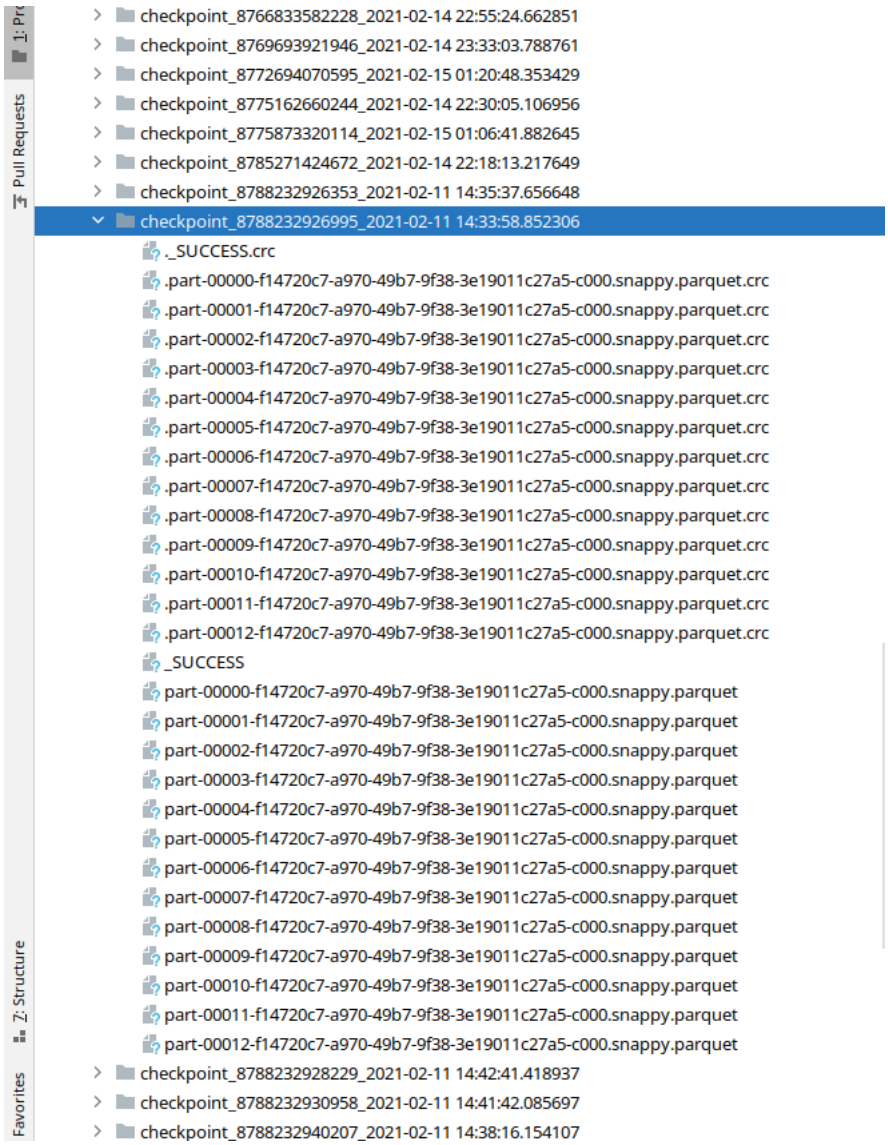


Figura 10: Format dels arxius del *checkpoint*
[Font: elaboració pròpia]

Dataset	**Nombre d'execució**	**Persist**	**Dos fluxos diferents**
Test (14 files)	Execució #1	0,549 segons	1,444 segons
	Execució #2	0,585 segons	1,387 segons
	Execució #3	0,574 segons	1,376 segons
	Mediana	0,574 segons	1,387 segons
Android-sample (1000 files)	Execució #1	0,543 segons	1,632 segons
	Execució #2	0,556 segons	1,455 segons
	Execució #3	0,542 segons	1,477 segons
	Mediana	0,543 segons	1,477 segons
HDFS_1 (11.175.629 files)	Execució #1	0,431 segons	16,140 segons
	Execució #2	0,455 segons	15,775 segons
	Execució #3	0,457 segons	15,959 segons
	Mediana	0,455 segons	15,959 segons

Taula 9: Temps necessari per a fer el *checkpoint*
[Font: elaboració pròpia]

Conclusió

En conclusió, utilitzarem exclusivament dos fluxos amb guardat i lectura d'un arxiu Parquet, per a aprofitar el paral·lelisme i fer més escalable l'aplicació per a conjunts de

dades d'una mida gran.

3.4 Demostració de la correctesa

Per a demostrar la correctesa del *Datasets Profiler*, utilitzarem dues formes. Primer, demostrarem que funciona correctament per a un conjunt de dades determinat i, posteriorment justificarem com hem verificat que eren consistents i seguien sent correctes les diferents versions que hem desenvolupat del *Datasets Profiler*.

Agafarem el conjunt de dades Ad Display/Click Data on Taobao.com sencer per a comprovar la correctesa i verificarem amb un petit script de Python; es pot trobar és informació sobre l'estructura dels repositoris de Github al capítol D dels apèndixs. Segons [39], la funció *describe*, permet veure els valors d'uns pocs descriptors estadístics, amb els que podrem comprovar que ens donen els mateixos resultats que a un perfil estadístic obtingut amb el *Datasets Profiler*.

summary	**User**	**DateTime**	**AdGroupId**	**PID**	**NonClk**	**Clk**
count	26557961	26557961	26557961	26557961	26557961	26557961
mean	568205.4524721984	1.494354798141573...	513017.4537252314	null	0.9485632198947803	0.051436780105219675
stddev	329750.2349182313	1.987552617516365E8	218378.2415685496	null	0.22088693848840144	0.22088693848840146
min	1	1494000000000	1	430539_1007	0	0
max	999999	1494691186000	99999	430548_1007	1	1

Taula 10: Resultat del script en Python utilitzat per a demostrar la correctesa
[Font: adaptació a partir de la taula resultant del script]

En la Taula 10 podem observar el resultat d'executar el script en Python i, en la Taula 11 podem observar també el resultat dels mateixos descriptors estadístics que en la Taula 10 del nostre *DatasetsProfiler*.

summary	**User**	**DateTime**	**AdGroupId**	**PID**	**NonClk**	**Clk**
count	26557961	26557961	26557961	26557961	26557961	26557961
mean	568205.4524721984	-	513017.4537252314	-	0.9485632198947803	0.051436780105219675
stddev	329750.2287101214	-	218378.23745719017	-	0.22088693433378248	0.22088693433378248
min	1	1494000000000	1	-	0	0
max	1141729	1494691186000	846811	-	1	1

Taula 11: Resultat del *Datasets Profiler*
[Font: adaptació a partir de la taula resultant del *Datasets Profiler*]

Observem que els resultats del *Datasets Profiler* són és precisos i, a més, amb el *Datasets Profiler* podem obtenir molts més descriptors estadístics, que no els hem mostrat a la taula Taula 11 per a tenir més claredat a l'hora de comparar. Les dues taules no coincideixen a la perfecció, encara que la majoria dels valors són gairebé exactes. En el resultat del script, observem que es calcula tant la desviació estàndard com la mitjana de la columna DateTime, encara que aquests valors no ens són útils, així que nosaltres no els vam calcular. Una segona discrepància la trobem a l'hora de veure el valor màxim en les columnes User i AdGroupId. Veiem que el *Datasets Profiler* té més precisió que no pas el resultat del script. L'última discrepància la trobem a la columna PID, que nosaltres l'hem tractat com a valors de cadena de text. Per tant, no té cap sentit buscar un mínim o màxim, i és aquesta la raó per la qual no l'hem calculat.

Podem observar que les taules obtingudes del *DatasetsProfiler* i del script en Python contenen els mateixos valors en la majoria dels casos i hem justificat les discrepàncies, per

tant, ja hem demostrat la correctesa del empíricament amb un conjunt de dades de referència.

També, entre les diferents versions del *Datasets Profiler*, vam executar una versió correcta i vam guardar el resultat. Per a les següents versions, generàvem una sortida amb una mateixa entrada i comprovàvem que les sortides seguien sent les mateixes amb un diff.

3.5 Conclusió

Finalment, voldríem mencionar que hem muntat aquest sistema per a poder fer el perfil estadístic d'uns conjunts de dades. En el capítol A dels apèndixs hi ha la discussió de la selecció feta pels experiments que explicarem en el capítol 5.

Capítol 4

Testbed

En aquest capítol ens centrarem en explicar l'eina *Testbed*, la necessitat de construir-la, les requisits, el tipus d'arquitectura utilitzada, veurem l'arquitectura i especificarem els detalls més rellevants de la implementació. Finalment, demostrarem la correctesa d'aquest programa.

4.1 Tipus d'arquitectura utilitzada

Els experiments s'hauran d'executar el mínim nombre possible de cops al clúster, ja que fer-ho comporta un cost que haurem d'optimitzar. Per tant, haurem de tenir una arquitectura que ens permeti optimitzar el cost del clúster pel fet d'executar el mínim de cops possible els experiments.

A diferència del *Datasets Profiler*, que podíem utilitzar un altre ordinador que no suposava un cost molt gran, per a executar els cops que necessitàvem el programa per a obtenir el perfil estadístic d'un conjunt de dades i esbrinar si aquest és correcte o no, en el *Testbed*, un requisit important és que es mesuri amb precisió el temps de les invocacions. Això només es podrà fer en el clúster.

L'arquitectura més adequada per a aquest sistema, de les presentades a la secció 3.1, és l'*EBI (Entity Boundary Interactor)*, que podem observar en la Figura 11, ja que permet produir una arquitectura agnòstica de l'implementació. Aquest tipus d'arquitectura posa un especial èmfasi en el disseny del sistema partint dels casos d'ús que necessitem.

Per a aquest sistema, ja sabíem des d'un començament quins serien els casos d'ús: crear una instrumentació de l'invocació de les operacions i cronometrar el temps i recollir altres mesures de l'invocació de les operacions de forma precisa. Per a complir el requisit de compartir el màxim d'implementació possible per a l'execució dels experiments entre els dos *frameworks*, aquesta arquitectura és ideal, ja que podem especificar en la *Boundary*, els adaptadors de cada operació de forma molt simple.

En aquesta arquitectura, podem fer una abstracció del *framework*, de forma que si nosaltres volem executar una selecció d'un conjunt de dades d'entrada, aleshores, aquesta operació estarà implementada per als dos *frameworks*. En funció de la implementació de l'adaptador

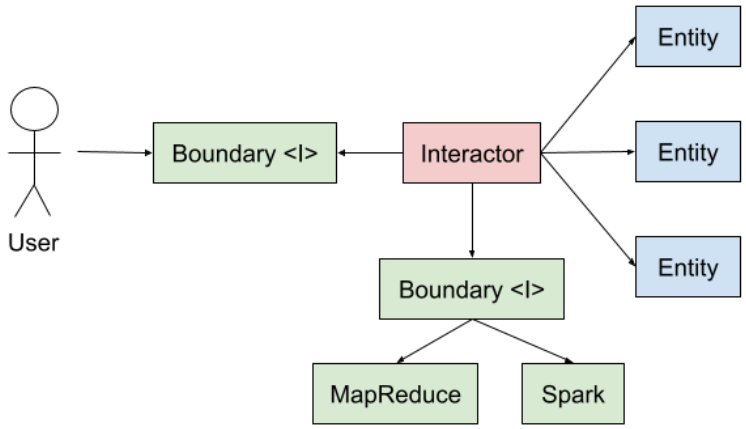


Figura 11: Arquitectura EBI del Testbed
[Font: elaboració pròpia]

que s'utilitzi, s'executarà una implementació o una altra.

Aquest tipus d'arquitectura seria més adequat per a un sistema gran, que depengués de diferents *frameworks* o sistemes externs, que els podríem organitzar tal com proposem a la Figura 11.

La desavantatge més gran de l'*EBI* és que no és un tipus d'arquitectura bo per a arquitectures molt canviants. Per a aquest tipus d'arquitectures, seria més convenient fer servir una arquitectura emergent. Nosaltres teníem ben clar des d'un primer moment quina seria l'arquitectura inicial d'aquest sistema i, aquesta va canviar molt poc al llarg del desenvolupament del codi.

També a l'hora de fer la implementació, haurem de saber en tot moment si la classe que estem utilitzant s'ha de classificar com a *Entity*, *Boundray* o *Interactor*. En alguns casos, la classe no encaixarà en cap dels tres tipus anteriors, per la qual cosa haurem de crear paquets addicionals amb codi. Per exemple, per als esquemes que s'intercanvien entre el *Boundray* i l'*Interactor*. Això vol dir que no es pot seguir sempre de forma estricta aquest model, sinó que en ocasion, s'hauran de fer excepcions, com a l'anterior exemple.

Una arquitectura emergent ens crearia en aquest cas una arquitectura més desordenada, ja que aquesta aplicació sembla a priori que sigui molt gran.

Respecte a l'ús d'una arquitectura de tres capes, aquesta aplicació no seguiria aquest model ja que no necessita cap capa de persistència. Només necessita una entrada i genera una sortida en base a les mètriques que es col·lecten durant l'execució de l'experiment o bé treu a la sortida la instrumentació de les invocacions.

D'aquesta forma, com podem veure, les avantatges d'utilitzar l'*EBI*, respecte l'ús d'una arquitectura emergent o l'arquitectura de tres capes, aporten més valor i compensen les desavantatges, per tant, aquesta serà el tipus d'arquitectura que utilitzarem.

4.2 Arquitectura

Un cop especificat el tipus d'arquitectura que utilitzarem, a partir dels requisits explicarem l'arquitectura del nostre sistema.

Per a complir amb el requisit esmentat a la secció 1.4.3 referent a que hem de tenir un arxiu de configuració en alt nivell que especifiqui les operacions a executar, nosaltres podrem especificar un *pipeline*, amb les operacions en un alt nivell, sense entrar en massa detall respecte els conjunts de dades. Per exemple, per a l'operació de selecció, especifiquem el percentatge de factor de selectivitat de les columnes, però no entrem en el detall de quina serà la cadena de text de la columna que farà que hi hagi un factor de selectivitat similar o exactament igual al esmentat. També fem el mateix per a l'operació de projecció, especificant el percentatge de columnes que es projecten. Hem volgut crear una arquitectura molt genèrica i flexible ja que no sabíem al començament del projecte quins experiments executaríem

Per a entendre millor com hem solucionat aquest problema, haurem d'entrar en més detall en l'arquitectura del *Testbed*.

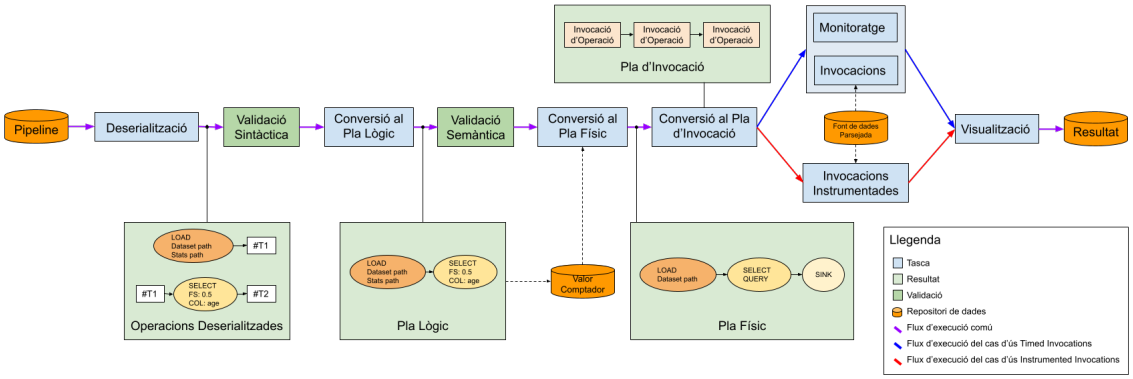


Figura 12: Arquitectura del *Testbed*
[Font: elaboració pròpia]

Si es mirem l'arquitectura de la Figura 12, es pot veure que podríem crear els *pipelines* amb les operacions en ordre, sense necessitar realment totes les diferents tasques que hem creat, encara que vam decidir implementar-les perquè ens proporcionen molta robustesa a l'hora del disseny dels experiments. A la Figura 12, podem observar que tota l'arquitectura es segmenta en diverses parts, que anirem explicant una a una a continuació:

4.2.1 Pipeline

Aquest és un repositori de dades de configuració que defineixi en un alt nivell les operacions. El contingut d'aquest arxiu és exactament el *pipeline* que volem executar.

4.2.2 Deserialització

Aquesta és la primera tasca i és l'encarregada de rebre un arxiu de configuració. Com a resultat d'aquesta tasca, hi ha les operacions deserialitzades en un llistat, tal com podem observar a la Figura 13. Un cop tinguem un llistat de totes les operacions deserialitzades,

juntament amb tots els seus paràmetres, aquestes primer, hi poden haver errors humans com per exemple, que no estigui especificat el valor per a algun camp i, segon estan desordenades.

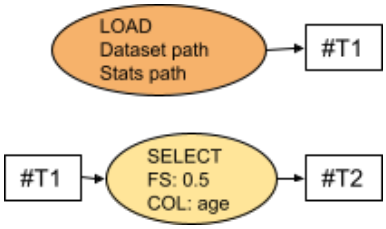


Figura 13: Llistat d'operacions deserialitzades
[Font: elaboració pròpia]

Per a solucionar el primer problema, les alternatives que tenim és fer la comprovació a l'hora de fer la deserialització de les operacions o bé fer-la una una fase de validació separada.

L'avantatge de fer-la en la fase de deserialització és que fa l'arquitectura i la implementació més senzilles, però si volem afegir més validacions, aleshores serà més complicat i podríem estar no respectant el *Open-Closed Principle*, i per tant incurrint en un deute tècnic. En canvi, implementar-lo en un altra fase provocaria que estiguem complicant una mica més l'arquitectura amb l'esperança que en un futur aquest sobre-esforç d'haver de crear una fase més suposi un benefici.

Per a prendre aquesta decisió, ens vam guiar pels bons principis de programació, que eren seguir primer els principis SOLID, i com a segona prioritat, mantenir l'arquitectura senzilla. Per tant, vam prendre la decisió de crear una fase de validació de l'entrada i l'explicarem a continuació.

4.2.3 Validació Sintàctica

En aquesta fase, que es pot observar a la Figura 12, només comprovem que tots els camps de les operacions deserialitzades tinguin un valor que no sigui nul.

Aquesta fase de validació està pensada per si s'han d'afegir més validacions que comprovin al nivell de sintaxis l'entrada de l'experiment.

4.2.4 Conversió al Pla Lògic

Per a solucionar el segon problema que teníem, que era el fet que disposàvem d'un llistat de totes les operacions sense ordre, vam decidir convertir aquest llistat a un pla lògic, convertint les dependències expressades en forma d'etiquetes a una estructura de dades que considerés aquestes dependències, tal com podem veure a la Figura 14.

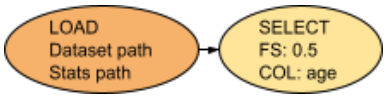


Figura 14: Pla Lògic
[Font: elaboració pròpia]

Aquesta tasca encara manté la informació de quin percentatge de files ha de seleccionar o el percentatge de files que ha de seleccionar, encara que aquestes dades no es poden

proporcionar de forma directa als *frameworks* per a que facin el càlcul, sinó que s'han de transformar especificant exactament quines files o columnes s'han de seleccionar exactament; aquesta serà una conversió que detallarem més endavant.

Tenint totes les operacions organitzades en una estructura de dades que té en compte les dependències, podrem fer més validacions per tal d'estalviar recursos en cas que el *pipeline* sigui incorrecte.

4.2.5 Validació Semàntica

En aquesta tasca, aprofitant que el Pla Lògic està emmagatzemat de forma que es tenen en compte els dependències, podem analitzar de forma eficient la quantitat d'entrades que té cada operació. D'aquesta forma, podrem veure si per exemple, especifiquem una operació que rep dues entrades, però per a les dues entrades té el la mateixa entrada o, per exemple, una de les dues entrades no existeix (perquè no hi ha cap operació que tingui com a sortida una etiqueta amb el mateix nom).

Aquests errors de disseny del *pipeline* també es poden evitar d'una forma fàcil amb aquestes validacions que ens permeten estalviar recursos al impedir que l'experiment s'executi per mitjà d'una fallada i un missatge d'error indicatiu del problema.

4.2.6 Valor Comptador

Aquest és un repositori de dades calculat amb el *Datasets Profiler* que ens dona la informació de quines columnes tenen els conjunt de dades i quines dades té cada columna. Aleshores, es pot convertir a partir d'aquesta informació el valor lògic d'un percentatge al valor físic, que seria per exemple, el valor de la columna que faria que tots els valors lexicogràficament més petits que aquest valor resultarien en el percentatge del factor de selectivitat desitjat, en el cas de la selecció. O el nom de les columnes que s'haurien de seleccionar per a que es satisfaci el percentatge del factor de selectivitat columnar desitjat, en el cas de la projecció.

Cal notar que estem suposant en tot moment que la distribució de les dades en el cas de la selecció és uniforme. Aquesta suposició és necessària per com estem seleccionant les dades i s'especificarà en més profunditat en la implementació.

4.2.7 Conversió al Pla Físic

Un cop tenim el Pla Lògic, aquest té valors per a les seleccions i projeccions que no són suficients per als *frameworks* per a poder executar les operacions. Per tant, ens cal una conversió més per a poder apropar-nos més al tipus d'operacions que executen aquests *frameworks* de processament de dades. Veiem un exemple de sortida a la Figura 15.

Figura 15: Pla Físic
[Font: elaboració pròpia]

Per a aquesta operació, necessitarem els valors i comptadors explicats a la secció 4.2.6.

Un problema que pot sorgir, és que a l'hora de trobar un valor determinar per a la columna, en el cas de la selecció o a l'hora de trobar un conjunt determinat de columnes, en el cas de la projecció, si comparem la sortida amb l'entrada i recalculem el valor real del percentatge de factor de selectivitat i factor de selectivitat columnar, aleshores veurem que hi ha molts cops un error. Nosaltres utilitzarem en tot moment un error relatiu del 5%. L'error relatiu [40] és la diferència entre el valor real i el valor estimat, en valor absolut i dividit pel valor real. Hem de destacar el fet que entenem com a valor estimat el valor proveït pel usuari i el valor real és el que obtenim de l'equació (4.1) [41].

$$\mathrm{SF} = \frac{\text{casos favorables}}{\text{casos possibles}} \tag{4.1}$$

Aquesta comprovació de l'error es farà en aquest punt, és a dir abans de fer qualsevol execució del *pipeline*. Això ho fem per a reduir recursos i fer més eficient l'execució d'un experiment permetent que els experiments que estiguin dissenyats malament fallin abans.

La sortida d'aquesta tasca és un Pla Físic que es pot traduir fàcilment a codi de Spark i codi de MapReduce per a poder fer la invocació de les operacions i que els *frameworks* es puguin encarregar de fer el processament de les dades.

4.2.8 Conversió al Pla d'Invocació

El Pla Físic només és una especificació genèrica de les operacions, que encara estan en una estructura de dades que té en compte les dependències, però no estan organitzades de forma seqüencial per a poder ser executades.

És per això que necessitem aquest Pla d'Invocació, que ens determini una darrere de l'altra l'operació que s'ha d'executar i ho deixi tot preparat per a que es puguin executar les invocacions, tal com podem veure a la Figura 16.

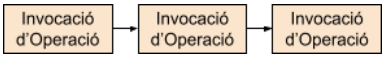


Figura 16: Pa d'Invocacions
[Font: elaboració pròpia]

Aquesta tasca només deixa d'una forma genèrica com quedarien ordenades les operacions, encara que no fa cap invocació, sinó que només crea el Pla d'Invocació, que serà utilitzat per la tasca encarregada de fer les invocacions.

Cal destacar que aquí s'acaba una fase comuna per a dos casos d'ús que ens permet executar aquest *Testbed*, que recordem que és cronometrar l'invocació de les operacions i instrumentar les operacions.

4.2.9 Font de dades Parsejada

Un cop desenvolupat el *Datasets Profiler* i després d'haver-lo utilitzat per a crear el perfil estadístic dels conjunts de dades i haver escollit els que utilitzarem, aquest *software* ja no ens servia per a res més. Després de començar a desenvolupar el *Testbed*, vam veure que aquest necessitava també les dades parsejades i també un petit processament per a fer-lo

una mica més eficient. Per tant, vam decidir reutilitzar tota l'arquitectura dissenyada del *Datasets Profiler*, per a aquest propòsit.

La nova arquitectura es detalla a la Figura 17.

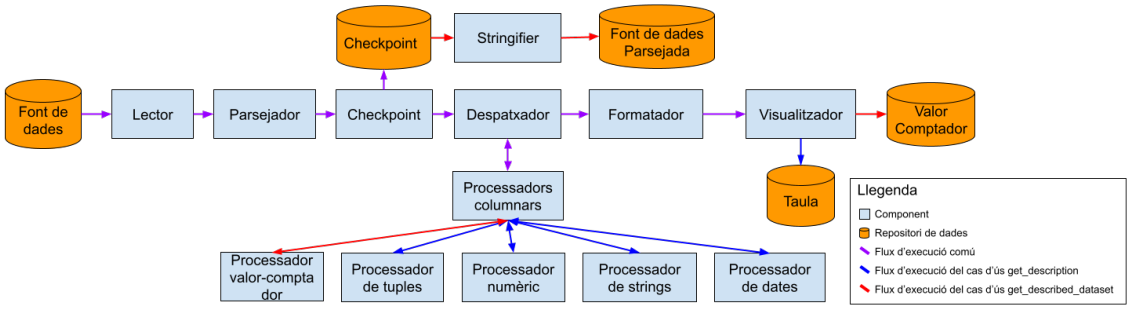


Figura 17: Tercera arquitectura del *Datasets Profiler*
[Font: elaboració pròpia]

Els canvis a nivell d'arquitectura són els següents:

- Tal com es va mencionar a la secció 3.2, es va fer una arquitectura amb una divisió com en els processadors, en una Unitat de Control i una Unitat de Processament. La Unitat de Control s'encarrega de gestionar els elements de la Unitat de Processament i implementa en alt nivell els casos d'ús, mentre que ens referim a l'Unitat de Processament com els components de color blau de l'arquitectura. Els casos d'ús desenvolupats al *Datasets Profiler*, després d'aquest canvi, són "get_description", que ens genera el perfil dels conjunts de dades, i el "get_described_dataset", que ens permet crear a partir d'un conjunt de dades, l'entrada per al *Testbed*.

- Es va afegir un nou processador anomenat processador Valor Comptador. Aquest és bàsicament un processador que per a tots els valors diferents de la columna, compta quants cops apareixen entre els valors de la columna. L'utilitat d'aquesta informació s'explicarà en més detall en la secció secció 4.2.6.

- S'ha afegit un nou component anomenat "Stringifier", que s'encarrega de convertir tots els valors del repositori de *checkpoint* a cadenes de text, també per al *Testbed*.

- La sortida del visualitzador, pot ser també un repositori del tipus Avro, que guarda els valors del conjunt de dades per files, per a empaquetar els valors i comptadors que es generen a la sortida del processador de valors i comptadors. Avro és un format de representació de les dades per files. Es pot trobar més informació a [42].

- La Font de dades Parsejada és un repositori de dades generat per a tot el conjunt de dades, mentre que el repositori de dades valor-comptador està generat per a cada columna del conjunt de dades. Tots dos repositoris formaran l'entrada del *Testbed*. El repositori valor-comptador es genera al mateix temps que el conjunt de dades parsejat i serveix com una estructura de dades que permet fer una acceleració per part del *Testbed*. L'arquitectura d'aquesta estructura de dades es detalla a la secció 4.2.6.

D'aquesta forma no s'han de reprogramar tots els parsejadors per al *Testbed*, sinó que es reaprofita quasi tota l'implementació.

Aleshores, aquesta característica del *Datasets Profiler*, ens permet obtenir la font de dades parsejada.

4.2.10 Invocacions

En quant les invocacions, aquestes s'executen de forma ordenada i eficient, una darrere l'altra tal com es va especificar al Pla d'Invocació. Ens referim a que s'executen de forma eficient al fet que les operacions que estem executant equivaldrien a fer manualment l'execució de les operacions, no com en el cas de l'instrumentació, on a cada operació mirem el nombre de files i columnes que entren i les que surten.

4.2.11 Monitoratge

Mentre es fan les invocacions, es fa un monitoratge utilitzant un servei extern que ens permet recollir mètriques per a poder mesurar i comparar posteriorment el rendiment dels dos *frameworks* de forma rigorosa. Les mètriques que recollim durant les invocacions es detallen a continuació:

- **Distributed file system written bytes (equal to read bytes count) without replication.** Aquesta mètrica indica el nombre de bytes escrits al sistema d'arxius distribuït, és a dir, HDFS. Aquest nombre equival al nombre de bytes llegits per com hem implementat el *Testbed*.
- **Distributed file system written bytes with replication.** Aquest nombre indica el nombre de bytes escrits de forma distribuïda a HDFS amb la replicació. Cal tenir en compte que al clúster hem utilitzar un factor de replicació de 3. Per tant, s'espera que aquest nombre sigui 3 cops el que aparegui a la mètrica "Distributed file system written bytes".
- **Initial instant.** És l'instant de temps en el que es comencen a fer les invocacions.
- **Final instant.** És l'instant de temps final en el que s'acaben de fer les invocacions.
- **Invocation time in nanoseconds.** És el temps en nanosegons necessari per a fer totes les invocacions del Pla d'Invocació. Aquesta mètrica ens determinarà quant de ràpid serà cada *framework* per a processar les dades.
- **Node average memory utilization in bytes.** Aquesta mètrica expressa l'utilització en mitjana en bytes de la memòria RAM del node.
- **Node average swap utilization in bytes.** Aquesta mètrica expressa la quantitat de bytes de memòria d'intercanvi o *swap* en mitjana que s'han utilitzat. La memòria *swap* [43] és una memòria que permet al Sistema Operatiu proveir més memòria a una aplicació que s'està executant de la que està disponible físicament com a RAM. Quan la memòria RAM s'ha exhaurit, el Sistema Operatiu pot utilitzar la memòria d'intercanvi per a obtenir més memòria. Cal destacar que fer un ús d'aquesta memòria penalitza el rendiment ja que implica fer accessos al disc.
- **Node cpu time io wait mode in seconds.** Aquesta mètrica fa referència al temps,

segons [44], expressat en segons que la CPU dedica a esperar per a que es completin les operacions d'entrada i sortida de disc o xarxa. Uns temps grans indiquen que la CPU està encallada degut a les operacions d'entrada i sortida. Per a un rendiment òptim, el valor d'aquesta mètrica hauria de ser baix.

- **Node cpu time system mode in seconds.** Aquesta mètrica indica la quantitat de temps expressada en segons que el processador gasta en executar funcions del Sistema Operatiu.

- **Node cpu time total in seconds.** Aquesta mètrica indica la quantitat total de temps expressada en segons que el processador ha gastat per a executar el programa. Aquesta quantitat de temps és la suma de tots els nuclis del processador.

- **Node cpu time user mode in seconds.** Aquesta mètrica expressa la quantitat de temps en segons que el processador dedica a executar el codi d'aplicació.

- **Node local file system read bytes.** Aquesta mètrica expressa la quantitat de bytes que es llegeixen del disc dur local de cada node del clúster. Cal destacar que els dos *frameworks* estan configurats per a escriure a una determinada carpeta, on hem muntat un dispositiu loopback d'una mida redimensionable. D'aquesta forma, podem recollir mètriques d'escriptura a una carpeta determinada. La tècnica que hem utilitzat està extreta de [45, 46].

- **Node local file system written bytes.** Aquesta mètrica expressa la quantitat de bytes que s'escriuen al disc dur local de cada node del clúster.

- **Node max memory utilization in bytes.** Aquesta mètrica mostra el màxim de memòria RAM expressat en bytes que s'està utilitzant mentre es fan les invocacions de les operacions de forma global a tot el sistema. Cal destacar que estem tenint en compte que ñ'únic programa que està executant el clúster és el *framework*, encara que això no és del tot cert. Tot i així, aquesta dada ens mostra dades reals reportades pel Sistema Operatiu, permetent fer la recollida de mètriques independent del *framework* de processament de dades.

- **Node max swap utilization in bytes.** El valor reportat per aquesta mètrica fa referència a l'ús màxim de memòria d'intercanvi expressat en bytes que s'està utilitzant durant totes les invocacions de les operacions.

- **Node min memory utilization in bytes.** Aquest valor representa el mínim de memòria RAM expressat en bytes que s'està utilitzant durant les invocacions de les operacions.

- **Node min swap utilization in bytes.** El valor representat per aquesta mètrica és el mínim de memòria d'intercanvi que s'està utilitzant durant les invocacions, expressat en bytes.

- **Node network received bytes.** Aquest valor fa referència a la quantitat de bytes que el node ha rebut a través de la xarxa, expressat en bytes. Aquest valor és important ja que la xarxa és l'eina que els nodes del clúster utilitzen per a intercanviar

dades. Aquesta ens permetrà mesurar la quantitat de dades que els nodes es necessiten intercanviar.

- **Node network transmitted bytes.** Aquest valor fa referència a la quantitat de bytes que el node ha transmès a través de la xarxa, expressat en bytes. Aquesta dada mostra la quantitat de dades que es necessita enviar als altres nodes del clúster.

Cal destacar que les mètriques que comencen amb "Node" apareixen per a tots els nodes del clúster, inclòs el node màster. Un altre fet a destacar és que hem decidit preservar el nom original al que ens referim en la sortida en format Excel del *Testbed* per a evitar confusions.

Si mirem amb deteniment les mètriques, observarem que no hi ha cap que sigui específica del *framework* de processament de dades. Això és perquè volíem recollir les mètriques que recolliríem també en un sistema posat en producció, ja que la majoria dels nostres *stakeholders* també utilitzaran els seus sistemes d'aquesta forma. També hem volgut desacoblar-nos de les especificitats dels *frameworks* i de com es calculen de forma desglossada les utilitzacions dels recursos per dos motius: primer, per a fer més robusts els experiments ja que si ens deixem una característica que utilitza la memòria RAM a l'hora de comptar l'utilització total de memòria RAM, per exemple, podria invalidar l'experiment al no estar considerant de forma rigorosa l'utilització. Un segon aspecte és que no volíem pensar en quant reporta cada *framework*, intentant mostrar-nos escèptics, ja que sempre hi poden haver parts que no estiguin documentades o característiques molt subtils dels *frameworks* que no estiguin documentades que puguin afectar a la recollida de les dades. Per tant, hem volgut ser molt rigorosos en aquest aspecte i, vam decidir per aquests motius utilitzar les mètriques de cada node reportades pel Sistema Operatiu.

Tal com veiem a la Figura 12, les invocacions utilitzen com a entrada la font de dades parsejada, provinent de la sortida del *Datasets Profiler* utilitzant el cas d'ús "get_described_dataset". La direcció en el sistema de fitxers distribuït, HDFS, está especificat en l'arxiu de configuració i aquesta informació s'ha anat propagant a través dels diferents plans fins a arribar al Pla d'Invocació, que és el que s'està executant en la tasca d'Invocacions del diagrama de l'arquitectura.

4.2.12 Invocacions Instrumentades

Per a tenir més visibilitat a l'hora d'executar les operacions, sobretot a l'hora de fer el disseny dels experiments i jugar amb els paràmetres fins a trobar els adequats que permetessin generar uns experiments interessants, vam crear el cas d'ús que ens permetés veure les operacions instrumentades. Cada operació està embolcallada d'una altra operació que fa un recompte de files i columnes que entren a l'operació i les que surten i guarda la informació d'aquest procés per a visualitzar-la en un pas posterior.

Les operacions instrumentades permeten observar la quantitat real de files i columnes que s'estan processant i que es processaran quan estiguem monitorant el *Testbed*, a l'hora d'executar les invocacions sense instrumentar. Ens pot ajudar a comprovar que l'error relatiu que s'està comprovant no supera el llindar del 5%. També ens ha servit a l'hora de debugar els dos *frameworks* per a comprovar que es processaven exactament el mateix nombre de files

i columnes per a totes les operacions implementades.

4.2.13 Visualització

Aquesta tasca consisteix en agafar les dades dels anteriors casos d'ús i transformar-les i disposar-les en un arxiu d'Excel que es pugui després d'una forma molt fàcil obrir, interpretar i ampliar, per exemple afegint més gràfiques o fórmules.

4.2.14 Resultat

Aquest és un repositori de dades que conté tota la informació visualitzada. Organitza tota la informació en taules i té un format *user-friendly*.

4.3 Detalls d'implementació

En aquesta secció mostrarem els detalls d'implementació dels components i raonarem sobre les decisions preses a l'hora de fer la implementació dels diferents components que formen el *Testbed*.

4.3.1 Pipeline

El contingut d'aquest arxiu és exactament el *pipeline* que volem executar. Cal destacar que hem escollit el format Json per la seva llegibilitat i facilitat de manipular. Mostrarem primer la forma que tenen aquests arxius per a entendre la necessitat d'aquesta deserialització. El format d'un arxiu de configuració que conté el *pipeline* es detalla a Fragment 4.1.

Observem al Fragment 4.1 que està compost per tres objectes Json. Podem destacar de la seva estructura que aquests objectes es poden especificar en qualsevol ordre en l'*array* en Json, però s'ha de preservar un ordre en les etiquetes. Per la definició de *pipeline*, on totes les tasques estan connectades en sèrie i on la sortida d'un element és l'entrada del següent, hem creat aquesta "unió" dels elements a través del que es coneix amb etiquetes. Aquestes etiquetes o *tags*, permeten especificar la relació entre dues operacions.

Per exemple, si una operació necessita com a paràmetre un *outputTag*, aleshores, això vol dir que l'operació pot emetre les dades cap a una altra operació. Aquesta etiqueta es pot utilitzar o no. Per a utilitzar aquesta etiqueta, s'haurà d'especificar en el camp *inputTag* de l'operació que vulguem executar a continuació. D'aquesta forma, s'estableixen unes dependències entre les diferents operacions que volem executar. D'aquesta manera, aquestes es pot especificar de forma còmode, a més, tenim la flexibilitat de poder especificar les operacions en qualsevol ordre, i d'una forma molt clara.

Addicionalment, al Fragment 4.1 podem observar com cada operació té els seus propis paràmetres. Per exemple, l'operació de *Load*, té com a paràmetre el input del *pipeline*. Aquest paràmetre fa referència al directori que ha resultat d'executar el cas d'ús "get_described_dataset" en el *Datasets Profiler* per a un conjunt de dades determinat. Les operacions implementades per a cada sistema s'expliquen en més detall en el capítol C dels apèndixs.

```
[
      {
          "operation": "LOAD",
          "outputTag": "dataset",
          "datasetDirectoryPath": "input/Ad_click_on_taobao"
      },
      {
          "operation": "SELECT",
          "inputTag": "dataset",
          "outputTag": "selectedDataset",
          "rowsSelectivityFactor": 0.5,
          "columnName": "DateTime"
      },
      {
          "operation": "PROJECT",
          "inputTag": "selectedDataset",
          "outputTag": "projectedDataset",
          "columnsSelectivityFactor": 0.5
      }
]
```

Fragment 4.1: Exemple de pipeline
[Font: elaboració pròpia]

4.3.2 Deserialització

Aquesta tasca es va implementar de forma reutilitzable i fàcilment ampliable a través d'anotacions utilitzant la classe ObjectMapper. Aquesta classe permet una gran flexibilitat a l'hora de deserialitzar en format Json les classes. La sortida d'aquesta classe és una llista amb totes les operacions deserialitzades.

En quant a l'arxiu de configuració, teníem diferents alternatives per a utilitzar. La primera d'aquestes era utilitzar un format no estàndard que consistia bàsicament de crear nosaltres una estructura per a totes les operacions. L'avantatge era que potencialment podia augmentar la relació de senyar-soroll. Aquesta relació fa referència, segons [47], a la proporció existent entre la quantitat d'una senyal i la quantitat de soroll que la corromp. Com a soroll interpretem aquella informació que no ens aporta cap informació a nosaltres, però sí que serviria a l'hora de deserialitzar aquella informació. Creant nosaltres un format determinar, podríem controlar aquesta relació. Tot i així, cal destacar que suposaria un esforç més gran que utilitzar un format estàndard perquè s'hauria d'implementar el component encarregat de la deserialització. Addicionalment, aquesta deserialització s'hauria de documentar molt bé per a que si els nostres *stakeholders* volen utilitzar aquest *Testbed*, sàpiguen com fer-ho.

L'avantatge d'utilitzar un format estàndard com és el Json, ens permetria invertir més temps en altres components de la implementació, a més a més que el format seria entès automàticament per moltes persones, sense necessitar documentació sobre com funciona el format. A més a més, cal destacar que hi ha molts deserialitzadors presents en el llenguatge que implementarem el *Testbed*, que és Java. Només cal remarcar el fet que a l'hora d'especificar les operacions, aquestes tindran caràcters que no aportaran cap informació útil per a un lector que intenti llegir l'arxiu de configuració.

La decisió que vam prendre és utilitzar el format Json, perquè vam trobar que les avantatges esmentades anteriorment superaven les desavantatges. Addicionalment, podem fer èmfasi en el fet que tot i que aquest format tingui caràcters utilitzats a l'hora de fer la deserialització, que no ens aporten cap informació a nosaltres, aquests, no són massa invasius en l'arxiu de configuració com per exemple en el cas de XML. Utilitzant un deserialitzador ja existent per a aquest format, vam estalviar temps que després vam poder invertir en la implementació d'altres components.

4.3.3 Validació Sintàctica

Respecte a aquest tipus de validació, cal destacar el fet que per a totes les operacions deserialitzades, es comprova que tots els camps tenen un valor no nul. Això vol dir que si en l'arxiu de configuració Json no hem posat un camp, aquest estarà automàticament a nul. Per tant, aquesta validació ens permetrà detectar aquest camp que falti i informar-ho per mitjà d'una excepció.

Per a llençar l'excepció, utilitzem precondicions, del paquet com.clearspring.analytics.util. Aquestes ens permeten especificar un missatge d'error de forma llegible que es mostrarà en cas que no es compleixi el requisit que tots els camps no siguin nuls.

Cal destacar també que aquesta validació automàticament mira quins camps té cada

operació. D'aquesta forma no s'ha d'especificar quins camps té cada operació i ens permet no haver de preocupar-nos de modificar aquesta classe cada cop que afegim nous camps per exemple en la deserialització de l'operació.

4.3.4 Conversió al Pla Lògic

Aquesta conversió està composta per un *manager* i per convertidors específics per a cada operació. Cada convertidor específic s'encarrega de transformar una operació deserialitzada en una operació lògica. En les operacions de *Group By*, *Join*, *Load*, *Select*, *Project* i *Union*, es tracta de copiar el valor dels atributs exceptuant les etiquetes que indiquen les dependències. Addicionalment, a cada operació s'assigna un identificador a partir de les etiquetes i un prefix, que es podrà utilitzar més endavant pels *frameworks* per a identificar la sortida d'una operació. Totes aquestes operacions implementen una interfície que ens permet abstraure de la implementació necessària per a cada operació.

El *manager* s'encarrega de crear un graf a partir de la informació obtinguda a partir de la llista d'operacions deserialitzades. Primer de tot, s'utilitzen les etiquetes d'entrada de cada operació deserialitzada per a identificar-la. Aleshores, cada operació s'afegeix a un Multimap. Un Multimap és una estructura de dades similar a un Map amb l'afegit que múltiples elements poden tenir les mateixes claus [48]. Es pot veure un exemple d'aquesta estructura de dades a la Figura 18. Això ens permet que es pugui utilitzar una mateixa operació de *Load* per a dues operacions diferents, que rebrien la mateixa etiqueta com a entrada, per exemple. Aleshores, en el Multimap, s'utilitza una de les entrades com a clau i com a valor s'utilitza l'operació deserialitzada. Naturalment, les operacions que no tenen entrades, és a dir, els *Loads*, les haurem de tractar de forma separada. Si una operació té dues entrades, s'afegirien al Multimap dues entrades, una per a cada etiqueta.

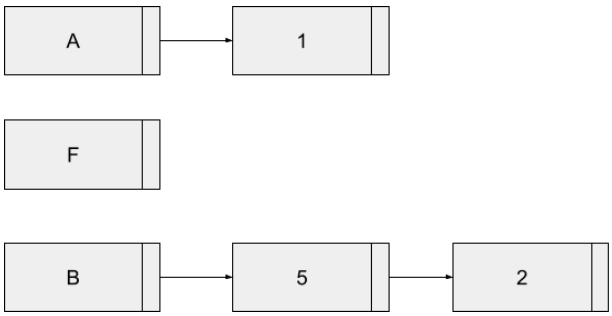


Figura 18: Exemple d'un Multimap
[Font: elaboració pròpia]

Aquest Multimap ens servirà per a consultar les operacions que depenen de l'operació que estiguem processant. Per exemple, si agafem d'una operació deserialitzada qualsevol l'etiqueta de sortida i la utilitzem per a consultar en el Multimap, obtindrem totes les operacions que utilitzen la sortida de l'operació que estiguem analitzant.

Tal com vam dir, les operacions de *Load*, s'han de processar de forma separada, i obtenir un llistat d'aquestes. Addicionalment, hem creat també un Map per a mapejar entre les operacions deserialitzades i les operacions lògiques, la rellevància d'aquesta estructura de dades s'entendrà millor més endavant.

Un cop tinguem totes les dades, ja podem recórrer totes les operacions, començant pel llistat de les operacions de *Loads*, per a crear un graf que tingui en compte les dependències. Les alternatives que teníem en aquest punt era si implementar l'algorisme DFS o BFS per a travessar el graf. Es pot veure visualment el recorregut que fan els dos algorismes l'exemple de la Figura 19.

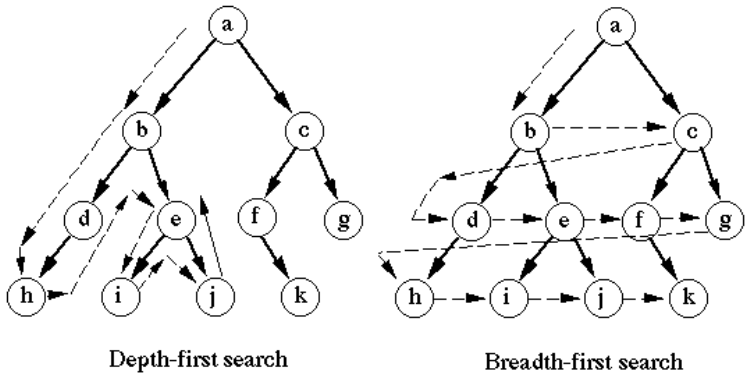


Figura 19: Exemple del recorregut d'un graf utilitzant els algorismes DFS i BFS [Font: [49]]

L'algorisme BFS (*Breadth First Search*) [50] consisteix en travessar el graf en amplada. Això vol dir que primer es visitarà un vèrtex, després es visitaran tots els veïns d'aquests vèrtex, després tots els veïns dels veïns i així successivament fins que tots els vèrtexs s'hagin visitat un cop. En el cas del DFS (*Depth First Search*) [51] primer es comença en un vèrtex, després, s'agafa un dels veïns del vèrtex i s'explora només un veï d'aquest vèrtex i així successivament fins que tots els vèrtexs s'hagin visitat un cop. En els dos casos, cal especificar que la complexitat temporal és la mateixa, encara que en el cas del DFS, la implementació pot resultar més elegant i llegible implementant-ho de forma recursiva, encara que es pot fer també de forma iterativa. En el cas del BFS, és millor implementar l'algorisme de forma iterativa.

La decisió que vam prendre és implementar de forma iterativa el BFS. Realment, no hi havia una gran diferència entre implementar DFS i BFS a nivell de cost de forma iterativa, ja que estaríem utilitzant memòria Heap. Si haguéssim implementat l'algorisme DFS de forma recursiva, ens haguéssim vist més limitats perquè a Java hi ha un límit de recursivitat, que sol estar baix. A [52], podem observar que la mida de la Stack en Java varia entre els 320 KiB i 1MiB. En canvi, la memòria Heap és una zona de memòria compartida que pot tenir valors molt més grans que els mencionats amb anterioritat.

Abans de començar l'execució de l'algorisme, encolarem totes les operacions de *Load* deserialitzades. Mentre la cua no estigui buida, agafarem un element de la cua i mirarem l'etiqueta de sortida i obtindrem totes les operacions deserialitzades que succeeixen aquesta operació. Aleshores, encolem totes les operacions que no hem visitat de moment i afegim arestes al graf. El graf conté les operacions lògiques i fem ús del Map mencionat amb anterioritat per a poder canviar de les operacions deserialitzades a les operacions lògiques.

Un cop tinguem tot el graf recorregut, ja podem crear el Pla Lògic, que consisteix del graf i de les operacions de *Load* lògiques.

4.3.5 Validació Semàntica

Un cop tinguem calculat el Pla Lògic, ja podem comprovar si cada operació té la quantitat d'entrades necessària. Hi ha exactament tres tipus d'operacions, si les classifiquem segons el nombre d'entrades: les operacions que no reben cap entrada; com per exemple l'operació de *Load*, les operacions que reben només una entrada, també anomenades operacions unàries; com per exemples l'operació de selecció i, finalment, les operacions que reben dues entrades, també anomenades operacions binàries; com per exemple, l'operació de *join*. Es pot donar el cas que per a una de les entrades de la *join* haguem especificat una etiqueta que no existeix. En aquest cas, el node de la *join* en el Pla Lògic, només tindrà un successor i no dos.

Recorrent tot el llistat de nodes del graf podrem determinar si hi ha algun que no compleixi la condició mencionada amb anterioritat i, en aquest cast, emetre una excepció juntament amb un missatge. El procediment per a emetre l'excepció és el mateix que es detalla a la secció 4.3.3.

4.3.6 Valor Comptador

Aquest repositori de dades és realment una carpeta amb parts en format Avro, per a cada columna del conjunt de dades i conté tots els valors possibles de la columna en format de cadena de text i ordenats de forma lexicogràfica, juntament amb un comptador del nombre de cops que ha aparegut cada element. S'obté com a sortida del *Datasets Profiler* quan s'executa el cas d'ús "get_described_dataset". El comptador no l'utilitzarem per a res en l'algorisme que ans ajudarà traduir de les operacions lògiques a les operacions físiques, però té una gran importància a l'hora de fer la planificació dels experiments. Sabent exactament quants cops es repeteix cada valor, podem calcular la quantitat de files que tindrà l'operació de la *join*, i això ens permetrà tenir un millor control sobre els experiments que dissenyem. És per això que vam decidir deixar el Comptador.

4.3.7 Conversió al Pla Físic

Un cop tinguem el Pla Lògic, tal com vam mencionar a la secció 4.2.7, les operacions lògiques encara en no ens proporcionen la informació necessària per als *frameworks* per a processar les dades.

Per tant, necessitem una de les entrades del *Testbed*, que és la carpeta de Valor Comptador, on la seva implementació està explicada a la secció 4.3.6.

El Valor ordenat de forma lexicogràfica actua com si fos un índex. Aleshores, podem trobar, a partir del factor de selectivitat, que és un número del 0 al 1, el valor exacte que ens permeti obtenir a la sortida de l'operació de selecció aquest factor de selectivitat. És com si estiguéssim intentant resoldre una equació, intentant trobant el valor exacte que fa que obtinguem un resultat determinat. En l'arxiu de configuració expressem el que volem obtenir a la sortida i el *Testbed* és el que ha de fer els càlculs necessaris per a poder satisfer les nostres necessitats. Sabem la quantitat de files totals que hi ha a cada columna, el nombre de files diferents, i si tots els valors són únics. Aquests valors estan en forma de metadades, en format Avro, en el la carpeta que conté les diferents parts en format Avro amb els Valors i Comptadors de la columna i té el nom de "metadata.avro". Una pregunta que es podria fer

el lector en aquest moment és perquè no hem utilitzat un format com per exemple Json, que sigui més llegible. La resposta seria que en aquest cas, seria que no ens aportaria realment cap valor que fos llegible ja que no utilitzaríem aquesta dada per a res. A més, volíem que el format fos homogeni amb la resta de les parts, que també tenien el format Avro.

El càlcul que es fa en l'operació de selecció per a determinar el valor exacte de la columna que fa que tots els valors més petits siguin filtrats a la següent operació, és el següent: primer s'agafa el nom de totes les parts a la carpeta en format Avro i s'ordena lexicogràficament. Es multiplica el factor de selectivitat pel nombre de parts que hi hagi i s'agafa aquella que correspongui. Això provocarà un determinat error. Cal mencionar que cada part Avro, resultant del *Datasets Profiler* pot tenir una o diverses files i la raó per la que es trobin en diferents parts és perquè Spark particiona les dades per a millorar el rendiment a l'hora de llegur reduint l'entrada i sortida de disc [53]. Aleshores, hem de calcular la fila exacta a dintre de la part. Això es fa estimant primer quantes files tindria aproximadament cada part. Aleshores, es calcula una diferència del factor de selectivitat que obtindríem utilitzant la primera fila de la part i el factor de selectivitat desitjat per l'usuari. Aquesta diferència la multipliquem per la quantitat de files estimades a cada part i obtenim l'índex a dintre de la part. Cal destacar el fet que aquestes són operacions numèriques, és a dir que no hem fet cap consulta a disc, per la qual cosa aquestes es farien de forma molt ràpida. Però un cop disposem de la fila exacta a dintre de la part sí que necessitem accedir a disc per a obtenir el valor de la columna. El que fem aleshores és obtenir un lector d'Avro i saltem una quantitat de files igual al valor de l'identificador de la fila a dintre de la part. Aleshores, podrem obtenir el valor de la fila que determinarà la consulta que haurem de fer per a obtenir el factor de selectivitat desitjat. El motiu d'haver se saltar les files en comptes d'anar directament és perquè no hi ha cap operació d'Avro que ens permeti accedir a l'índex de forma directa.

Vam dissenyar aquesta operació per a que sigui el més ràpida possible i és per això que els detalls semblen complicats. Per a entendre millor la necessitat d'aquesta complexitat, analitzem la quantitat de dades que hem de processar. Si un conjunt de dades que vulguem processar, com per exemple Thunderbird, s'ha d'agafar el valor que fa que tots els valors més petits lexicogràficament que aquest siguin agafats, o dit d'una altra manera, tenir un factor de selectivitat de 1, aleshores fem la suposició que aquest índex que hem creat no existeix. El que hauríem de fer es escollir a partir del conjunt de dades parsejat la columna que ens interessa i fer un escaneig per tota la columna guardant tots els valors per a poder fer una consulta de la següent forma: x = 'A' or 'B' or 'C', on x és el valor de la columna i 'A', 'B' i 'C' són diferents valors de la columna que ens permeten obtenir el factor de selectivitat desitjat. Si la columna té molt pocs valors diferents, aleshores aquests cabran a memòria i el programa, encara que tardi una estona, podrà arribar a computar el valor. Però, si es dona el cas en que la quantitat de valors diferents és de l'ordre del nombre de files que té la columna, aleshores tenim un problema, perquè estarem bàsicament carregant tots els valors diferents d'un conjunt de dades a memòria d'un node, que és el que estigui executant aquest càlcul. Aquest càlcul, a més d'ineficient, tampoc funcionaria. La necessitat d'utilitzar una consulta amb "or" en comptes de "<" és pel fet que no sabem a priori quins valors hi ha a la columna, per la qual cosa, suposem una distribució uniforme i anem agafant els valors diferents de la fila. També ens caldria saber el nombre de repeticions mitjanes que hi hauria

per a cada valor i, per a això hauríem de dividir el nombre de files totals pel nombre de files diferents, per a obtenir una estimació. Un problema a l'hora de treballar amb *Big Data* és que hem de ser força conscients en tot moment de com poden escalar les dades a l'hora de ser processades, perquè un node no hauria de processar tot el conjunt de dades sencer, perquè necessitaria massa recursos. És per això la necessitat de la complexitat d'aquest algorisme. A més, aquesta operació s'ha de fer de forma ràpida ja que s'haurà d'executar abans de cada execució de l'experiment en el *framework* de processament de dades.

Un cop hem obtingut la consulta de la selecció, comprovarem si l'error relatiu és menor que un cert llindar que especificarem com un paràmetre del programa. Si aquest requeriment no es compleix, el programa fallarà i ens avisarà que l'error que hem comés és més gran o igual al límit posat. Per tant, l'error comès haurà de ser sempre més petit que l'error tolerable indicat en forma d'argument al programa.

En el cas de l'operació de projecció, es mira les columnes en el perfil estadístic. Es compta quantes hi ha i es multiplica aquest nombre pel factor de selectivitat columnar especificat. Aquest nombre, agafant la part entera, ens determinarà quantes columnes agafar entre els noms de les columnes que es trobaran ordenades lexicogràficament. La raó d'ordenar lexicogràficament les columnes és per a tenir un criteri a l'hora de saber quines columnes sortiran a l'hora de fer el disseny dels experiments, ja que algunes operacions posteriors poden estar utilitzant una columna de les que estiguin projectades.

Després de saber el nom exacte de les columnes a projectar, podem comprovar l'error al igual que fem en l'operació de selecció, mirant que l'error comès no superi o sigui igual a l'error especificat com a argument del programa.

La resta d'operacions es tradueixen de forma senzilla, copiant només els valors d'un objecte a un altre. La raó de no reutilitzar els mateixos objectes és que ens permetia fer una divisió lògica, a nivell conceptual de com estàvem utilitzant els objectes. El concepte a seguir en el Pla Lògic és que les operacions no són directament aprofitables pel *framework* de processament de dades i necessiten una determinada adaptació. Les operacions en el Pla Físic ja estan en un format genèric interpretable pel *framework* de processament de dades. Hem volgut que el fet de segur l'arquitectura d'esquerra a dreta sigui una tasca fàcil i intuïtiva i fer que en el Pla Lògic apareguin operacions físiques no tenia massa sentit, per tant, tot i que sembli que no s'estigui fent res a nivell de codi, aquest canvi d'operació lògica a física sí té un sentit conceptual.

Al igual que en la secció 4.3.4, aquesta conversió està composta per un *manager* i per convertidors específics per a cada operació. Els convertidors específics són els que a hem detallat fins ara. El *manager* és un component que s'encarrega d'orquestrar els conversors específics per a poder crear el Pla Físic. El *manager* primer crea un llistat de les estimacions del perfil de totes les operacions de *Load* que estan emmagatzemades en el Pla Lògic. Les estimacions del perfil són una estructura que empaqueta l'operació lògica, el perfil del conjunt de dades, el camí a la carpeta que conté totes les subcarpetes amb els valors comptadors de les columnes i el conjunt de dades parsejat resultants del *Datasets Profiler*, i el valor del llindar de l'error relatiu. Aquesta estructura és una estructura intermèdia que només utilitzem en aquesta conversió. Aquesta estructura intermèdia ens servirà per a construir el

graf per a poder crear el Pla Físic.

L'algorisme que hem seguit en aquesta implementació per a construir el graf per al Pla Físic ha sigut el DFS de forma iterativa, encara que com hem mencionat amb anterioritat, no hi hauria cap diferència a nivell de complexitat temporal si el comparem amb el BFS. A nivell d'implementació també són molt similars, ja que en el cas del BFS s'utilitza una cua i en el cas del DFS s'utilitza una pila. El graf el comencem construint afegint totes les estimacions de perfil de les operacions de *Load* calculades anteriorment a una pila. Aleshores, extraiem la primera operació i obtenim totes les poeracions lògique successives del graf d'operacions lògiques que hi ha en el Pla Lògic. Per a totes les operacions, calculem les estimacions dels perfils i afegim els perfils estadístics que encara no hem visitat a la pila. Aleshores, convertim del perfil estadístic de l'operació actual en l'operació física. Per a tots els perfils estadístics de les operacions successives, les convertim a operacions físiques i afegim una aresta en el graf que utilitzarem posteriorment per a crear el Pla Físic. Si una operació lògic no té operacions successives, aleshores afegim una operació física anomenada *Sink*. Aquesta ens servirà per a l *framework* Spark per a provocar que faci una execució de totes les operacions invocades.

Un cop tinguem el graf de les operacions físiques, cal transformar les operacions de *Load* de lògiques a físiques, Un cop tinguem aquestes dues informacions, és a dir el graf i les operacions de *Load* físiques, ja podem crear el Pla Físic.

4.3.8 Conversió al Pla d'Invocació

Un cop tinguem el Pla Físic, ja tenim tota l'informació que un *framework* necessitaria per a fer les invocacions. La feina d'aquesta tasca és justament transformar el Pla Físic en un llistat d'operacions per a que sigui senzill iterar després sobre aquest llistat i poder fer les invocacions. Això ens estalviarà haver de travessar un arbre per a fer les invocacions i permetrà que a l'hora de mesurar el temps, no afegim un *overhead* significatiu. El llistat d'operacions seria tal que consideraria les dependències de les operacions.

Aquesta tasca, implementa l'algorisme d'ordenació topològica per a ordenar totes les operacions per a que considerin les dependències i poder crear el llistat d'operacions. L'ordenació topològica [54] és un algorisme que ens permet ordenar de forma lineal els vèrtexs (en el nostre problema serien operacions) de tal forma que per a cada aresta dirigida del vèrtex u a v, el vèrtex u apareix abans que v en l'ordenació. Es pot veure un exemple d'ordenació topològica en la Figura 20. Aquest algorisme no és possible si el graf no és un DAG (Graf Dirigit Acíclic). El graf del Pla Físic és dirigit ja que hem anat afegint totes les arestes dirigides i és acíclic ja que no té sentit que una operació retroalimenti a una altra que estigui en una part anterior del *pipeline*. No es comprova que passi aquest esdeveniment, encara que tindrem en compte a l'hora de dissenyar els experiments i ens assegurarem que això no passi.

Aleshores, podem crear una ordenació topològica. L'algorisme consisteix en primer crear un Map de les operacions físiques a un comptador del nombre d'entrades que tenen. Després es mouen totes les operacions físiques sense dependències a una pila. Mentre la pila tingui elements, agafem la primera operació física de la pila i creem un objecte d'invocació d'operació i l'afegim a un llistat. Aleshores, decrementem en 1 el nombre comptador de de-

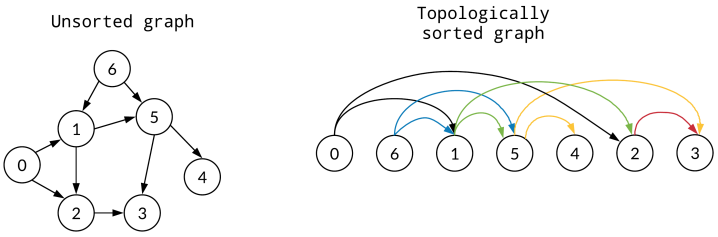


Figura 20: Exemple d'ordenació topològica
[Font: [55]]

pendències d'entrada per a totes les operacions successives i movem totes les operacions sense dependències a la pila i repetim aquest procediment sense que ens quedem sense operacions a la pila.

Cal recalcar que en aquest moment, les invocacions de les operacions no són exactament el codi Spark o MapReduce sinó que és un objecte que conté tota la informació per a fer la invocació en un dels dos *frameworks*.

4.3.9 Font de dades Parsejada

Aquesta és en realitat una carpeta que conté a dins parts en format Parquet i representa el conjunt de dades parsejat pel *Datasets Profiler* després d'executar el cas d'ús "get_described_dataset", que tal com vam explicar a la secció 4.2.9, ens permet estalviar haver de repetir la mateixa implementació per al *Testbed*.

A nivell d'implementació, els canvis introduïts són els següents:

- Primer, vam haver de crear una classe per a fer un *dispatching* només a un processador, independentment del seu tipus.
- Vam crear dos classes amb dos casos d'ús que manejaven la resta de les classes i decidien com havien d'interactuar amb les dades.
- Es van crear un serializador necessari per a poder exportar en format Avro els valors. Tot i que aquest canvi no ens ajuda a crear la font de dades parsejada, es va utilitzar per a crear al mateix temps que es creava el conjunt de dades parsejat, una estructura de dades que feia més ràpid el processament per al *Testbed*. Aquesta estructura de dades s'explica a la secció 4.3.6 a nivell d'implementació.

Cal destacar que el que més ens va ajudar per a afegir aquest cas d'ús és que tots els components feien només una cosa (Single Responsibility Principle, SRP), aleshores, era molt fàcil reaprofitar-los ja que se sabia quins components feien el què. També cal recalcar el fet que la injecció de dependències ens va ajudar també a definir diferents implementacions i, posteriorment, escollir en tot moment quina implementació utilitzar. Finalment, recalcar que hem sapigut utilitzar de forma intel·ligent també el *checkpoint*, creat per a fer més eficients els processadors columnars. Aquestes dades estan parsejades i tenen un format fàcilment llegible tant per a Spark com per a MapReduce, per la qual cosa ens és ideal per a servir com a entrada del *Testbed*.

El format utilitzat per al conjunt de dades Parsejat és Parquet, ja que bàsicament és una còpia del *checkpoint*, on tots els valors estan transformats a cadenes de text.

4.3.10 Invocacions

Aquesta tasca consisteix en primer traduir totes les invocacions d'operacions a codi específic del *framework* que estiguem mesurant. Per a determinar quin dels dos *frameworks* volem mesurar, hem d'especificar-ho a través d'un argument a l'hora d'executar el *Testbed*. També creem un Multimap per a tots els conjunts de dades intermedis. Cal destacar que ens referim a conjunt de dades intermedi a una estructura de dades genèrica que serveix tant per a Spark com per a MapReduce. En realitat, per a Spark serà un objecte Dataset, mentre que per a MapReduce, serà la ruta a HDFS del resultat intermedi. Abans de d'executar el bucles que fa totes les invocacions, una darrere de l'altra, començarem a monitorar les invocacions i quan s'hagi acabat d'executar el bucle, pararem de monitorar les invocacions. El monitoratge és una tasca que està descrita en més detall, a nivell d'implementació a secció 4.3.11.

Abans de fer les invocacions, el *Testbed* utilitza el BeanFactoryAnnotationUtils per a poder obtenir els adaptadors correctes per al *framework* de processament de dades que haguem especificat com a argument del *Testbed*. Aquesta factoria que pertany al *framework* Spring que utilitzem per a injectar les dependències, ens permet injectar els *Beans* correctes, en funció de l'operació que vulguem invocar. És en aquest moment, en el qual ja disposem de codi específic de Spark o MapReduce. Un cop tinguem el codi específic de les operacions, és quan comencem a fer el monitoratge.

Cada cop que invoquem una operació, recollim el conjunt de dades de sortida, i el posem en un Multimap, amb la clau igual a l'operació física a la que fa referència l'invocació i com a valor el conjunt de dades de sortida. L'ordenació topològica ens assegura que quan processem una determinada operació, totes les seves dependències s'ha processat abans i, si per a cada invocació d'operació posem per a cada operació que depèn de l'invocació d'operació que estiguem processant l'operació física com a clau en el Multimap i com a valor el resultat d'invocar l'operació, sempre tindrem per a una operació determinada les seves entrades en el Multimap. Cada cop que es processa una invocació d'operació, s'elimina l'operació física associada a l'invocació del Multimap. Al acabar de fer les invocacions, el Multimap sempre quedaria buit, ja que s'haurien processat totes les operacions. L'invocació d'operació empaqueta l'operació física a la que es fa referència, una col·lecció d'operacions físiques que depenen d'aquesta operació i també especifica si és o no l'última operació abans de l'operació de *Sink*.

En el cas de Spark, l'operació de *Load* carrega en un objecte dataset el conjunt de dades parsejat en format Parquet i aquest objecte és transformat a través de les diferents operacions del *pipeline*, tot i que no s'executa res fins que a'arriba a l'operació de *Sink*, que obliga al *framework* a fer l'execució de les operacions. És per això perquè fem una distinció entre invocació i execució, ja que és el *framework* de processament de dades l'encarregat de decidir el moment més adequat per a fer les execucions. Parquet és un format de representació de les dades per columnes; es pot trobar més informació a [56].

En el cas de MapReduce, l'operació de *Load* i l'operació de *Sink* són *dummys*, és a dir

no realitzen cap operació del tipus Map o Reduce. L'operació de *Load* només prepara la següent operació que s'executi per a que llegeixi el conjunt de dades parsejat en format Parquet posant en el conjunt dades intermedi el *path* del conjunt de dades parsejat. Cada operació escriurà el resultat del Reducer o del Mapper (si no hi ha Reducer) a HDFS i aquesta ruta es col·locarà en el conjunt de dades intermedi.

Cal destacar que tant el conjunt de dades parsejat, com els conjunts de dades intermedis no es comprimeixen amb algorismes que fan una *heavy compression*, però sí que el format Parquet aplica internament compressió *lightweight*, com podem observar a [57]. Els algorismes que fan una *heavy compression* són algorismes com Snappy, LZ77, d'entre altres, que aconsegueixen una ràtio de compressió molt alta. Això vol dir que la grandària del fitxer comprimit és molt més petita que el fitxer descomprimit. Els algorismes que fan una *lightweight compression* requereixen molts menys recursos que els que fan una *heavy compression*, encara que la grandària del fitxer comprimit és més gran que la grandària del fitxer comprimit amb un algorisme que fa una *heavy compression*. Nosaltres vam desactivar la *heavy compression* per tal d'evitar esbiaixis a l'hora de fer les comparacions entre els dos *frameworks*.

Per a poder determinar com implementar les operacions relacionals per als dos *frameworks* de processament de dades, vam fer una anàlisi per a cada una per a determinar l'implementació. Aquesta anàlisi es pot trobar en el capítol C dels apèndixs.

4.3.11 Monitoratge

El monitoratge està implementat de tal forma que hi ha un monitor que s'encarrega de mesurar una o més mètriques que estiguin molt relacionades. Això ho hem fet per a que sigui molt fàcil poder especificar a l'hora de declarar els *Beans*, quins monitors utilitzarem.

Un monitor consisteix d'una classe que rep un objecte callable i el Pla d'Invocacions i retorna informació de monitoratge. El callable està pensat per a que consisteixi del bucle que fa les invocacions o altres monitors. D'aquesta forma, podem composar els monitors de forma senzilla, com si fossin operacions matemàtiques. L'operació de composició g . f [58] aplica la funció f a x i passa el resultat a la funció g. Aquesta és la tècnica que hem seguit per a poder complir el *Single Responsibility Principle* i tenir al mateix temps la màxima visibilitat possible a l'hora de prendre mesures, ja que podíem per exemple prendre el valor de temps de CPU en el mode usuari des del moment en el que es va iniciar la màquina abans de cridar el callable i després de cridar-lo. Aleshores podíem obtenir la diferència i reportar-la.

Cal fer especial menció que hem utilitzat el sistema Prometheus [59] per a poder recollir les mètriques. Aquest sistema s'acostuma a utilitzar en sistemes en producció per a tenir més visibilitat sobre els clústers d'ordinadors. Nosaltres, en comptes d'utilitzar-lo per a monitorar un sistema, el vam utilitzar per a monitorar el funcionament dels *frameworks* de processament de dades. Prometheus és un sistema de recol·lecció de mètriques i una eina per a crear alertes desenvolupara per SoundCloud [60]. Aquest és programari Open Source i és un sistema de disseny similar al sistema de monitoratge Borgmon de Google. Les parts que composen Prometheus són les següents:

- Recol·lecció de mètriques, emmagatzematge en una base de dades i opcionalment recpuerar-les quan es consulten. La base de dades és una *Time Series Data Base*, que és una base de dades que està optimitzada per a dades de sèries de temps o que contenen una marca de temps [61]. Les dades de sèries de temps són simplement mesurament o events que són rastrejats, monitorats, reduïts de resolució (també anomenat *downsampling*) i agregats al llarg del temps. Les mètriques poden ser mètriques de servidor, monitoratge del rendiment de l'aplicació, dades d'utilització de la xarxa, entre altres. Una base de dades de sèries de temps té propietats de disseny arquitecturals clau que les fan molt diferents de les altres bases de dades. Aquestes inclouen emmagatzemament de dades que tenen marques de temps i compressió, administració del cicle de vida d'aquestes, resum de les dades, habilitat de manejar grans sèries de temps, i poder fer consultes sobre aquestes dades.

- Els exportadors són programes externs que ingereixen dades d'una varietat de fonts i les transformen en mètriques que Prometheus pot recolilr.

- El component AlertManager és un sistema d'administració d'alarmes que vé integrat. Nosaltres no utilitzarem aquest sistema.

- Es poden utilitzar llibreries de cluent per a instrumentar aplicacions.

També caldria aclarir la definició de mètrica. Una mètrica és una eina que ens permet monitorar el valor d'una magnitud al llarg del temps. Nosaltres utilitzarem l'exportador node_exporter, que es pot trobar a [62], que ens permetrà exportar les mètriques del *hardware* i del Sistema Operatiu exposat pels kernels *NIX. Una gran característica de Prometheus per a nosaltres és que podem obtenir els valors d'aquestes mètriques de forma senzilla per mitjà de consultes en el llenguatge PromQL. PromQL (Prometheus *Query Language*) és el llenguatge que utilitza Prometheus per a permetre als usuaris seleccionar i agregar les dades de sèries de temps en temps real [63].

Cal recalcar el fet que no hem trobar cap client en Java que pugues fer consultes a Prometheus, vam haver de crear nosaltres el client que accedia a través de l'API a la funcionalitat de fer consultes.

Els següents monitors els hem implementat fent consultes a través del client que vam implementar a Prometheus:

- LocalFileSystemWrittenBytesMonitor

- LocalFileSystemReadBytesMonitor

- CPUSystemTimeMonitor

- CPUSystemTimeMonitor

- CPUTotalTimeMonitor

- CPUUserTimeMonitor

- MinMemoryUtilizationMonitor

- MaxMemoryUtilizationMonitor
- AverageMemoryUtilizationMonitor
- MinSwapUtilizationMonitor
- MaxSwapUtilizationMonitor
- AverageSwapUtilizationMonitor
- NetworkReceivedBytesMonitor
- NetworkTransmittedBytesMonitor

En canvi, els monitors següents, els hem implementat sense fer cap consulta a Prometheus:

- ChronometerMonitor
- ExecutionInstantsMonitor
- DistributedFileSystemMonitor

Cal remarcar primer, que el monitor ChronometerMonitor és el que està més proper al bucle de les invocacions, per a que sigui més precisa la presa del temps. Segon, el monitor DistributedFileSystemMonitor fa un monitoratge exclusivament de l'ús que es fa dels resultats intermedis, és a dir, excloent el conjunt de dades parsejat i el conjunt de dades de sortida. Per últim, el monitor ExecutionInstantsMonitor registra els instants de temps en el que ha començar i acabat la invocació de les operacions i també es troba molt pròxim al bucle que fa les invocacions. Aquests dos instants ens permeten fer consultes a l'interfície de Prometheus i recollir les mètriques que no vam tenir en compte a l'hora de fer els experiments. Cal recordar que Prometheus emmagatzema les mètriques que exporta el node_exporter en una base de dades de temps real. Aleshores, podem fer la consulta i obtenir els punts amb els valors de les mètriques que estiguem buscant. Això ens evita tornar a llançar un experiment pel simple fet que ens falti una mètrica, estalviant recursos.

Les mètriques es recullen cada 100 mil·lisegons i no tenen un gran efecte que es pugui notar en el rendiment dels nodes.

La versió de Prometheus utilitzada és la 2.26.0.linux-amd64 i la versió del node_exporter és la 1.1.2.linux-amd64.

Una altra alternativa que vam tenir a l'hora de decidir com implementar el monitoratge de les invocacions era crear un sistema que s'executés als nodes executors i que recollissin dades utilitzant una llibreria de Java que, per sota, consultava l'informació retornada per iostat [64]. Aleshores, aquest subsistema encarregat del monitoratge, pujava en un format determinat, com per exemple Json, les mètriques periòdicament a HDFS, d'on el node *master* recollia les dades periòdicament i les processava. Cal destacar que aquesta solució no era la millor perquè primer, estàvem construint un sistema que no estava provat, per la qual cosa, necessitava molt temps de desenvolupament per a solucionar els errors que podien sorgir. Addicionalment, aquest sistema era menys flexible, ja que no tenia una base de dades de

temps real per sota, que ens permetés fer consultes. També cal considerar que el temps d'implementació no era nul, per la qual cosa, era més senzill instal·lar Prometheus al clúster que implementar un nou sistema per a monitoratge i estar reinventant la roda. Prendre la decisió d'utilitzar Prometheus ens va estalviar aproximadament una setmana de treball d'implementació com a mínim, i ens va acabar aportant més valor que crear un sistema de monitoratge.

4.3.12 Invocacions Instrumentades

La tasca encarregada de fer les invocacions instrumentades, comparteix el mateix codi que la tasca encarregada de fer les invocacions. L'única excepció és a l'hora d'injectar els *Beans*. Spark injecta uns *Beans* que estan envoltats d'una operació que mesura la quantitat de files i el nom de les columnes d'entrada de l'operació i la quantitat de files i el nom de les columnes de la sortida de l'operació. Aquesta informació posteriorment, es visualitzarà en un *sheet*.

Es pot veure clarament que l'operació que fa el recompte de files té un pes determinat i és aquest el motiu pel qual vam crear dos casos d'ús; és per a separar aquest comportament, per a que no afecti a la mesura del temps.

Es pot especifica a través d'un argument del programa l'hora d'executar el *Testbed* si volem que l'execució sigui instrumentada o no.

4.3.13 Visualització

Aquesta tasca consisteix en recollir tota la informació del monitoratge i de les invocacions instrumentades, formatar-la i presentar-la en un arxiu que sigui fàcilment interpretable per als experiments.

Per a implementar aquesta tasca, teníem dues alternatives diferents. La primera era utilitzar el format CSV. Aquest és un format senzill de manipular i no requereix llibreries massa complexes. La desavantatge és que no ens permet tenir fórmules, ni *sheets* separats. Totes les dades es guarden en un format de text i aquestes s'han de parsejar per programes com per exemple Libre Office o Excel. La segona alternativa que vam tenir és utilitzar un format més avançat, com és el XLSX, que permet crear fórmules, posar colors a les taules i tenir diferents *sheets*. Les desavantatge que té aquest format són primer, la complexitat i la necessitat de planificar com es posen les dades i en quines cel·les. Aquest esforç estava també en el format CSV, però no requeria un disseny massa complex de la classe encarregada de la visualització. Segon, hi ha restriccions implícites a l'hora de crear els arxius XLSX. Per exemple, hi ha un límit de 32 caràcters per al nom dels *sheets* [65], o hi ha un límit per al nombre d'estils de les cel·les.

Durant les primeres iteracions del *Testbed*, vam implementar l'adaptador que permetia crear una sortida amb CSV. Encara que a mesura que el programa continuava creixent, vam decidir crear un adaptador per a XLSX, que ens permetés crear una sortida en aquest format. Aquesta implementació ens podia estalviar molt temps a l'hora de crear els Excels, de forma que ens ha semblat que el temps de desenvolupament compensava haver de tractar tots els

arxius manualment posteriorment. També, els arxius de sortida dels experiments quedaven amb un format molt professional sense requerir un esforç massa gran, per tant, aquesta va ser l'alternativa definitiva que vam escollir.

Addicionalment, cal recalcar que utilitzar l'alternativa que ens permetia crear una sortida en XLSX, ens permetia tractar l'arxiu de sortida com si fos una base de dades. És a dir, primer executàvem un experiment, es creava l'arxiu i posàvem la sortida. En la segona execució, el *Testbed* detectava que ja s'havia llegit un primer cop i escrivia al costat i així successivament.

Per a fer una sobreescriptura d'aquest comportament, vam afegir un *flag* en els arguments del programa que permetia fer-ho. Si el flag estava activat, aleshores independentment del contingut de l'arxiu, aquest s'esborrava i s'escrivia de nou començant per la primera columna, que era la primera execució.

4.3.14 Resultat

Aquest és un arxiu que recull tota la informació de sortida de la tasca de Visualització. S'utilitza el format XLSX pel raonament explicat a la secció 4.3.13.

4.3.15 Altres detalls

En aquesta secció veurem altres detalls també importants de la implementació del *Testbed*.

Llenguatge de programació

El llenguatge de programació a escollir estava entre dues alternatives: utilitzar Python o Java. Si utilitzéssim Python, aquest en acceleraria el desenvolupament per a la implementació del codi Spark, ja que Spark proporciona la llibreria PySpark [66], que permet la utilització del *framework* des d'aquest llenguatge de programació. També hi ha suport per a MapReduce, que es detalla en més profunditat a [67]. Python és un llenguatge de programació interpretat, en alt nivell i de propòsit general que està orientat a objectes [68]. Aquest llenguatge de programació està enfocat en la llegibilitat del codi. Aquest llenguatge de programació és ideal per a:

- Programadors nous, és a dir, sense experiència anterior en desenvolupament de codi.
- Implementar prototipus de forma ràpida, és a dir, sense tenir un cost de desenvolupament alt.
- Tenir una gran llegibilitat del codi que permet que els lectors l'entenguin d'una forma ràpida.

De les alternatives anteriors, les que més importància per al nostre projecte tenen és la capacitat d'implementar prototipus sense tenir un cost de desenvolupament alt i obtenir una bona llegibilitat del codi. Cal destacar, però que al estar interpretat, el rendiment que s'obtindrà no serà el mateix que quan una màquina executi codi natiu.

En canvi, Java també és un llenguatge de programació de propòsit general i és orientat a objectes. Aquest llenguatge de programació, és que està dissenyat per ser executat en qualsevol lloc i, per tant, utilitza la JVM (Java Virtual Machine). Aquest component actua com l'interpretador del codi. Java es va utilitzar durant molt temps i va crear una gran comunitat, que van crear moltes llibreries i donant-lo suport. La programació en aquest llenguatge de programació és relativament senzilla ja que té moltes llibreries que permeten escriure codi per a un propòsit específic. Cal destacar que la sintaxis d'aquest llenguatge de programació és més verbosa, en comparació amb Python. Una diferència important respecte Python és que utilitza un compilador JIT (Just in Time) [69], que permet millorar el rendiment de les aplicacions Java, compilant els codis en codi màquina natiu en temps d'execució, permetent obtenir rendiments molt més alts que en Python. A més, cal fer èmfasi en el fet que la JVM permet més flexibilitat que l'interpretador de Python, per exemple restringint la quantitat de memòria que s'estigui utilitzant.

L'alternativa per la que ens vam decantar és per utilitzar Java, principalment perquè els dos *frameworks* de processament de dades utilitzaven aquest mateix llenguatge de programació en la seva implementació. Tot i que les possibilitat de Python que ofereix per a crear un codi més llegible són importants, el problema gran que podria comportar utilitzar aquest llenguatge de programació és que MapReduce havia d'executar els scripts de Python com a Mappers o Reducers, mentre que Spark manejava les invocacions internament en Java. El rendiment de MapReduce depèn bàsicament del rendiment dels Mappers i Reducer que estiguem utilitzant i, per tal de no penalitzar a un *framework* més que a un altre, vam decidir prendre aquesta decisió.

Execució Local

Cal destacar el fet que podem especificar a través d'un flag a l'hora d'executar el programa si volem que l'execució es faci de forma local o en el clúster. Si volem que es faci de forma local, el *Testbed* utilitzarà les llibreries de Spark i MapReduce en mode local. Si volem que s'executi al clúster, indicarà a les llibreries que utilitzin Yarn, que és un planificador dels treballs que es troben en execució al clúster [70]. Aquesta característica ens ha permès estalviar recursos al poder executar el *Testbed* al portàtil en el que es feia el desenvolupament, sense necessitat de tenir el clúster encès. Tot i així, a l'hora de fer la migració al clúster, ens hem trobat amb una sèrie de problemes, que es detallen en el capítol B dels apèndixs.

Java Streams

Cal fer especial menció que hem utilitzat aquesta característica de Java en profunditat. Els Java Streams [71] són una característica de Java que ens permet executar una seqüencia d'operacions sobre els elements d'una col·lecció de dades i agregar els seus resultats. Es necessiten 3 parts: un generador, operacions intermèdies i una operació terminal. Les operacions intermèdies retornen un nou *stream* modificat. Els *streams* tenen una invocació *lazy*, és a dir que les operacions només s'invocaran si és necessari per a l'execució de l'operació terminal. Per exemple, una operació terminal seria findFirst(), que simplement processaria el *stream* fins que trobaria un únic element.

Cal destacar que l'ordenació de les operacions juga un paper fonamental a l'hora d'exe-

cutar els Java Streams, ja que sempre tindrà un major cost computacional processar totes les dades i després filtrar-les que fer-ho al revés, si és possible.

Els Java Streams ens permeten expressar d'una forma llegible un conjunt de transformacions que s'efectuen sobre col·leccions de dades i també fa que el codi sigui molt canviable. Per tant, ens va semblar una característica molt interessant.

Functional Java

A l'hora de fer el desenvolupament del *Testbed*, vam intentar seguir la programació funcional [72], amb les característiques que ens ofereix Java per això. La programació funcional és un estil d'escriptura de programes d'ordinador que tracten les computacions com si s'avaluessin funcions matemàtiques. Una funció matemàtica és una expressió que relaciona un conjunt d'entrada a un conjunt de sortida. La sortida d0una funció només depèn de la seva entrada i podem composar dues o més funcions per a crear una nova funció.

Una característica important utilitzada va ser el càlcul de lambdas, molt utilitzat juntament amb els Java Streams, especificats a la secció 4.3.15.

Una altra característica va ser la immutabilitat dels objectes. La immutabilitat és un dels principis fonamentals de la programació funcional i es refereix a la propietat d'una entitat de no ser modificada després de ser instanciada. No hem pogut mantenir aquest concepte per a tota la implementació, encara que hem fet un esforç per fer que tots els paràmetres de les funcions siguin immutables i reduir de forma significativa, per mitjà dels streams, les variables que canviaven el seu contingut un cop inicialitzades.

La composició de funcions la vam utilitzar per als monitors, ja que ens permetia més flexibilitat a l'hora de fer els mesuraments.

La raó principal d'utilitzar aquest estil, per mitjà de les característiques de Java, és perquè permeten tenir un codi sense efectes laterals, fàcils de llegir, raonar, testejar i mantenir.

Logs

També caldria fer menció de la funcionalitat de produir *logs* del *Testbed*. Això ens va permetre detectar els *bugs* i problemes a l'hora d'implementar el *Testbed*. A mesura que l'execució avançava per totes les tasques de l'arquitectura especificada a la secció 4.2, es produïa una línia de *log* que imprimia el contingut de tots els plans.

Injecció de dependències

Com hem vist al llarg de l'implementació del *Testbed*, ha sigut molt important l'injecció de dependències, ja que per mitjà d'un argument que es passa a l'hora d'executar el *Testbed*, podem especificar quin *framework* de processament de dades volem i si volem que funcioni amb el cas d'ús instrumentat o de mesurament del temps. D'aquesta forma, hem pogut reutilitzar el màxim de codi possible, injectant només els *Beans* amb la implementació que tocava.

Els *Beans* [73] són els objectes que formen la columna vertebral de la nostra aplicació i

són administrades pel Spring IoC Container. IoC o *Inversion of Control* és un procés e el que un objecte defineix les seves dependències sense crear-les. L'objecte delega el treball de construcicó d'aquestes dependències a un IoC Container. Aquesta va ser l'única característica de les que oferia Spring que vam utilitzar.

Hi havia altres alternatives a Spring com per exemple el Guice o Dagger, tal com podem veure a [74], que també permeten utilitzar un IoC Container. De les tres alternatives, vam utilitzar Spring perquè és un *framework* ric en característiques i no sabíem al començament si necessitaríem més característiques o no, agafant-lo ens estalviaria temps de desenvolupament en aquest sentit perquè no hauríem de canviar a un altre *framework* posteriorment. Les altres alternatives són més lleugeres a nivell de *framework*, encara que no tenen tantes integracions com Spring.

Execució en script

Per a executar el *Testbed*, que es trobava en un JAR, vam crear un script en bash que permetia fer tres invocacions successives mesurant els temps i finalment una execució del *Testbed* per a fer la instrumentació de les invocacions. Quan aquest experiment acabava, es pujava el resultat en format XLSX al Google Drive.

Vam utilitzar també scripts de Python que ens permetien dissenyat molts *pipelines* a partir d'un *template* i invocaven al script que executava els experiments per a cada pipeline.

4.4 Demostració de la correctesa

Per a demostrar la correctesa del *Testbed*, mostrarem un *pipeline* que utilitzarà el nostre *Testbed* i un fragment de codi en Python que faci les mateixes operacions que el *pipeline*. Modificarem l'operació de *Sink* del *Testbed* per a que després d'escriure el resultat, ens expliqui el Pla Físic de Spark. També farem que Spark expliqui el Pla Físic seu en el script. Aleshores, comprovarem que no hi hagi diferències i d'aquesta manera demostrarem que el *pipeline* és equivalent al que obtindríem amb el script.

Si els plans físics de Spark obtinguts del *Testbed* i del fragment de codi en Pyhon són iguals, també serà correcte el codi en MapReduce, ja que com hem vist a l'arquitectura, la gran diferència entre executar un *framework* o un altre radica en el fet de quins *Beans* estem injectant.

El *pipeline* que utilitzarem com a entrada del *Testbed* es mostra en el Fragment 4.2. El script en Python que utilitzarem per a demostrar la correctesa és el que es mostra al Fragment 4.3.

En el Fragment 4.4 podem observar el Pla Físic de Spark que hem obtingut del script en Python i en el Fragment 4.5 podem observar el Pla Físic de Spark que hem obtingut del *Testbed*. No confondre el Pla Físic de Spark amb el Pla Físic del *Testbed*, ja que el Pla Físic de Spark és una descripció dels algorismes que utilitzarà aquest internament per a executar les invocacions que fem i és específic del *framework*. El Pla Físic del *Testbed* és un resultat obtingut que és agnòstic del *framework* que estiguem utilitzant.

```
[
      {
         "operation": "LOAD",
         "outputTag": "dataset1",
         "datasetDirectoryPath": "input/Ad_click_on_taobao_10000"
      },
      {
         "operation": "LOAD",
         "outputTag": "dataset2",
         "datasetDirectoryPath": "input/Ad_click_on_taobao_V2_10000"
      },
      {
         "operation": "PROJECT",
         "inputTag": "dataset1",
         "outputTag": "projectedDataset",
         "columnSelectivityFactor": 0.70
      },
      {
         "operation": "SELECT",
         "inputTag": "dataset2",
         "outputTag": "selectedDataset",
         "rowsSelectivityFactor": 0.25,
         "columnName": "User"
      },
      {
         "operation": "JOIN",
         "leftInputTag": "projectedDataset",
         "rightInputTag": "selectedDataset",
         "outputTag": "joinedDataset",
         "joinLeftColumnName": "DateTime",
         "joinRightColumnName": "DateTime"
      }
]
```

Fragment 4.2: Pipeline utilitzat per a demostrar la correctesa
[Font: elaboració pròpia]

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

MASTER = "master"
LOCAL = "local[*]"
LEFT_PREFIX = "Left"
RIGHT_PREFIX = "Right"

LEFT_DATASET_PATH = "input/Ad_click_on_taobao_10000/parsed_dataset"
RIGHT_DATASET_PATH = "input/Ad_click_on_taobao_V2_10000/parsed_dataset"
PROJECTED_COLUMNS = ["AdGroupId", "Clk", "DateTime", "NonClk"]
SELECT_QUERY = "User <= '233060'"
LEFT_JOIN_ATTRIBUTE = "LeftDateTime"
RIGHT_JOIN_ATTRIBUTE = "RightDateTime"
OUTPUT_DATASET_PATH = "output"

def add_prefix(dataframe, prefix):
   renamed_dataframe = dataframe
    for column_name in dataframe.columns:
   renamed_dataframe = renamed_dataframe.withColumnRenamed(column_name, prefix +
       column_name)
   return renamed_dataframe

if __name__ == "__main__":
    spark = SparkSession \
              .builder \
              .config(MASTER, LOCAL) \
              .getOrCreate()

left_dataset = spark.read.parquet(LEFT_DATASET_PATH)
right_dataset = spark.read.parquet(RIGHT_DATASET_PATH)
projectedDataset = left_dataset.select(*PROJECTED_COLUMNS)
projectedDatasetWithPrefix = add_prefix(projectedDataset, LEFT_PREFIX)
selectedDataset = right_dataset.filter(SELECT_QUERY)
selectedDatasetWithPrefix = add_prefix(selectedDataset, RIGHT_PREFIX)
joinedDataset = projectedDatasetWithPrefix.join(selectedDatasetWithPrefix,
    col(LEFT_JOIN_ATTRIBUTE) == col(RIGHT_JOIN_ATTRIBUTE))
joinedDataset.write.parquet(OUTPUT_DATASET_PATH)
joinedDataset.explain()
```

Fragment 4.3: Script en Python utilitzat per a demostrar la correctesa
[Font: elaboració pròpia]

```
== Physical Plan ==
*(2) BroadcastHashJoin [LeftDateTime#38], [RightDateTime#55], Inner, BuildLeft
:- BroadcastExchange HashedRelationBroadcastMode(List(input[2, string, true])), [id=#83]
:  +- *(1) Project [AdGroupId#2 AS LeftAdGroupId#28, Clk#5 AS LeftClk#33, DateTime#1 AS
    LeftDateTime#38, NonClk#4 AS LeftNonClk#43]
:     +- *(1) Filter isnotnull(DateTime#1)
:        +- *(1) ColumnarToRow
:           +- FileScan parquet [DateTime#1,AdGroupId#2,NonClk#4,Clk#5] Batched: true,
    DataFilters: [isnotnull(DateTime#1)], Format: Parquet, Location:
    InMemoryFileIndex[file:/run/media/user/4C0679F60679E0FE/Data/Universitat/Assignatures/
TFG/Executa..., PartitionFilters: [], PushedFilters: [IsNotNull(DateTime)], ReadSchema:
    struct<DateTime:string,AdGroupId:string,NonClk:string,Clk:string>
+- *(2) Project [User#12 AS RightUser#48, DateTime#13 AS RightDateTime#55, AdGroupId#14 AS
    RightAdGroupId#62, PID#15 AS RightPID#69, NonClk#16 AS RightNonClk#76, Clk#17 AS
    RightClk#83]
+- *(2) Filter ((isnotnull(User#12) AND (User#12 <= 233060)) AND isnotnull(DateTime#13))
+- *(2) ColumnarToRow
+- FileScan parquet [User#12,DateTime#13,AdGroupId#14,PID#15,NonClk#16,Clk#17] Batched:
    true, DataFilters: [isnotnull(User#12), (User#12 <= 233060), isnotnull(DateTime#13)],
    Format: Parquet, Location:
    InMemoryFileIndex[file:/run/media/user/4C0679F60679E0FE/Data/Universitat/Assignatures/
TFG/Executa..., PartitionFilters: [], PushedFilters: [IsNotNull(User),
    LessThanOrEqual(User,233060), IsNotNull(DateTime)], ReadSchema:
    struct<User:string,DateTime:string,AdGroupId:string,PID:string,NonClk:string,Clk:string>
```

Fragment 4.4: Pla Físic de Spark del script
[Font: elaboració pròpia]

```
== Physical Plan ==
*(2) BroadcastHashJoin [LeftDateTime#38], [RightDateTime#55], Inner, BuildLeft, false
:- BroadcastExchange HashedRelationBroadcastMode(List(input[2, string, true]),false),
    [id=#34]
:  +- *(1) Project [AdGroupId#2 AS LeftAdGroupId#28, Clk#5 AS LeftClk#33, DateTime#1 AS
    LeftDateTime#38, NonClk#4 AS LeftNonClk#43]
:     +- *(1) Filter isnotnull(DateTime#1)
:        +- *(1) ColumnarToRow
:           +- FileScan parquet [DateTime#1,AdGroupId#2,NonClk#4,Clk#5] Batched: true,
    DataFilters: [isnotnull(DateTime#1)], Format: Parquet, Location:
    InMemoryFileIndex[file:/run/media/user/4C0679F60679E0FE/Data/Universitat/Assignatures/
TFG/Executa..., PartitionFilters: [], PushedFilters: [IsNotNull(DateTime)], ReadSchema:
    struct<DateTime:string,AdGroupId:string,NonClk:string,Clk:string>
+- *(2) Project [User#16 AS RightUser#48, DateTime#17 AS RightDateTime#55, AdGroupId#18 AS
    RightAdGroupId#62, PID#19 AS RightPID#69, NonClk#20 AS RightNonClk#76, Clk#21 AS
    RightClk#83]
+- *(2) Filter ((isnotnull(User#16) AND (User#16 <= 233060)) AND isnotnull(DateTime#17))
+- *(2) ColumnarToRow
+- FileScan parquet [User#16,DateTime#17,AdGroupId#18,PID#19,NonClk#20,Clk#21] Batched:
    true, DataFilters: [isnotnull(User#16), (User#16 <= 233060), isnotnull(DateTime#17)],
    Format: Parquet, Location:
    InMemoryFileIndex[file:/run/media/user/4C0679F60679E0FE/Data/Universitat/Assignatures/
TFG/Executa..., PartitionFilters: [], PushedFilters: [IsNotNull(User),
    LessThanOrEqual(User,233060), IsNotNull(DateTime)], ReadSchema:
    struct<User:string,DateTime:string,AdGroupId:string,PID:string,NonClk:string,Clk:string>
```

Fragment 4.5: Pla Físic de Spark del *Testbed*
[Font: elaboració pròpia]

Podem observar tant en el Fragment 4.4 com en el Fragment 4.5 que els dos plans físics de Spark coincideixen. Per tant, hem demostrat la correctesa del *Testbed*.

4.5 Conclusió

Finalment, voldríem mencionar que hem construït el sistema explicat en aquest capítol per a poder executar els experiments que els expliquem en el capítol 5.

Capítol 5

Experiments

Un experiment és una procés pel qual es manipula de manera intencionada una o més variables independents, definides com a causes, per a una posterior anàlisi de les conseqüències que tenen sobre altres variables identificades com a efectes [75].

Abans d'executar cap experiment, ens hauríem de plantejar la necessitat real de fer-ho. Hi ha un article a [76] de Jeffrey Ullman en el que es menciona que no s'haurien d'acceptar els experiments com un substitut per a una anàlisi més exhaustiu i general, a menys que no hi hagi cap altra manera per a parametritzar l'espai d'entrada adequadament. També especifica que hauríem de parar de pensar en els experiments com un substitut per a una anàlisi i un enteniment més profund de perquè alguns nostres algorismes funcionen millor que altres.

En el nostre cas, necessitem veure quin dels dos *frameworks* de processament de dades funciona millor sota alguns escenaris. Aquests escenaris tenen una configuració específica i volem observar mètriques de rendiment com per exemple el temps d'execució, ús de memòria RAM o ús del disc. Aquestes mètriques no és possible obtenir-les de forma exacta per mitjà d'una anàlisi exhaustiva, sinó que només es podria obtenir una aproximació, ja que depenen en gran mesura de l'implementació dels algorismes. L'experimentació ens podria donar valors exactes per a aquestes mètriques, que posteriorment, podríem utilitzar per a justificar quin dels sistemes funcionaria millor sota unes determinades circumstàncies. Un altre argument a favor de l'experimentació és el fet que no acostumem a trobar els dos *frameworks* aïllats d'altres sistemes. L'entorn on s'acostumen a utilitzar més aquest tipus d'eines és en l'entorn de producció, on cal tenir en compte limitacions imposades pel sistema operatiu, limitacions de la xarxa i molts més detalls que són importants però no podem observar amb una anàlisi exhaustiva.

Addicionalment, cal recalcar que quan s'empren solucions al núvol dels dos *frameworks*, aquestes tenen un cost econòmic i entendre en detall el funcionament dels dos sistemes és essencial per a poder preveure aquest cost.

Per tant, es fa imprescindible l'ús d'experiments per a la comparativa i, en la resta del capítol validarem l'arquitectura creada per a executar els experiments. Començarem definint una configuració general i explicarem l'experimentació feta.

5.1 Configuració general

En aquesta secció detallarem els detalls hardware del clúster que hem utilitzat i que seran els mateixos per a tots els experiments.

Cluster

El clúster que hem utilitzat és el proveït pel grup d'investigació al RDLab [77] i consta de 4 nodes amb iguals característiques de hardware. Aquests nodes són màquines virtuals, és a dir que els servidors sobre els que s'executen aquestes màquines virtuals són compartits entre altres investigadors. Una màquina virtual disposa de 8 cores amb 32 GiB de memòria RAM i 1 TB de disc. Tots els nodes del clúster executen el sistema operatiu Ubuntu 20.04.2 LTS.

El clúster està distribuït entre un node que fa de *master*, anomenat dtim i tres node executors o *workers*, anomenats dtim1, dtim2 i dtim3.

La versió de Spark que utilitzarem és 3.0.2 i la versió de Hadoop a utilitzar és la 3.2.2.

La configuració de Spark es detalla en la Taula 12. La configuració de Yarn es detalla en la Taula 13. En la Taula 13, quantitat mínima de memòria RAM alocatable en un node fa referència a la grandària d'un contenidor. La configuració de MapReduce es detalla en la Taula 14. Aquestes seran les configuracions utilitzades al clúster a excepció de l'experiment en la secció 5.2.6.

Descripció	**Valor absolut**	**Valor relatiu al *dataset* petit**	**Valor relatiu al *dataset* mitjà**	**Valor relatiu al *dataset* gran**
Memòria del *driver*	11301 MiB	6.243,65 %	1.570,46 %	56,31 %
Memòria utilitzada per al *overhead* del *driver*	851 MiB	470,17 %	118,26 %	4,24 %
Memòria del *executor*	11301 MiB	6.243,65 %	1.570,46 %	56,31 %
Memòria utilitzada per al *overhead* del *executor*	851 MiB	470,17 %	118,26 %	4,24 %
Nombre de nuclis del *driver*	8	-	-	-
Nombre de nuclis de l'*executor*	8	-	-	-
Nombre d'instàncies de l'*executor*	5	-	-	-

Taula 12: Configuració de Spark utilitzada en els experiments
[Font: elaboració pròpia]

Descripció	**Valor absolut**	**Valor relatiu al *dataset* petit**	**Valor relatiu al *dataset* mitjà**	**Valor relatiu al *dataset* gran**
Memòria del *node manager*	24304 MiB	13.427,62 %	3.377,43 %	121,09 %
Quantitat mínima alocatable de processadors virtuals	1	-	-	-
Quantitat màxima alocatable de processadors virtuals	8	-	-	-
Quantitat màxima de memòria RAM alocatable en un node	24304 MiB	13.427,62 %	3.377,43 %	121,09 %
Quantitat mínima de memòria RAM alocatable en un node (grandària d'un contenidor)	3038 MiB	1.678,45 %	422,18 %	15,14 %
Memòria utilitzada pel contenidor de l'*application master* [78]	6076 MiB	3.356,91 %	844,36 %	30,27 %
Memòria utilitzada per la màquina virtual de Java de l'*application master*	4861 MiB	2.685,64 %	675,51 %	24,22 %

Taula 13: Configuració de Yarn utilitzada en els experiments
[Font: elaboració pròpia]

Descripció	**Valor absolut**	**Valor relatiu al *dataset* petit**	**Valor relatiu al *dataset* mitjà**	**Valor relatiu al *dataset* gran**
Memòria del Map	3038 MiB	1.678,45 %	422,18 %	15,14 %
Memòria del Reduce	6076 MiB	3.356,91 %	844,36 %	30,27 %
Memòria de la màquina virtual de Java del Map	2431 MiB	1.343,09 %	337,83 %	12,11 %
Memòria de la màquina virtual de Java del Reduce	4861 MiB	2.685,64 %	675,51 %	24,22 %

Taula 14: Configuració de MapReduce utilitzada en els experiments
[Font: elaboració pròpia]

Finalment, en l'arxiu de configuració de Hadoop, la grandària del *heap* és 24.304 MiB, que representa el 13.427,62 %, 3.377,43 % i 121,09 % dels *datasets* petit, mitjà i gran, respectivament.

Mètriques

Per als experiments, utilitzarem el *Testbed*, que ajudarà també a recollir les mètriques de les invocacions, com es detalla més en profunditat en les seccions 4.2.11 i 4.3.11.

Durant els experiments, analitzarem les següents mètriques de forma general, si hi ha alguna especificitat, s'especificarà a l'experiment que pertoqui:

- **Temps d'invocació.** És el temps d'invocació de les operacions definides al *pipeline*. Cronometrar aquest procés ens pot donar una bona idea de quin dels dos sistemes seria més preferible a l'hora d'escollir un sistema que funcioni de forma eficient en quant a temps.
- **Utilització mitjana de memòria RAM dels nodes executors.** És l'utilització mitjana de la memòria RAM durant l'experiment. Aquesta mètrica mesurarà quant d'eficient és cada *framework* amb els recursos disponibles i es basa en quants bytes s'estan utilitzant en tot el sistema durant el temps que durin les invocacions.
- **Utilització del disc local a cada node executor per a lectura.** Indica l'utilització del disc local a cada node executor per a lectura per a cada node executor.
- **Utilització del disc local a cada node executor per a escriptura.** Aquesta mètrica ens indica el grau d'utilització del disc local a cada node executor per a l'escriptura.
- **Utilització de la memòria *swap*.** Aquesta mètrica ens serà útil per a veure quanta memòria d'intercanvi s'està utilitzant. És especialment important perquè si s'utilitza més memòria RAM que la que el sistema té disponible, el Sistema Operatiu començarà a utilitzar la memòria d'intercanvi per tal de satisfer la necessitat de memòria de les aplicacions i això penalitzarà al rendiment. Per a que el rendiment no es vegi penalitzat, els *frameworks* hauran de ser conscients de quanta memòria RAM hi ha disponible i planificar correctament el seu ús.
- **Recepció d'informació a través de la xarxa.** Recepció d'informació a través de la xarxa. Aquesta dada ens serà útil ja que en els sistemes distribuïts, acostuma a tenir un fort impacte la quantitat d'informació intercanviada a través d'aquest medi en el rendiment general del *framework*.
- **Transmissió d'informació a través de la xarxa.** Transmissió d'informació a través de la xarxa. Al igual que la mètrica "Recepció d'informació a través de la xarxa", aquesta mètrica és important per a veure com s'està utilitzant la xarxa ja que aquesta acostuma a tenir un fort impacte en els sistemes distribuïts.
- **Temps Total de CPU.** Aquesta mètrica ens indica la suma total del temps dedicat per part de tots els processadors lògics de tots els nodes executors a tots els modes possibles de CPU. Aquesta mètrica ens serà útil per a veure quin dels dos *frameworks* és més intensiu en quant a ús de la CPU.
- **Temps de CPU en el *User Mode*.** Aquesta mètrica ens informarà de l'ús que es fa de tots els processadors lògics de tots els nodes executors per part del codi d'aplicacions

que s'executen en el clúster. Aquest codi poden ser tant el codi dels Mappers com el codi que s'està executant per Spark.

- **Temps de CPU en el *System Mode*.** Aquesta mètrica indica quin ús es fa de tos els processadors lògics de tots els nodes executors per part del Sistema Operatiu. Aquest temps és dedicat a l'execució del kernel i no és aprofitat per a executar codi d'usuari.
- **Temps de CPU en el *IO Wait Mode*.** Aquesta mètrica indica quin ús es fa de tots els processadors lògics de tots els nodes executors per a esperar per a que es completin les operacions d'entrada i sortida de disc o xarxa.
- **Nombre de fallades de cada *framework*.** Aquest valor ens permetrà veure la robustesa de cada *framework*.

Pipelines

Executarem els *pipelines* per al conjunt de dades petit, mitjà i gran per a veure el comportament dels dos sistemes amb diferents grandàries del conjunt de dades a processar. Els *pipelines* que utilitzarem són els següents:

- Selecció
- Projecció
- Selecció i Projecció combinades
- *Join*
- *Join* gran

Conjunts de dades

Per als experiments, hem utilitzat els conjunts de dades seleccionats, tal com es va explicar en el capítol A dels apèndixs, com a recordatori els mencionem a continuació:

- Conjunt de dades petit: Obama Visitor Logs
- Conjunt de dades mitjà: Ad Display/Click Data on Taobao.com
- Conjunt de dades gran: Thunderbird

Ordre d'execució

Els experiments s'executaran entrellaçats. Això vol dir que primer executarem l'experiment amb MapReduce, després amb Spark, després un altre cop amb MapReduce i així successivament. Fem això per a evitar que els resultats de les gràfiques ens donin resultats falsos, on impliquin que un *framework* pot ser millor que un altre, encara que realment no depengui del *framework*.

Cal recalcar el fet que s'esborren els continguts de la carpeta que conté els resultats intermits a HDFS, s'esborren també els resultats temporals del disc local de cada node executor i es netegen les caches tal com s'especifica a [79] després de cada execució per a

garantir que no hi ha interferències entre experiments. La configuració s'executa tres cops per a evitar interferències externes i agafem la mediana dels valors de les mètriques de les tres execucions.

En els experiments, quan ens referim a percentatges per a comparar, si volem comparar el sistema A amb el sistema B i volem obtenir el percentatge, aleshores, per exemple si utilitzem l'indicador de temps d'invocació, si el temps d'invocació del sistema A dividit pel temps d'invocació del sistema B ens donarà una relació. Si restem 1 a aquesta relació i multipliquem tot per 100, obtindrem el percentatge de millora d'un sistema respecte a l'altre.

5.2 Experiments

En aquesta secció detallarem tots els experiments realitzats, seguint una estructura similar: proposarem primer l'objectiu de l'experiment, on detallarem què és el que volem experimentar, després explicarem la configuració de l'experiment, passarem a exposar els resultats obtinguts i, finalment farem una discussió sobre els resultats.

5.2.1 Operació de Selecció

Objectiu

L'objectiu d'aquest experiment és mesurar les mètriques especificades en la secció 5.1 per diferents execucions de la selecció amb diferents factors de selectivitat i sobre els tres conjunts de dades discutits anteriorment. Aquesta operació consisteix en filtrar d'un conjunt de files d'entrada i obtenir un conjunt de sortida que consisteix en les files que han complert amb un determinat predicat, mentre que aquelles files del conjunt d'entrada que no han complert amb el predicat, són descartades.

Aquesta operació és considerada una *narrow dependency*, és a dir que no és necessari intercanviar dades a través de la xarxa i, a més, es pot executar de forma distribuïda sense necessitat d'interaccionar amb els altres nodes. El rendiment d'aquesta operació hauria de ser bo per a Spark ja que no necessitaria intercanviar dades a través de la xarxa i es podria executar de forma distribuïda sense necessitat d'interaccionar amb els altres nodes. Respecte a MapReduce, aquesta operació està implementada amb només un Mapper, per la qual cosa tampoc seria necessari un intercanvi de dades per a transportar les dades als Reducers, ja que aquests estan anul·lats. En ambdós casos es tractaria de l'execució d'un únic *job* i, per tant, esperem un bon rendiment en tots dos sistemes.

Configuració

El *pipeline* que executarem per a aquest experiment, variant el factor de selectivitat és el *pipeline* de Selecció.

Els Factors de Selectivitat amb els que experimentarem són 1%, 3%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% i 100%. Cal recalcar que quan el factor de selectivitat és petit, aleshores és també important fer més experiments per a detectar la sensibilitat dels *frameworks* davant una petita selecció de les dades. Cal notar també que moltes cerques que

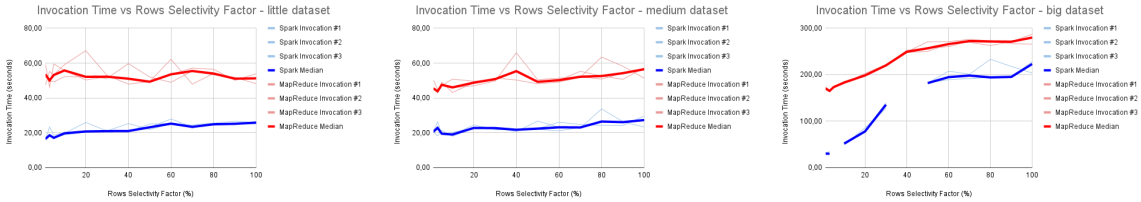
es faran sobre *logs*, no tindran Factors de Selectivitat massa grans, per tant, és important saber el rendiment dels *frameworks* en aquests cassos per a veure el seu rendiment.

Executarem el *pipeline* per al conjunt de dades petit, mitjà i gran per a veure el comportament dels dos sistemes amb diferents grandàries del conjunt de dades a processar.

Per a aquest experiment, mesurarem els valors de totes les mètriques especificades a la secció 5.1.

Resultats

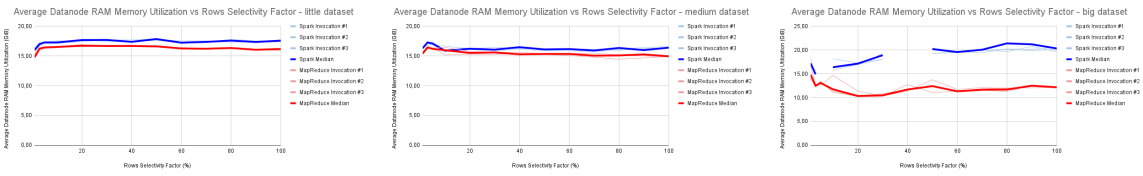
En aquest apartat, mostrarem els resultats per cada una de les mètriques abans presentades.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 21: Temps d'invocació vs. factor de selectivitat per files
[Font: elaboració pròpia]

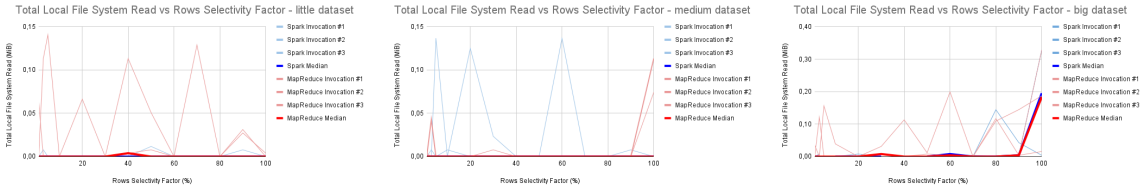
En la Figura 21 observem que per als tres conjunts de dades, MapReduce triga més temps per a fer les invocacions. Observem també que Spark és més estable que MapReduce al llarg de les tres invocacions. En el cas del conjunt de dades gran, Spark va fallar en dues ocasions, per al factor de selectivitat de 5 % i 40 % per problemes a l'hora de crear la consulta en SQL. Si mirem com es dissenya la consulta en Spark SQL, veurem que les cometes que converteixen el valor en cadena de text es poden anular en funció de l'entrada. Aquest problema és similar a la injecció SQL que es produeix en els sistemes relacionals. En MapReduce, no hi ha aquest problema, ja que no estem treballant amb SQL en cap moment.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 22: Utilització mitjana de memòria RAM dels nodes executors vs. factor de selectivitat per files
[Font: elaboració pròpia]

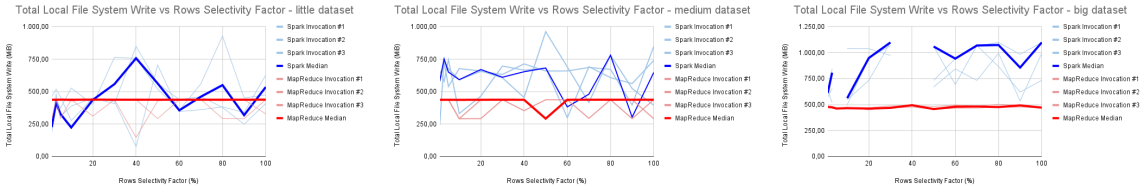
En la Figura 22 observem que MapReduce utilitza en els casos dels conjunts de dades petit i mitjà al voltant del 6 % menys memòria RAM que Spark. En el cas del conjunt de dades mitjà, aquesta diferència es redueix a un 5,4 % de mitjana aproximadament. En el cas del conjunt de dades gran, MapReduce utilitza fins un 58,9 % de mitjana menys memòria RAM que Spark.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 23: Utilització del disc local a cada node executor per a lectura vs. factor de selectivitat per files
[Font: elaboració pròpia]

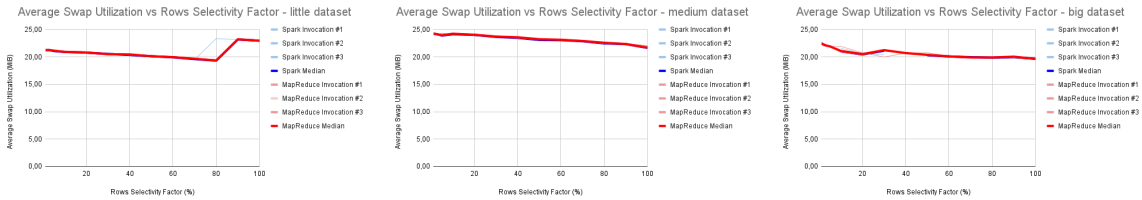
En la Figura 23 podem observar que la utilització del disc local, per part dels dos *frameworks* de processament de dades no és rellevant. Per tant, observem que el valor d'aquest indicador no és rellevant.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 24: Utilització del disc local a cada node executor per a escriptura vs. factor de selectivitat per files
[Font: elaboració pròpia]

En la Figura 24 destaca molt l'estabilitat de MapReduce, comparant-lo amb Spark. Podem observar en totes tres gràfiques que Spark és lleugerament més inestable que MapReduce en quant a l'escriptura. Cal destacar que Spark no necessitaria realment escriure a disc les dades, per tant, el més probable és que estigui escrivint arxius temporals. En el cas de MapReduce, tampoc hauria d'escriure al disc ja que el Mapper no ha d'intercanviar cap dada amb els Reducers perquè aquests estan anul·lats, encara que podem veure una certa escriptura. Per tant, podem utilitzar la mateixa justificació que per a Spark i raonar que s'estan escrivint arxius temporals. Més concretament, segons [80], es pot observar que en l'opció spark.local.dir de Spark, aquest utilitza l'espai per si ha de refer els càlculs des de 0 incloent els arxius a la sortida del Map i els RDDs que s'emmagatzemen a disc.

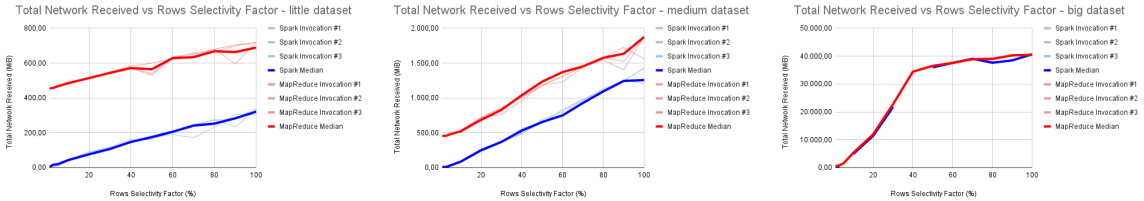


(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 25: Utilització de la memòria *swap* vs. factor de selectivitat per files
[Font: elaboració pròpia]

En la Figura 25 podem observar que en els dos *frameworks*, les gràfiques estan pràcticament

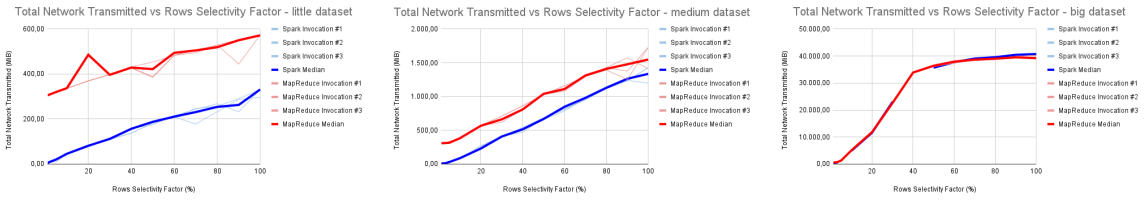
solapades, per tant, en aquest cas, com no podem isolar la memòria d'intercanvi que està utilitzant cada *framework*, aquesta memòria segurament està necessitada pel sistema operatiu i, segurament no depèn del *framework* que estem utilitzant. Tal com vam explicar a la secció 5.1, estem executant els experiments de forma entrellaçada i, en el cas en el que les dues gràfiques siguin molt iguals, voldrà dir que no depèn del *framework* el resultat de la gràfica, i aquest és el cas en aquestes gràfiques.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 26: Recepció d'informació a través de la xarxa vs. factor de selectivitat per files [Font: elaboració pròpia]

En la Figura 26, podem observar que MapReduce en el conjunt de dades petit rep de mitjana un 1330 % més de dades a través de la xarxa que Spark. En el cas del conjunt de dades mitjà, MapReduce rep de mitjana un 908 % de dades més que Spark. Per al conjunt de dades gran, aquesta diferència és només del 35 %. I, de fet, en el gràfic, podem observar també com es reflecteixen aquests percentatges. Cal destacar que l'ús de la xarxa és molt important en els sistemes distribuït perquè és bàsicament l'eina que estan utilitzant per a sincronitzar-se entre ells. Si aquest medi es satura, aleshores, els nodes no es podran comunicar de forma eficient entre ells. Aquesta millora significativa per part de Spark és, segons la seva documentació oficial a [80], per defecte té activada aquesta opció: spark.broadcast.compress i s'utilitza el còdec LZ4 per a fer una compressió de totes les dades que es transmeten a través de la xarxa. Aquests són paràmetres que estan per defecte, és a dir una persona que no té experiència amb Spark, té aquesta característica activa i no com passa amb MapReduce, que no la té, tal com podem observar en els gràfics.

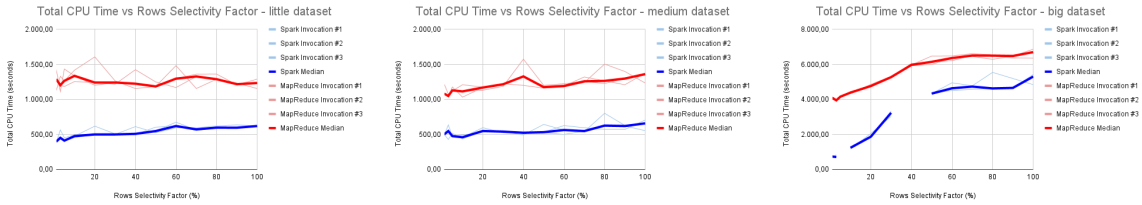


(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 27: Transmissió d'informació a través de la xarxa vs. factor de selectivitat per files [Font: elaboració pròpia]

En la Figura 27 podem observar la transmissió d'informació i, visualment observem que es segueix el mateix patró que en el cas de la recepció d'informació.

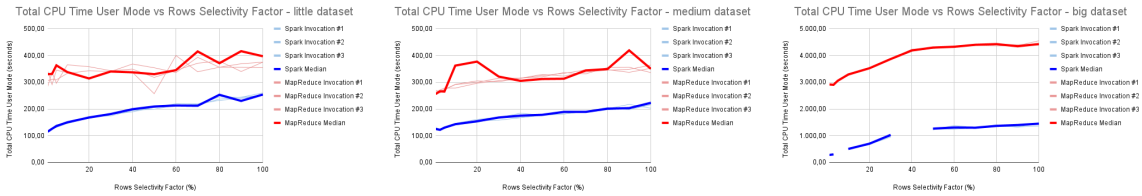
En la Figura 28 observem que MapReduce acostuma a utilitzar una mitjana del 145 % més temps de CPU que Spark en el conjunt de dades petit, un 120 % en el cas del conjunt de dades mitjà i un 145 % en el cas del conjunt de dades gran. Pot semblar més petita la



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 28: Temps Total de CPU vs. factor de selectivitat per files
[Font: elaboració pròpia]

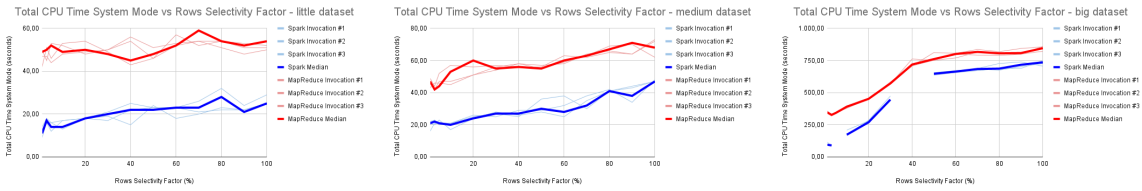
diferència, encara que cal recalcar que aquest percentatge és una mitjana i, al començament, quan el factor de selectivitat és molt baix, aleshores, MapReduce utilitza al voltant del 453 % més temps de CPU total que Spark.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 29: Temps de CPU en el *User Mode* vs. factor de selectivitat per files
[Font: elaboració pròpia]

En la Figura 29 podem observar el temps de CPU dedicat a executar codi en el mode Usuari, és a dir, codi que no pertany al *kernel* del Sistema Operatiu. Veiem que al igual que en la Figura 28, MapReduce consumeix més temps de CPU per al mode Usuari. En el conjunt de dades petit , MapReduce utilitza un 99 % més memòria RAM que Spark, en el cas del conjunt de dades mitjà aquest percentatge es redueix al 96 % i, finalment, en el cas del conjunt de dades gran, MapReduce utilitza un 398 % més memòria RAM que Spark.

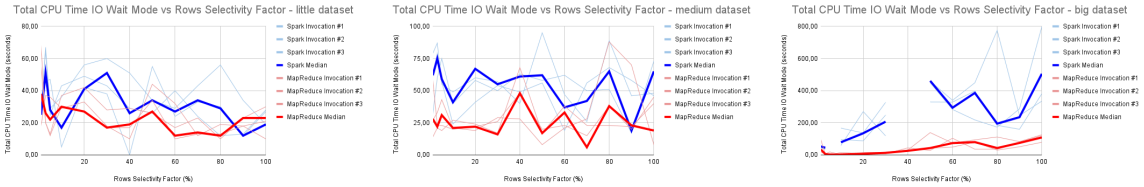


(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 30: Temps de CPU en el *System Mode* vs. factor de selectivitat per files
[Font: elaboració pròpia]

En la Figura 30 observem que es segueix un patró similar a l'observat en les Figures 28 i 29, encara que els percentatges que MapReduce utilitza més temps de CPU en el mode Sistema són 172 %, 103 % i 76 % per als conjunts de dades petit, mitjà i gran, respectivament. Podem observar, per tant, que quan la grandària d'un conjunt de dades fa que el nombre de crides sigui menys rellevant, aleshores s'iguala el temps de CPU en el mode Sistema dels dos *frameworks*. Si els conjunts de dades són més petits, aleshores, l'*overhead* de crides que fa

MapReduce tingui més rellevància.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 31: Temps de CPU en el *IO Wait Mode* vs. factor de selectivitat per files
[Font: elaboració pròpia]

En la Figura 31 podem veure que en alguns casos, MapReduce arriba a consumir més temps de CPU en el mode *IO Wait*, encara que la majoria del temps veiem que Spark és el sistema que fa que la CPU estigui en aquest mode. De mitjana Spark utilitza un 54 %, 169 % i un 831 % temps més que MapReduce la CPU en aquest mode. La conclusió a la que vam arribar és que Spark fa operacions utilitzant la memòria RAM de forma molt ràpida, de forma que la CPU ha d'esperar per a l'escriptura a disc.

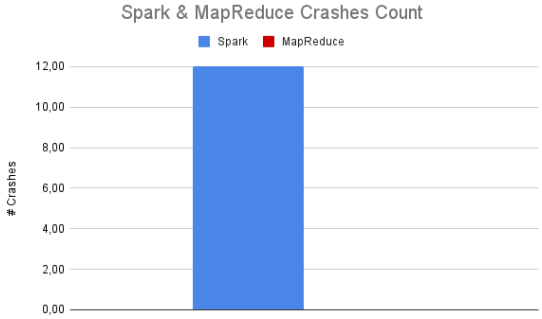


Figura 32: Nombre de fallades a cada *framework*
[Font: elaboració pròpia]

En la Figura 32 observem que MapReduce és més robust que Spark perquè no té les fallades causades per les consultes en SQL, que tal com vam mencionar a la descripció de la Figura 21, existeix el mateix problema que amb el SQL de les bases de dades relacionals, on depenent de l'entrada del conjunt de dades, en la consulta SQL, es poden anul·lar les cometes i es pot produir un error, de forma similar a com funciona la SQL *Injection* [81].

5.2.2 Operació de Projecció

Objectiu

L'objectiu d'aquest experiment és mesurar les mètriques especificades en la secció 5.1 per diferents execucions de la projecció, projectant una quantitat diferent de columnes i sobre els tres conjunts de dades discutits anteriorment. Aquesta operació de projecció consisteix en filtrar quines columnes del conjunt d'entrada passen al conjunt de sortida; aquesta operació no canvia el nombre de files, només el nombre de columnes a la sortida respecte a l'entrada.

Aquesta operació és considerada, al igual que la selecció, una *narrow dependency*, és a dir que no és necessari intercanviar dades a través de la xarxa i, a més, es pot executar de

forma distribuïda sense necessitat d'interaccionar amb els altres nodes. El rendiment hauria de ser bo per a Spark ja que no necessitaria intercanviar dades a través de la xarxa i es podria executar de forma distribuïda sense necessitat d'interaccionar amb els altres nodes. Respecte a MapReduce, aquesta operació està implementada amb un Mapper, per la qual cosa tampoc seria necessari un intercanvi de dades per a transportar les dades als Reducer, la que aquests estan anul·lats. En ambdós casos es tractaria de l'execució d'un únic *job* i, per tant, esperem un bon rendiment en tots dos sistemes.

Configuració

El *pipeline* que executarem per a aquest experiment, variant la quantitat de columnes seleccionades és el *pipeline* de Projecció.

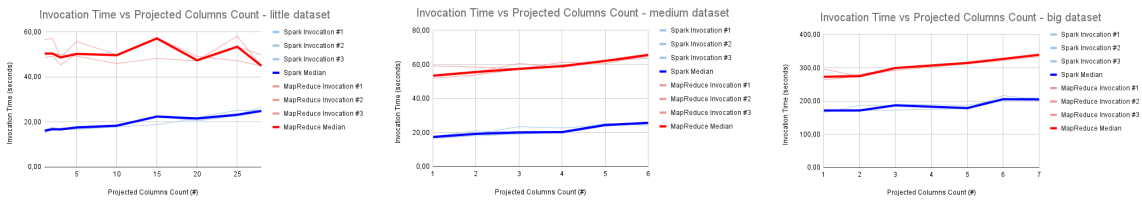
Per als conjunts de dades mitjà i gran, experimentarem amb totes les columnes perquè no disposen de massa columnes. En canvi, per al conjunt de dades petit, que té 28 columnes, experimentarem amb 1, 2, 3, 5, 10, 15, 20, 25 i 28 columnes projectades. Cal recalcar que quan el factor de selectivitat columnar és petit; és a dir que es projecten poques columnes, aleshores és també important fer més experiments per a detectar la sensibilitat dels *frameworks* en aquest escenari. Moltes de les consultes que es facin sobre *logs* poden fer projeccions de poques columnes, per tant, és important saber el rendiment dels *frameworks* en aquests casos per a veure el seu rendiment.

Executarem el *pipeline* per al conjunt de dades petit, mitjà i gran per a veure el comportament dels dos sistemes amb diferents grandàries del conjunt de dades a processar.

Per a aquest experiment, mesurarem els valors de totes les mètriques especificades a la secció 5.1.

Resultats

En aquest apartat, mostrarem els resultats per cada una de les mètriques abans presentades.

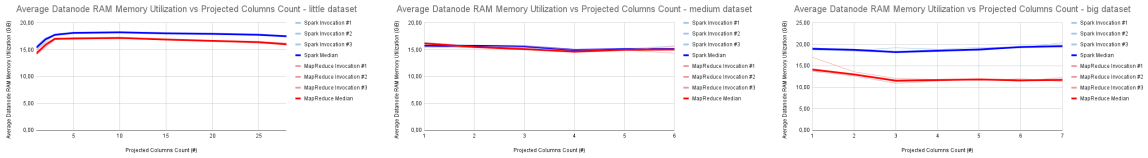


(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 33: Temps d'invocació vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

En la Figura 33 observem que el temps d'invocació per part de MapReduce torna a ser considerablement més gran que Spark, al igual que passava en l'experiment de selecció. Específicament, per al conjunt de dades petit, MapReduce necessita un 160 % més que Spark per a fer la invocació de les mateixes operacions. Per al conjunt de dades mitjà, aquest percentatge és de 180 % i, en el cas del conjunt de dades gran, el percentatge és del

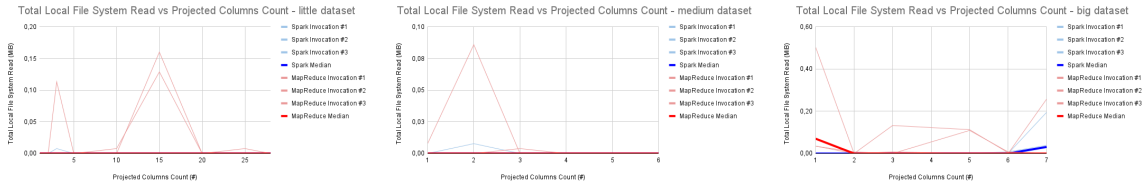
63 %. Observem que per aal conjunt de dades gran, Spark es comença a assemblar més a MapReduce. És possible que utilitzant conjunts de dades encara més grans que el conjunt de dades gran que estem utilitzant, hi pugui arribar un moment en el que tinguin un temps molt similar. Cal destacar que els conjunts de dades petit i mitjà tenen una mida similar, i és per això que el temps d'invocació és també molt similar.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 34: Utilització mitjana de memòria RAM dels nodes executors vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

En la Figura 34 observem com en el conjunt de dades petit, Spark utilitza només un 7 % més memòria RAM que MapReduce, mentre que per al conjunt de dades mitjà, Spark utilitza de mitjana només un 1 % més que MapReduce i, en algunes determinades execucions del *Testbed*, és MapReduce el sistema que necessita més memòria RAM que Spark. Aquesta similitud es pot donar pel fet que els conjunts de dades petit i mitjà tenen una grandària similar. En canvi, si observem el conjunt de dades gran, observem clarament que MapReduce està utilitzant una mitjana de 55 % menys memòria RAM que Spark. Observem en el sistema MapReduce que en el conjunt de dades gran està fins i tot utilitzant menys memòria RAM que en el conjunt de dades petit. Aquesta diferència pot estar subjecta també al fet que la mètrica recollida de la memòria RAM està recollida a nivell general del sistema operatiu, és a dir que no tenim en compte la memòria que està utilitzant el *framework* només, sinó que també estem tenint en compte la memòria que utilitzen altres processos que s'estan executant. Hem intentat reduir aquest esbiaix entrellaçant l'execució dels experiments entre els dos sistemes, tal com vam mencionar en la secció 5.1.

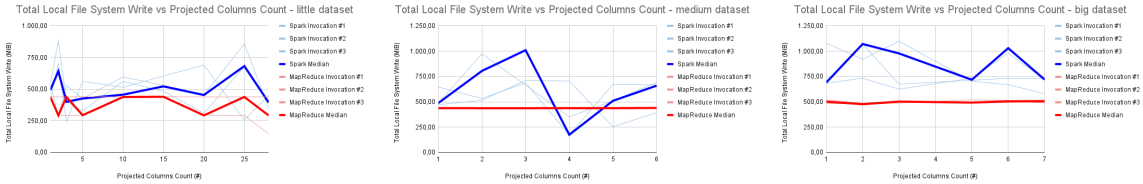


(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 35: Utilització del disc local a cada node executor per a lectura vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

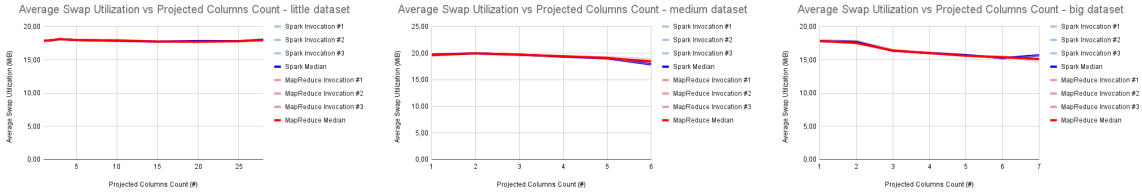
En la Figura 35 no podem apreciar cap ús important del disc local per a lectura, tant per part de MapReduce com per part de Spark.

En la Figura 36 observem que els dos sistemes estan escrivint moltes més dades que les que es llegeixen, tal com podem apreciar en la Figura 35. Com vam mencionar en l'experiment de la selecció, aquesta escriptura la podem relacionar amb l'escriptura d'arxius temporals.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

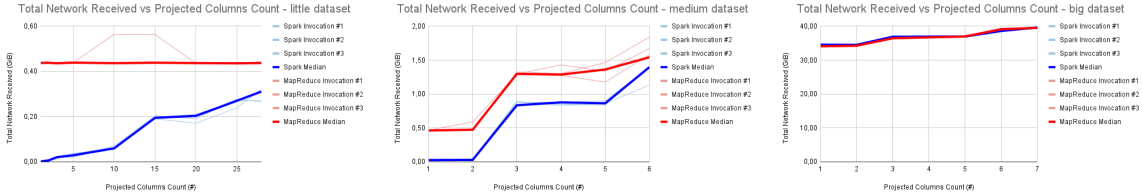
Figura 36: Utilització del disc local a cada node executor per a escriptura vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 37: Utilització de la memòria *swap* vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

En la Figura 37 tornem a observa al igual que en l'experiment de l'operació de la selecció que la memòria d'intercanvi no depèn del *framework* de processament de dades que estiguem utilitzant.

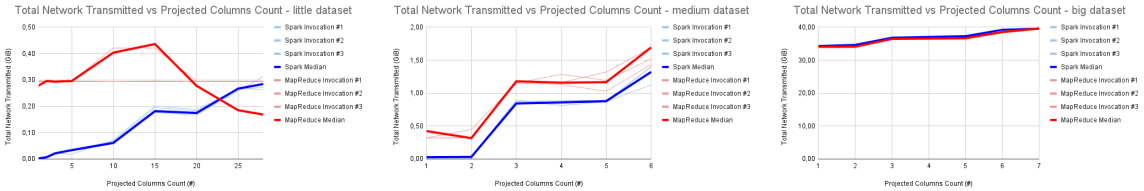


(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 38: Recepció d'informació a través de la xarxa vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

En la Figura 38 observem que en el conjunt de dades petit, MapReduce té un ús pràcticament constant de la recepció de dades a través de la xarxa. En mitjana, per al conjunt de dades petit, MapReduce està utilitzant un 2.193 % més la xarxa que Spark. En teoria, l'operació de selecció i projecció, com són considerades *narrow dependencies*, aquestes no haurien de fer cap ús de la xarxa. En canvi, podem observar al gràfic que sí que està utilitzant la xarxa i, a més aquest ús depèn de quantes columnes estiguem projectant. Es pot observar en les gràfiques dels tres conjunts de dades que la recepció d'informació a través de la xarxa depèn de la quantitat de columnes projectades. En el cas del conjunt de dades mitjà, MapReduce està utilitzant un 413 % de mitjana més la xarxa que Spark. Finalment, en el cas del conjunt de dades gran, podem observar que fins i tot Spark està utilitzant lleugerament més la xarxa per a rebre un 1 % de mitjana més informació que MapReduce.

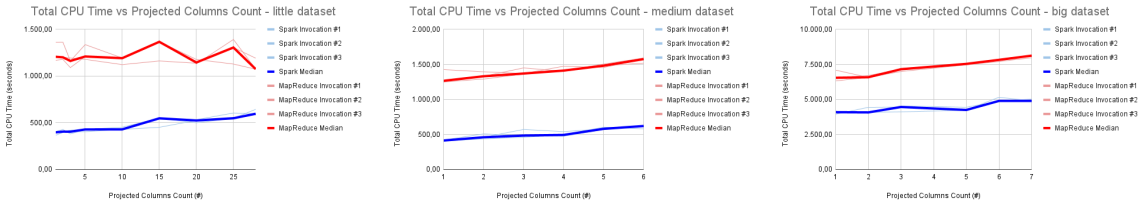
En la Figura 39 observem que en el cas del conjunt de dades petit, Spark arriba a



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 39: Transmissió d'informació a través de la xarxa vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

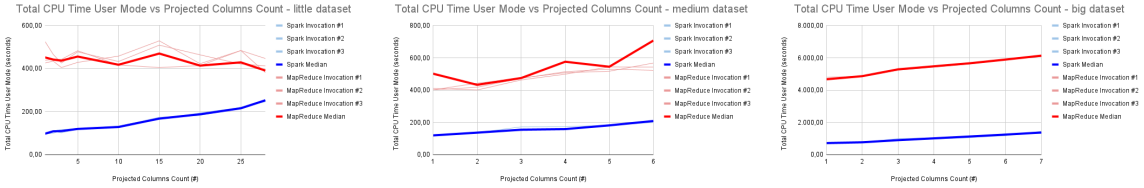
transmetre més informació a través de la xarxa que MapReduce i podem apreciar que aquesta gràfica s'assembla a les gràfiques de la Figura 38 per Spark però no per a MapReduce per al conjunt de dades petit. Tot i així, Spark transmet per al conjunt de dades petit un 2193 % menys dades que MapReduce. Per al conjunt de dades mitjà, aquest percentatge es redueix a un 413 % i, per al conjunt de dades gran passa al igual que en la Figura 38, on Spark utilitza un 1 % més xarxa que MapReduce.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 40: Temps Total de CPU vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

En la Figura 40, MapReduce està utilitzant més temps de CPU que Spark. Per al conjunt de dades petit, MapReduce utilitza un 159 % més de temps de CPU en tots els modes que Spark. Per al conjunt de dades mitjà, el percentatge incrementa a un 178 %, mentre que per al conjunt de dades gran, el percentatge disminueix a un 64 %. Observem que per al conjunt de dades gran, Spark utilitza més recursos i s'assembla més a MapReduce que per als conjunts de dades petits.

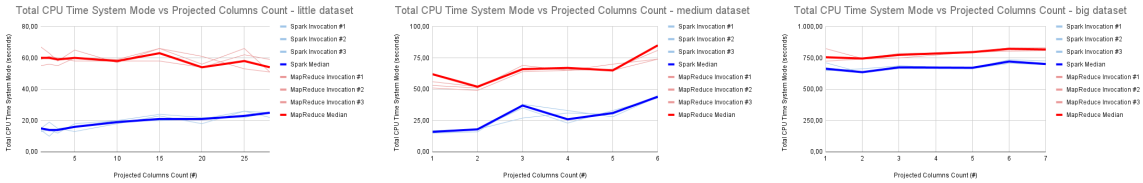


(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 41: Temps de CPU en el *User Mode* vs. Nombre de Columnes Projectades
[Font: elaboració pròpia]

En la Figura 41, Spark segueix utilitzant una menor quantitat de temps de CPU, en aquest cas en mode Usuari. Si mirem les gràfiques, aquestes segueixen un patró similar

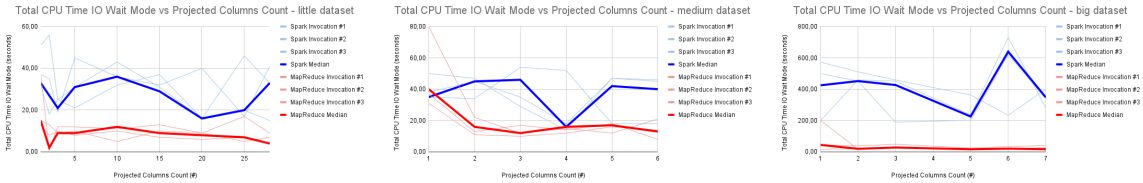
a la Figura 40. En el conjunt de dades petit, Spark utilitza un 213 % menys temps que MapReduce, en el conjunt de dades mitjà, el percentatge augmenta a un 240 % i per al conjunt de dades gran, el percentatge creix a un 446 %.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 42: Temps de CPU en el *System Mode* vs. Nombre de Columnes Projectades [Font: elaboració pròpia]

En la Figura 42 podem observar que per al conjunt de dades petit i mitjà, MapReduce utilitza més temps de CPU que Spark, concretament 228 % i 152 %. Per al conjunt de dades gran, observem que Spark s'apropa molt a MapReduce i veiem que MapReduce utilitza només un 15 % més de temps en aquest mode de CPU que Spark. Observem per tant, que per a conjunts de dades grans, tant MapReduce com Spark provoquen que s'executi aproximadament al voltant de la mateixa quantitat de codi del *kernel* a través de les crides a sistema. Tot i així, en general, el nombre de crides a sistema sembla ser més reduït en el cas de Spark.



(a) Conjunt de dades petit (b) Conjunt de dades mitjà (c) Conjunt de dades gran

Figura 43: Temps de CPU en el *IO Wait Mode* vs. Nombre de Columnes Projectades [Font: elaboració pròpia]

En la Figura 43 observem que Spark utilitza més la CPU en aquest mode que MapReduce. Per tant obtenim unes gràfiques molt similars a les de la Figura 31. En el conjunt de dades petit, Spark utilitza un 353 % més temps de CPU en mode *IO Wait* que MapReduce. En el conjunt de dades mitjà, aquest percentatge es redueix a un 134 %. Però, la diferència es converteix en molt important per al conjunt de dades gran, on el percentatge és 1759 %. Observem que Spark no està aprofitant tant el processador al mantenir-lo en aquest mode.

En la Figura 44 observem que en totes les execucions, no ha sorgit cap error.

5.2.3 Operació de Selecció i Projecció

Objectiu

L'objectiu d'aquest experiment és mesurar les mètriques especificades en la secció 5.1 per diferents execucions de la selecció i de la projecció amb diferents factors de selectivitat i diferents nombres de columnes projectades sobre els tres conjunts de dades discutits ante-

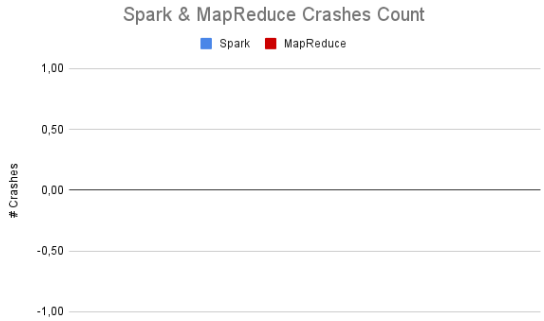


Figura 44: Nombre de fallades a cada *framework*
[Font: elaboració pròpia]

riorment. Primer executarem una selecció i, sobre el resultat de la selecció, aplicarem una projecció.

Aquestes operacions són considerades *narrow dependencies*, és a dir que no és necessari intercanviar dades a través de la xarxa i, a més, es pot executar de forma distribuïda sense necessitat d'interaccionar amb els altres nodes. El rendiment hauria de ser bo per a Spark, ja que no es necessitaria enviar dades a través de la xarxa i es podria executar de forma distribuïda sense necessitat d'interaccionar amb els altres nodes. Contràriament, el rendiment del MapReduce hauria de ser pitjor ja que aquest ha de posar el resultat de la selecció a HDFS i l'operació de projecció els ha de llegir. Tant l'operació de selecció com la de projecció estan implementades amb un Mapper, encara que escriure i llegir a disc hauria d'impactar significativament el rendiment. A més, en el cas de Spark, s'estaria executant un únic *job*, mentre que en el cas de MapReduce, ens veiem forçats a executar dos *jobs*.

Configuració

El *pipeline* que executarem per a aquest experiment, variant el factor de selectivitat i el nombre de columnes projectades és el *pipeline* de Selecció i Projecció combinades.

Els Factors de Selectivitat amb els que experimentarem són 1%, 3%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% i 100%. Per als conjunts de dades mitjà i gran, experimentarem amb totes les columnes perquè no disposen de massa columnes. En canvi, per al conjunt de dades petit, que té 28 columnes, experimentarem amb 1, 2, 3, 5, 10, 15, 20, 25 i 28 columnes projectades. Tot i així, per a fer més clars els gràfics, reportarem tant les invocacions com la mediana de només una columna i totes les columnes projectades.

A més de les mètriques indicades en la secció 5.1, vam decidir incloure les següents mètriques:

- **Escriptura en el sistema d'arxius distribuït (és igual a la lectura) sense replicació.** Aquesta mètrica és especialment important en aquest experiment, a diferència de la resta d'experiment, on aquesta mètrica no està inclosa, pel fet que al tenir encadenades dues operacions, en el cas del *framework* MapReduce, aquest es veu forçat a escriure a disc els resultats del *Select*, per a que l'operació de projecció els llegeixi de disc i els processi. Això no és necessari en el cas de Spark perquè aquest comparteix

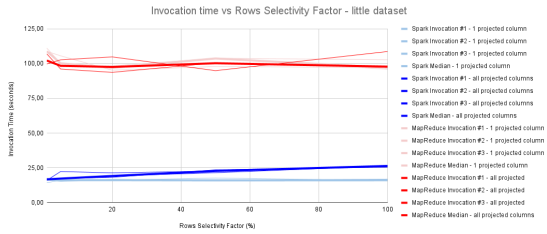
els resultats intermedis a la memòria RAM dels nodes executors. Aleshores, veurem un valor de 0 en el cas de Spark. Com a resultats intermedis ens referim al conjunt de dades resultant de l'operació de *Select*.

- **Escriptura en el sistema d'arxius distribuït amb replicació.** Aquest valor és també important perquè ens reporta la grandària total dels resultats intermedis total, tenint en compte el factor de replicació o quants cops està disponible un mateix fragment del conjunt de dades en el clúster.

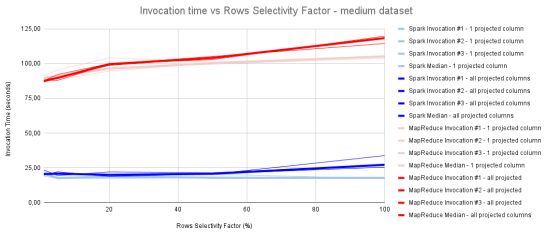
Executarem el *pipeline* per al conjunt de dades petit, mitjà i gran per a veure el comportament dels dos sistemes amb diferents grandàries del conjunt de dades a processar.

Resultats

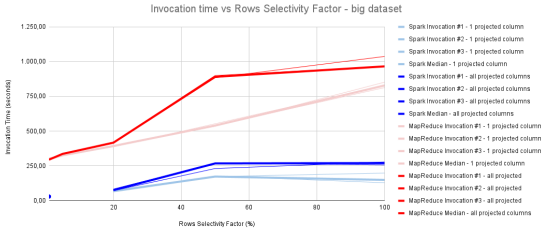
En aquest apartat, mostrarem els resultats per cada una de les mètriques abans presentades.



(a) Conjunt de dades petit



(b) Conjunt de dades mitjà

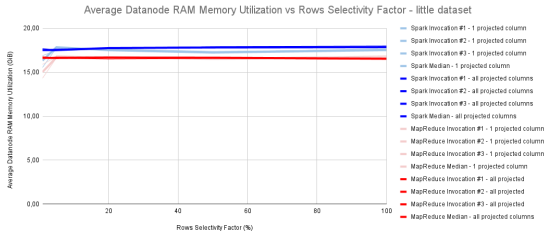


(c) Conjunt de dades gran

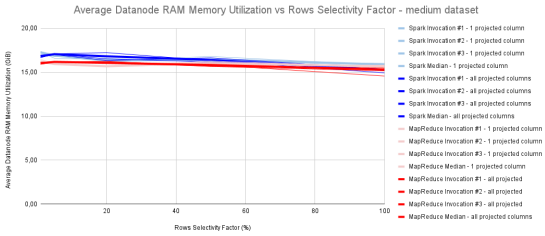
Figura 45: Temps d'invocació vs. factor de selectivitat per files
[Font: elaboració pròpia]

En la Figura 45 observem que el temps d'invocació és més gran en MapReduce que en Spark. En concret, si mirem la projecció d'una única columna, Spark és un 527 % més ràpid en fer les invocacions que Spark i un 400 % a l'hora de processar el conjunt de dades petit. Per al conjunt de dades mitjà i gran, el percentatge manté el mateix ordre de magnitud. En el cas del conjunt de dades mitjà, per a una columna projectada, Spark és un 422 % més ràpid i un 359 % en el cas de projectar totes les columnes. A l'hora de processar el conjunt de dades gran, projectant una columna, obtenim que Spark és un 534 % més ràpid i un 426 % quan es projecten totes les columnes.

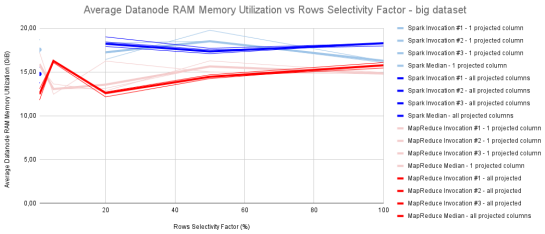
En la Figura 46 observem que tot i que l'ús que fa Spark de la memòria RAM és més elevat que MapReduce, aquest últim tampoc està a una distància massa gran. Per a ser precisos, MapReduce consumeix al voltant d'un 6 % menys memòria RAM que Spark per al



(a) Conjunt de dades petit



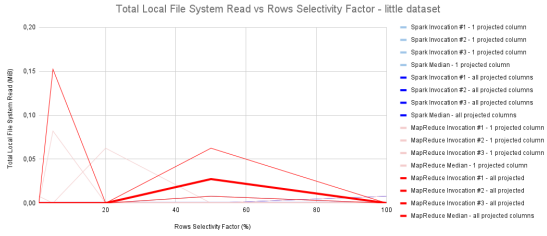
(b) Conjunt de dades mitjà



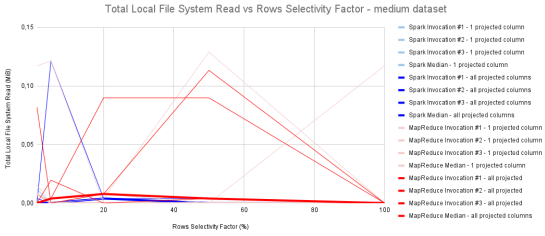
(c) Conjunt de dades gran

Figura 46: Utilització mitjana de memòria RAM dels nodes executors vs. factor de selectivitat per files
[Font: elaboració pròpia]

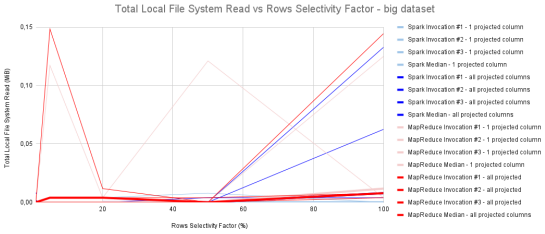
conjunt de dades petit. Per al conjunt de dades mitjà, el la utilització de la memòria RAM està al voltant del 4 % menys utilització per part de MapReduce. En el cas del conjunt de dades gran, la diferència és més gran principalment pel fet que MapReduce utilitza menys memòria RAM que fins i tot a l'hora de processar el conjunt de dades petit. Per al conjunt de dades gran, MapReduce utilitza entre un 16 % i un 25 % menys memòria RAM, en funció de quantes columnes estem projectant, és a dir, una columna i totes les columnes projectades, respectivament.



(a) Conjunt de dades petit



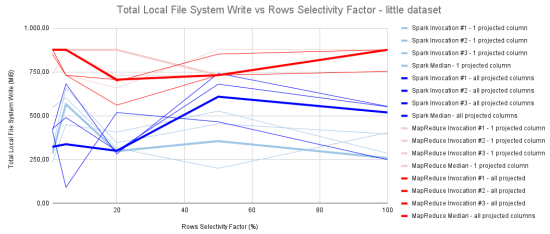
(b) Conjunt de dades mitjà



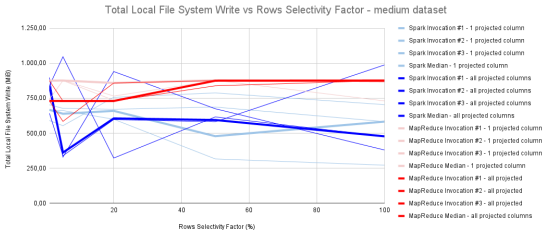
(c) Conjunt de dades gran

Figura 47: Utilització del disc local a cada node executor per a lectura vs. factor de selectivitat per files
[Font: elaboració pròpia]

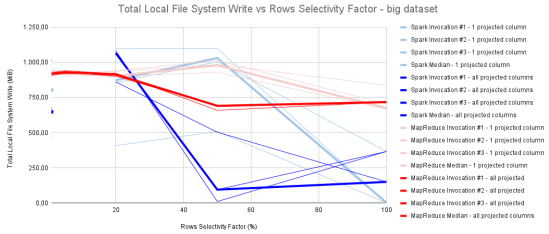
En la Figura 47 tornem a observar al igual que en els experiments anteriors que la lectura del disc no té gaire importància perquè estem concatenant dues operacions que realment són *narrow dependencies*. Així que en el cas de Spark, aquestes no suposaran un ús del disc local. En el cas del MapReduce tampoc suposa un ús d'aquest recurs perquè ni l'operació de selecció ni l'operació de projecció s'implementen utilitzant algun Reducer. Compartir les dades entre les operacions es fa utilitzant el disc, però en aquest cas s'està utilitzant HDFS. Per tant, l'ús d'aquest recurs no és massa rellevant.



(a) Conjunt de dades petit



(b) Conjunt de dades mitjà



(c) Conjunt de dades gran

Figura 48: Utilització del disc local a cada node executor per a escriptura vs. factor de selectivitat per files
[Font: elaboració pròpia]

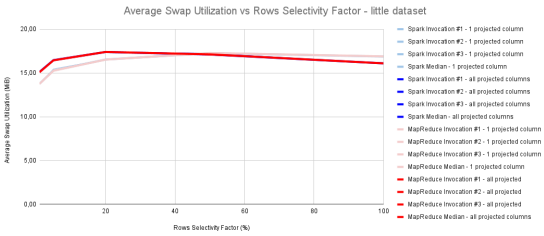
En la Figura 48 observem al igual que en els experiments anteriors que existeix una escriptura, que al igual que abans, podríem interpretar que és per a arxius temporals.

En la Figura 49 observem un altre cop que la memòria d'intercanvi no depèn del *framework* de processament de dades que estiguem utilitzant.

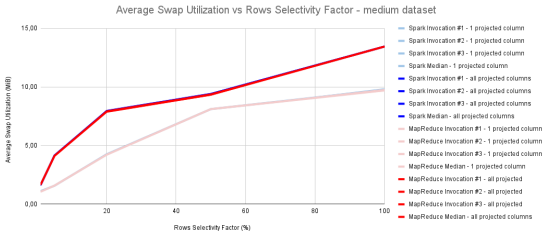
En la Figura 50 observem que MapReduce fa un ús més intensiu que la xarxa, al igual que vam observar en els experiments anteriors. La justificació seria bàsicament la mateixa que es va seguir a la Figura 26.

En la Figura 51 tornem a observar un patró d'ús de la xarxa similar al vist en la Figura 26.

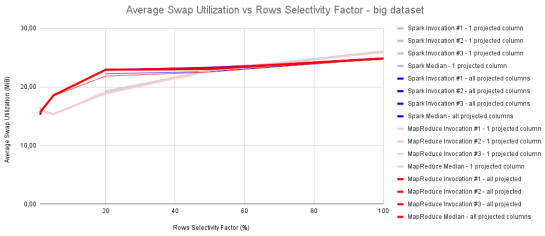
En la Figura 52 observem que per a tots els conjunts de dades, MapReduce utilitza un 528 % i un 396 % per a la projecció d'una i totes les columnes per al conjunt de dades petit, per al mitjà, els percentatges són 419 % i 355 % i per al conjunt de dades gran els percentatges són 537 % i 464 % més temps de CPU que Spark. Com ja vam justificar en anteriors experiments, aquest ús de la CPU l'associem al *overhead* de crear els *jobs*. En aquest experiment en particular, Spark executa un únic *job*, mentre que MapReduce està executant dos *jobs* diferents i és per això perquè veiem un percentatge d'utilització més elevat que en anteriors experiments.



(a) Conjunt de dades petit

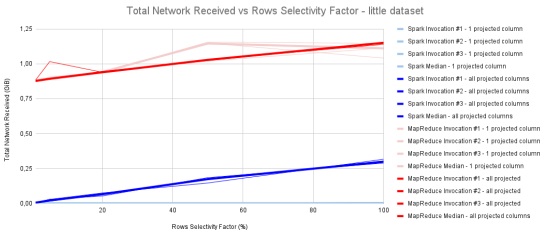


(b) Conjunt de dades mitjà

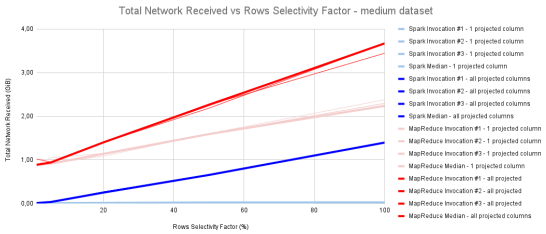


(c) Conjunt de dades gran

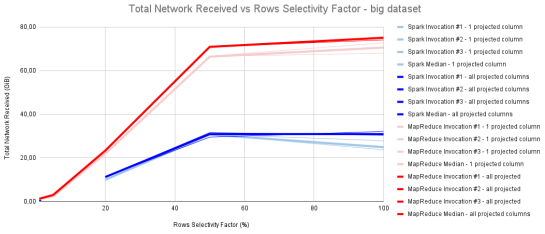
Figura 49: Utilització de la memòria *swap* vs. factor de selectivitat per files [Font: elaboració pròpia]



(a) Conjunt de dades petit



(b) Conjunt de dades mitjà

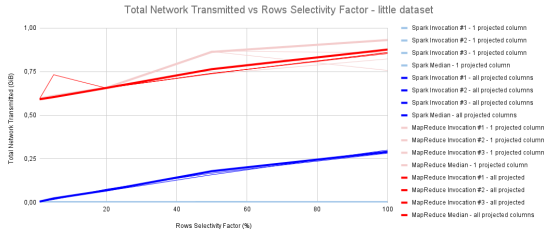


(c) Conjunt de dades gran

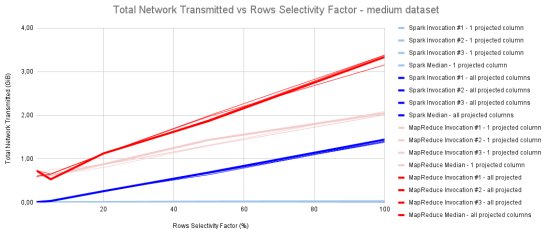
Figura 50: Recepció d'informació a través de la xarxa vs. factor de selectivitat per files [Font: elaboració pròpia]

En la Figura 53 observem que el fet de seleccionar una o diverses columnes afecta directament en les gràfiques. Addicionalment, cal recalcar que MapReduce està fent un ús molt més intensiu, per sobre del 1000 % més que Spark de la CPU en mode Usuari en el conjunt de dades gran. Aquesta diferència, segueix sent gran també en els conjunts de dades petit i mitjà, que tenen una utilització de la CPU en mode usuari de 618 % i 351 % per a una i totes les columnes projectades i de 406 % i 316 % per a una i totes les columnes projectades, respectivament.

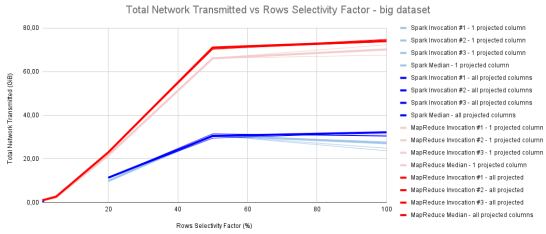
En la Figura 54 veiem clarament que MapReduce està utilitzant durant més temps la



(a) Conjunt de dades petit

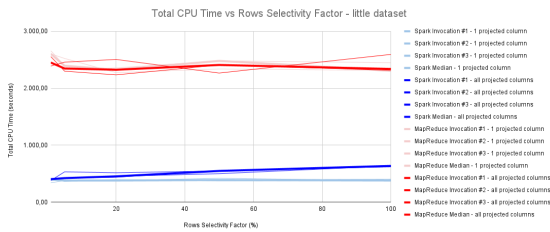


(b) Conjunt de dades mitjà

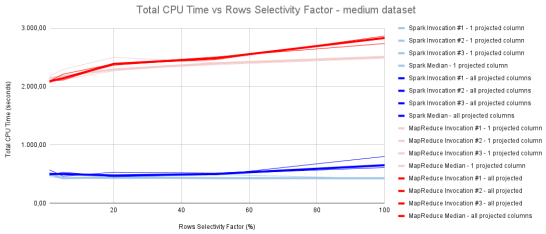


(c) Conjunt de dades gran

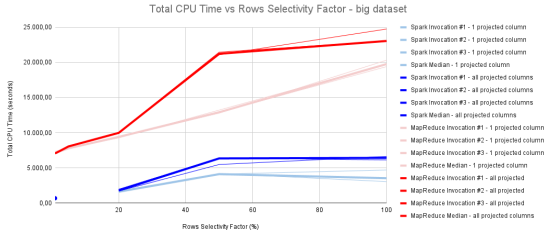
Figura 51: Transmissió d'informació a través de la xarxa vs. factor de selectivitat per files [Font: elaboració pròpia]



(a) Conjunt de dades petit



(b) Conjunt de dades mitjà

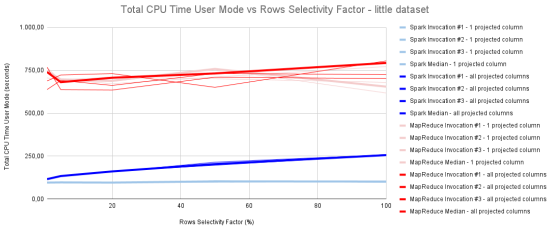


(c) Conjunt de dades gran

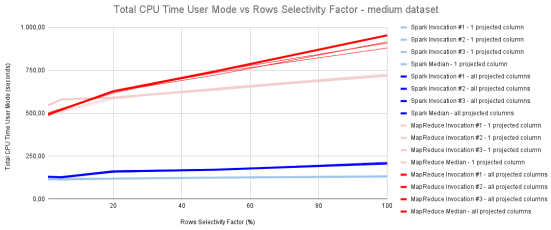
Figura 52: Temps Total de CPU vs. factor de selectivitat per files [Font: elaboració pròpia]

CPU en mode Sistema, que al igual que vam comentar en els anteriors experiments, vol dir que està fent més crides a sistema i provocant l'execució de més codi del *kernel*.

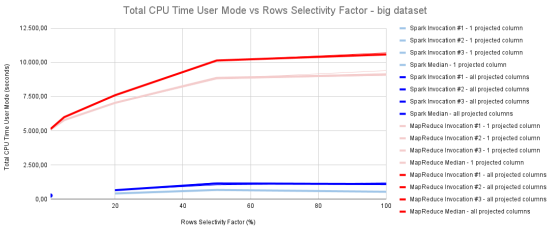
En la Figura 55 podem observar que tant Spark com MapReduce estan utilitzant aproximadament la mateixa quantitat de temps en aquest mode per als conjunts de dades petit i mitjà. Per al conjunt de dades gran, observem que en aquest cas és MapReduce el sistema que està fent un ús més intensiu de la CPU en aquest mode. La raó que vam trobar és que MapReduce, al haver de gestionar dos *jobs*, en comptes d'un, aquest genera més *overhead* perquè ha de comunicar la sortida d'un amb l'entrada de l'altre i, per això ha d'utilitzar el



(a) Conjunt de dades petit

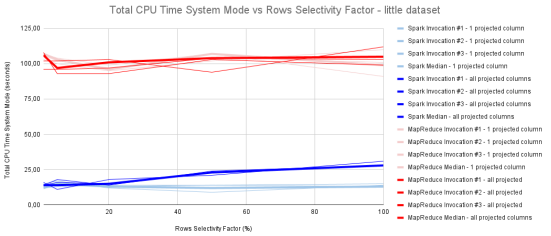


(b) Conjunt de dades mitjà

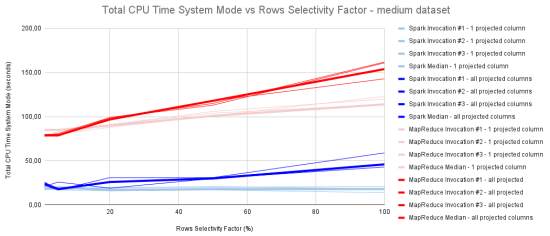


(c) Conjunt de dades gran

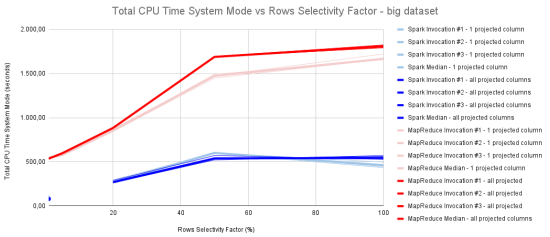
Figura 53: Temps de CPU en el *User Mode* vs. factor de selectivitat per files [Font: elaboració pròpia]



(a) Conjunt de dades petit



(b) Conjunt de dades mitjà

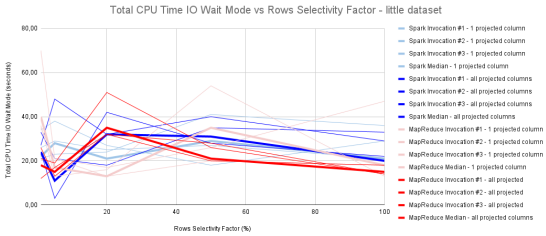


(c) Conjunt de dades gran

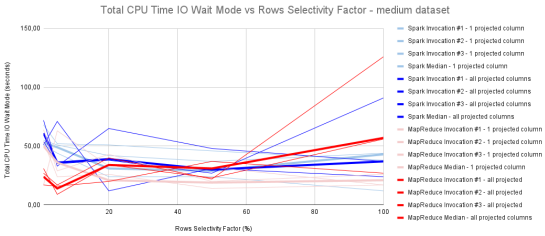
Figura 54: Temps de CPU en el *System Mode* vs. factor de selectivitat per files [Font: elaboració pròpia]

sistema d'arxius distribuït, que és HDFS.

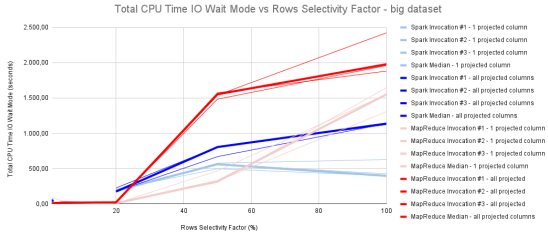
En la Figura 56 podem observar com MapReduce està fent ús del sistema d'arxius distribuït. També podem observar que quant més gran sigui el factor de selectivitat, més s'escriurà al sistema d'arxius distribuït. Per als conjunts de dades petit i mitjà veiem que hi ha una correlació lineal, on en el cas del conjunt de dades petit, al tenir un 100 % de factor de selectivitat, s'estan escrivint 131,55 MiB, mentre que per al conjunt de dades mitjà, s'estan escrivint 0,69 GiB. Observem també que en el conjutn de dades gran, hi ha un punt d'inflexió situat en el punt en el que tenim un Factor de selectivitat del 50 %. Segurament a partir



(a) Conjunt de dades petit

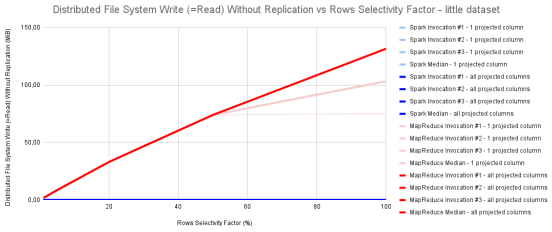


(b) Conjunt de dades mitjà

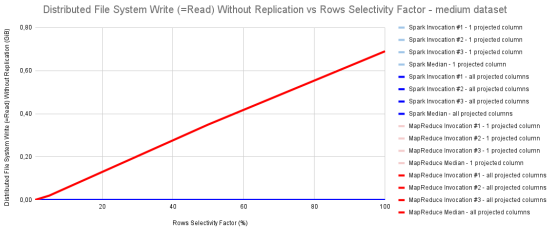


(c) Conjunt de dades gran

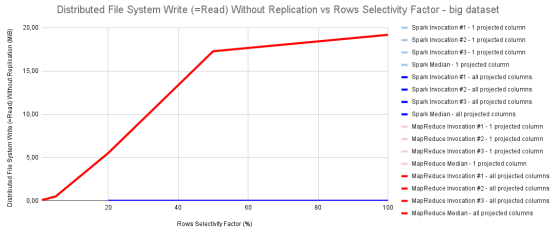
Figura 55: Temps de CPU en el *IO Wait Mode* vs. factor de selectivitat per files
[Font: elaboració pròpia]



(a) Conjunt de dades petit



(b) Conjunt de dades mitjà



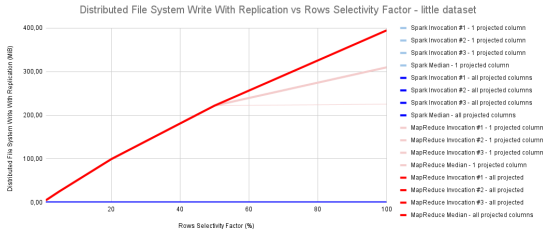
(c) Conjunt de dades gran

Figura 56: Escriptura en el sistema d'arxius distribuït (és igual a la lectura) sense replicació vs. factor de selectivitat per files
[Font: elaboració pròpia]

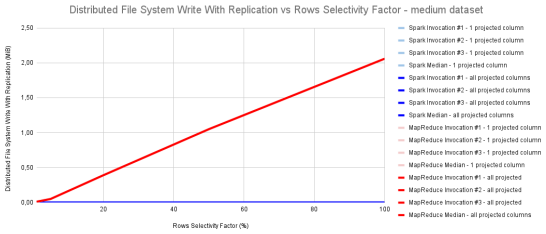
d'aquest punt, MapReduce comenci a fer algun tipus de compressió *lightweight* de forma es es comencen a reduir la quantitat de dades a escriure.

En la Figura 57 observem que es manté la mateixa forma que en la Figura 56 i veiem que els valors d'aquesta gràfica estan multiplicats per 3, que és el factor de replicació que s'està utilitzant a Hadoop per defecte per a millorar l'eficiència dels sistemes distribuïts.

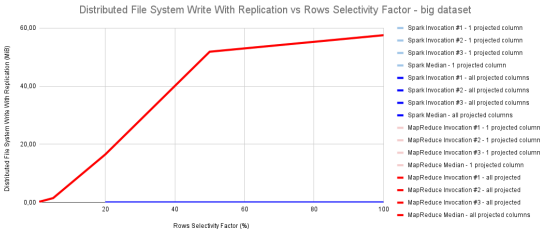
En la Figura 58 observem que els dos sistemes han sofert errors. En el cas de Spark, es repeteixen els mateixos errors detallats en la descripció de la Figura 32. En el cas de



(a) Conjunt de dades petit



(b) Conjunt de dades mitjà



(c) Conjunt de dades gran

Figura 57: Escriptura en el sistema d'arxius distribuït amb replicació vs. factor de selectivitat per files
[Font: elaboració pròpia]

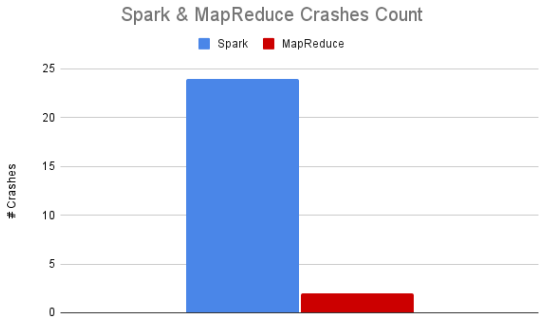


Figura 58: Nombre de fallades a cada *framework*
[Font: elaboració pròpia]

MapReduce, veiem que també hi ha 2 errors relacionats a com està gestionant MapReduce el format Parquet. Es tractava d'un mateix error que va sorgir en temps d'execució relacionat amb el fet que no es trobava la columna APPT_CANCEL_DATE del conjunt de dades petit. Aquest és un problema que va sorgir només per a aquest conjunt de dades i només per a dues execucions determinada del *Testbed*. Cal recalcar que són dos casos isolats perquè les dues execucions anteriors que utilitzaven exactament la mateixa configuració no van donar aquest error.

5.2.4 Operació de *Join*

Objectiu

L'objectiu d'aquest experiment és mesurar les mètriques especificades en la secció 5.1 per diferents execucions de la *join* amb diferents taules auxiliars i sobre el conjunt de dades mitjà. Aquesta operació és molt útil si es volen creuar dades de *logs* diferents.

Aquesta operació equival a fer un producte cartesià de dos conjunt de dades i seleccionar les files que compleixen un determinat predicat. Aquesta seria l'explicació a nivell lògic, encara que hi ha diferents algorismes que permeten implementar de forma eficient una *join*. Cal destacar que Spark pot decidir un Pla Físic, tal com vam veure a l'hora de demostrar la correctesa del *Testbed* a la secció 4.4, on pot escollir un algorismes de *join*, mentre que en MapReduce, ens hem de limitar a implementar un algorisme manualment que pot no ser tant eficient com el que esculli el planificador de les operacions integrat a Spark.

L'operació de *join* és una *wide dependency*, és a dir que s'han d'intercanviar necessàriament dades a través de la xarxa i no es pot executar de forma distribuïda sense necessitat d'interaccionar amb els altres nodes. Tot i així, tal com vam mencionar, el rendiment podria ser millor per part de Spark, encara que quedaria per veure a través dels experiments quin sistemes funcionaria millor en aquest cas d'ús.

Configuració

El *pipeline* que executarem per a aquest experiment és el *pipeline* de *join*.

Executarem el *pipeline* per al conjunt de dades mitjà i farem la *join* amb dos conjunts de dades auxiliars, anomenats Ad Feature i User Profile ja que és l'únic conjunt de dades dels seleccionats que venien amb conjunts de dades auxiliars que feien *join*.

En el cas del conjunt de dades Ad Feature, per a cada fila del conjunt de dades mitjà, aquesta fa *join*, de mitjana amb 32 files (agafant la part superior). La sortida estimada d'aquesta *join* és de 23,67 GiB, fent una projecció de la grandària del conjunt de dades mitjà.

En el cas del conjunt de dades User Profile, per a cada fila del conjunt de dades mitjà, aquesta fa *join*, de mitjana amb 12 files (agafant la part superior). La sortida estimada d'aquesta *join* és de 16,54 GiB, fent una projecció de la grandària del conjunt de dades mitjà.

Per a aquest experiment, mesurarem els valors de totes les mètriques especificades a la secció 5.1.

Resultats

En aquest apartat, mostrarem els resultats per cada una de les mètriques abans presentades.

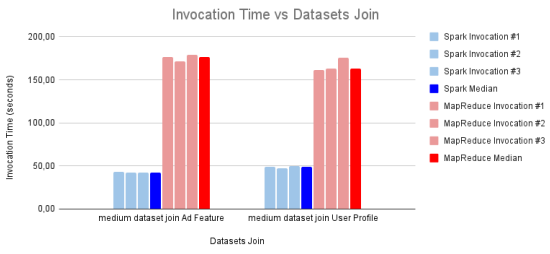


Figura 59: Temps d'invocació vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 59 podem veure que al igual que en els experiments anteriors, el temps d'execució de MapReduce és superior al de Spark. Concretament, MapReduce és de mitjana un 275 % més lent en fer les invocacions que Spark.

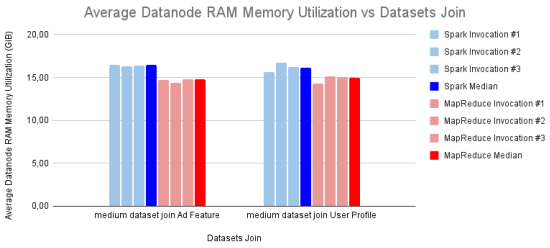


Figura 60: Utilització mitjana de memòria RAM dels nodes executors vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 60 podem observar que Spark utilitza un 10 % més de memòria RAM que MapReduce.

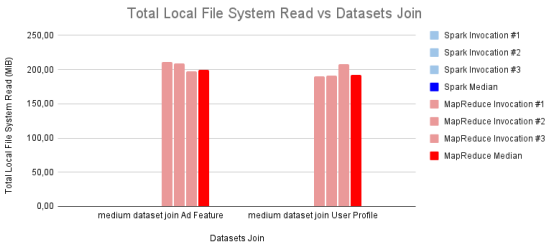


Figura 61: Utilització del disc local a cada node executor per a lectura vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 61 clarament podem observar que MapReduce està llegint del disc local, mentre que Spark no llegeix quasi res. Aquest comportament és l'esperat ja que MapReduce, després d'acabar la fase de Map, ha de fer un Shuffle i executar els Reducers i, per a comunicar aquestes fases entre elles utilitza el disc local del node executor.

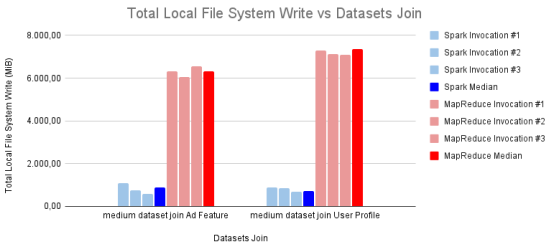


Figura 62: Utilització del disc local a cada node executor per a escriptura vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 62 observem que Spark escriu al disc local. Tal com vam mencionar en anteriors experiments, considerem que aquesta escriptura consisteix en escriure arxius temporals. En canvi, podem observar una escriptura molt alta de MapReduce, on observem que està escrivint fins al voltant de 7 GiB d'informació al disc. En la Figura 62 observem que

només s'estan llegint una mica més de 200 MiB. Aquesta diferència pot estar més lligada als detalls d'implementació del *framework*, al igual que en el cas de Spark, on fem referència a que està escrivint arxius temporals.

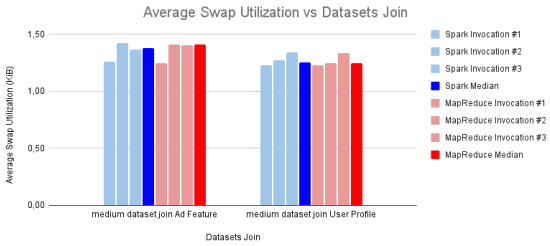


Figura 63: Utilització de la memòria *swap* vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 63 observem que la memòria d'intercanvi, al igual que en els altres experiments està molt igualada entre els dos *frameworks* i ens indueix a pensar que no depèn d'aquests.

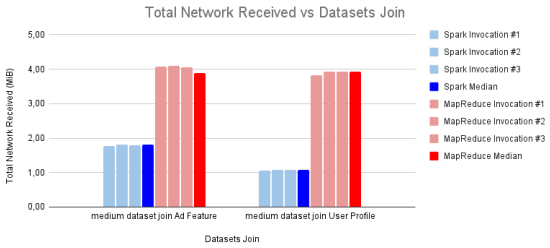


Figura 64: Recepció d'informació a través de la xarxa vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la figura Figura 64 observem un cop més que Spark utilitza més eficientment la xarxa que MapReduce, al estar comprimint des dades abans d'enviar-les amb un algorisme de *highweight* compression. En aquest gràfic, Spark utilitza al voltant d'un 190 % menys la xarxa que MapReduce.

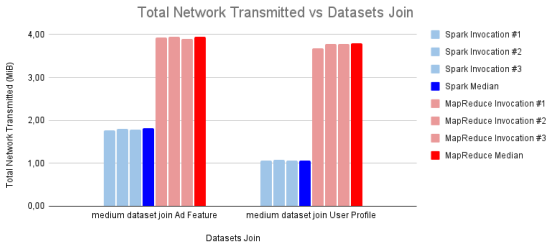


Figura 65: Transmissió d'informació a través de la xarxa vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 65 observem un patró similar a la Figura 64, on MapReduce està transportant més dades a través de la xarxa. Podem observar que tot i que la *Join* és una operació que té una *wide dependency*, és possible que es faci més eficient aquest transport per la compressió.

En la Figura 66 observem que el temps que MapReduce utilitza la CPU en tots els seus

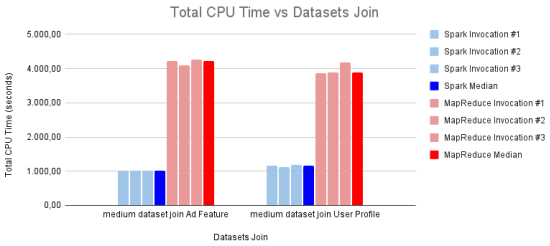


Figura 66: Temps Total de CPU vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

modes és un 272 % més gran que Spark.

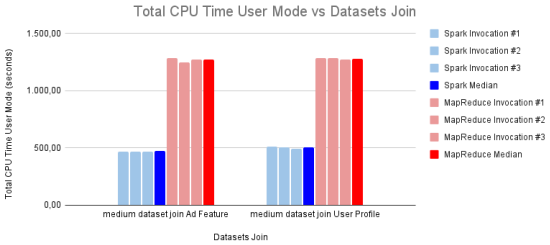


Figura 67: Temps de CPU en el *User Mode* vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 67 podem veure que MapReduce utilitza un 160 % més la CPU en mode usuari que Spark.

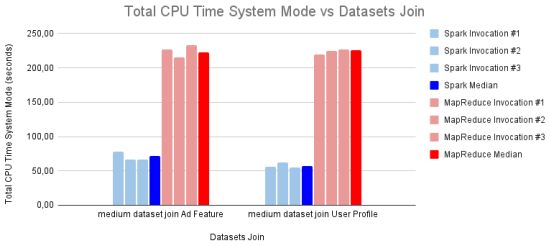


Figura 68: Temps de CPU en el *System Mode* vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 68 veiem com MapReduce està utilitzant un 253 % més la CPU en mode Sistema que Spark.

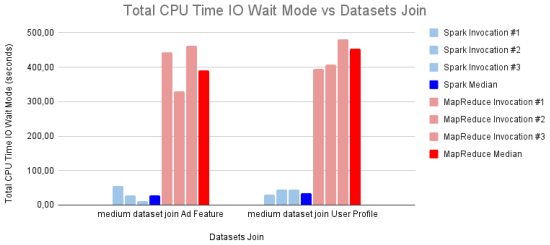


Figura 69: Temps de CPU en el *IO Wait Mode* vs. *Join* dels conjunts de dades
[Font: elaboració pròpia]

En la Figura 69 observem que MapReduce està considerablement utilitzant més la CPU

en el mode *IO Wait*, concretament un 1.240 %, ja que estem escrivint els resultats del Mapper al disc i després aquestes dades es llegeixen per part dels Reducers.

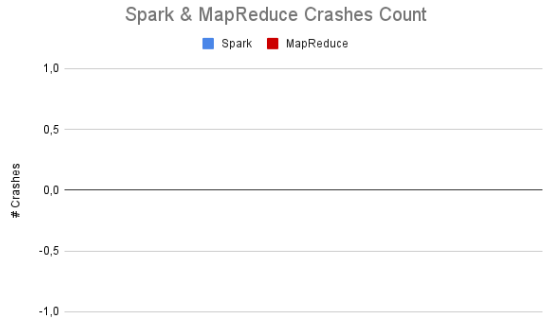


Figura 70: Nombre de fallades a cada *framework*
[Font: elaboració pròpia]

En la Figura 70 es pot veure que no ha sorgit cap error durant l'execució d'aquest experiment.

5.2.5 Operació de *Join* gran

Objectiu

L'objectiu d'aquest experiment és mesurar les mètriques especificades en la secció 5.1 per diferents execucions de la *join* sobre el conjunt de dades mitjà utilitzant diferents columnes.

Cal destacar que per la naturalesa dels *logs*, aquests normalment acostumen a tenir valors repetits a totes les columnes i, a vegades també valors nuls. Per tant, no compleixen amb el requisit de tenir clau primària estrictament relacional. En canvi, tenen columnes amb pocs valors repetits i, algunes columnes no tenen valors nuls. Aleshores interpretarem aquestes columnes com aquelles columnes "clau primària". Per tant, hem de comprendre que la *join* que estem executant tot i tenir una certa assimilatòria amb l'operació de *self join* relacional, no és la mateixa en el sentit estricte relacional, per la naturalesa del conjunt de dades.

Amb aquesta operació analitzarem les columnes de "clau primària" i "clau candidata" perquè la resta tindrien una sortida excessivament gran que ens portaria un temps no raonable per processar. Aquestes dues columnes ens permetran obtenir una sortida molt gran i provar els dos sistemes en un cas amb una sortida molt gran.

L'operació de *join* és una *wide dependency*, és a dir que s'han d'intercanviar necessàriament dades a través de la xarxa i no es pot executar de forma distribuïda sense necessitat d'interaccionar amb els altres nodes. Encara que tal com vam mencionar en l'experiment on s'analitzava l'operació de *join*, el rendiment podria ser millor per part de Spark, encara que quedaria per veure a través dels experiments quin sistema funcionaria millor en aquest cas d'ús.

Configuració

El *pipeline* que executarem per a aquest experiment és el *pipeline* de *join* gran.

Executarem els *pipelines* per al conjunt de dades mitjà i farem la *join* amb el mateix conjunt de dades.

En el conjunt de dades mitjà, hi ha dues columnes que segueixen aquestes especificacions i aquestes són la columna DateTime i User. Aquestes columnes fan *join* de mitjana amb 1.681 i 6.495 files agafant la part entera superior, respectivament i la sortida que generen els resultats de les *joins* agafant aquestes columnes són 1,27 TiB i 4,9 TiB, respectivament. La resta de columnes generaven una sortida més gran de 31 TiB i arribaven a l'ordre de PiB. Per aquesta raó i per a estar executant durant molt temps aquesta operació i conseqüentment, crear una despesa excessiva en l'ús del clúster, vam decidir fer només les columnes analitzades anteriorment, que són DateTime i User.

Per a aquest experiment, mesurarem els valors de totes les mètriques especificades a la secció 5.1.

Resultats

En aquest apartat, mostrarem els resultats per cada una de les mètriques abans presentades.

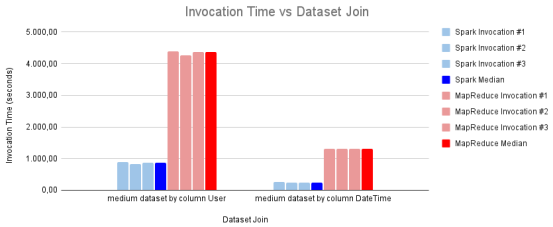


Figura 71: Temps d'invocació vs. *Join* del conjunt de dades
[Font: elaboració pròpia]

En la Figura 71 observem que de mitjana, MapReduce triga un 418 % més temps que Spark en fer les invocacions.

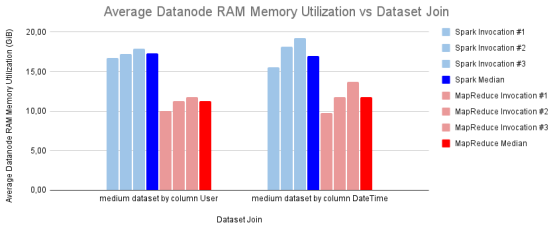


Figura 72: Utilització mitjana de memòria RAM dels nodes executors vs. *Join* del conjunt de dades
[Font: elaboració pròpia]

En la Figura 72 podem veure que Spark utilitza un 49 % més memòria RAM que MapReduce.

En la Figura 73 observem que l'ús que es fa per a lectura per part del Spark és menyspreable en comparació amb MapReduce. Aquest últim, llegeix de l'ordre de 400 MiB del disc local.

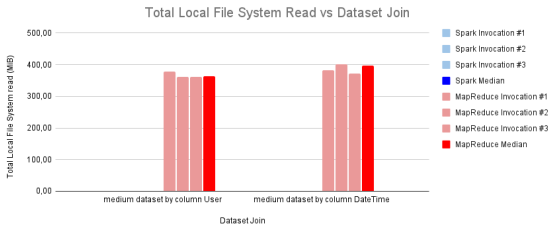


Figura 73: Utilització del disc local a cada node executor per a lectura vs. *Join* del conjunt de dades
[Font: elaboració pròpia]

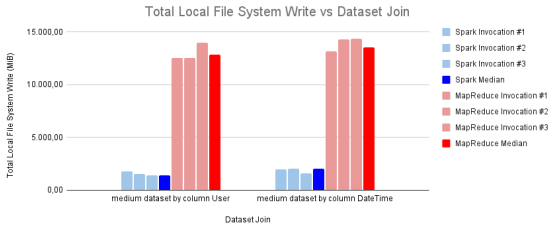


Figura 74: Utilització del disc local a cada node executor per a escriptura vs. *Join* del conjunt de dades
[Font: elaboració pròpia]

En la Figura 74 podem observar un gràfic com en els experiments anteriors, on Spark i MapReduce escriuen al disc local de cada node. En aquest cas, MapReduce escriu un 686 % més que Spark al disc. MapReduce escriu de l'ordre de 13 GiB de dades al disc.

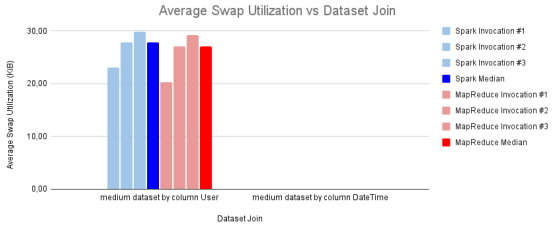


Figura 75: Utilització de la memòria *swap* vs. *Join* del conjunt de dades
[Font: elaboració pròpia]

En la Figura 75 tornem a veure que la memòria d'intercanvi no té cap relació amb el *framework*. Podem observar que la segona *Join*, fins i tot la memòria *swap* utilitzada és 0. Això és perquè vam re-executar en un altre moment aquest experiment, just després de reiniciar el clúster i després d'haver executat tots els experiments.

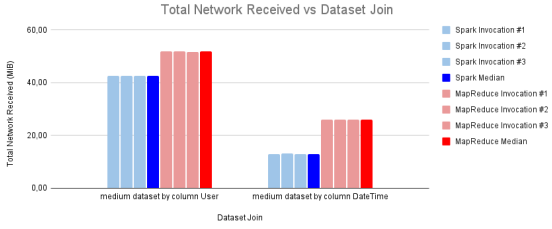


Figura 76: Recepció d'informació a través de la xarxa vs. *Join* del conjunt de dades
[Font: elaboració pròpia]

En la Figura 76 veiem com Spark en la primera *join* utilitza un 22 % menys la xarxa que MapReduce. En la segona *join*, Spark utilitza un 100 % menys xarxa que MapReduce.

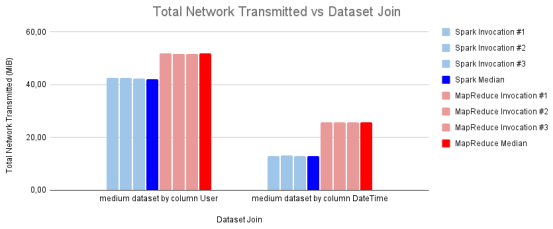


Figura 77: Transmissió d'informació a través de la xarxa vs. *Join* del conjunt de dades [Font: elaboració pròpia]

En la Figura 77 observem el mateix patró que en la Figura 77 i també els valors numèrics dels percentatges també es mantenen aproximadament iguals.

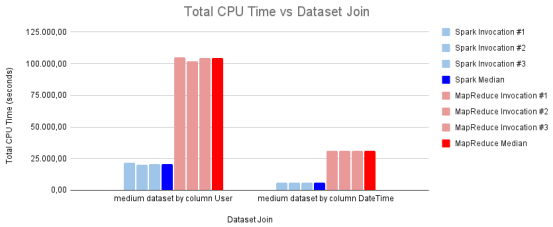


Figura 78: Temps Total de CPU vs. *Join* del conjunt de dades [Font: elaboració pròpia]

En la Figura 78 veiem com MapReduce està utilitzant un 417 % més CPU en tots els modes que Spark.

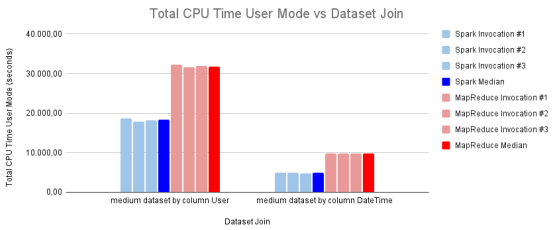


Figura 79: Temps de CPU en el *User Mode* vs. *Join* del conjunt de dades [Font: elaboració pròpia]

En la Figura 79 veiem que MapReduce utilitza un 85 % més temps la CPU en el mode Usuari.

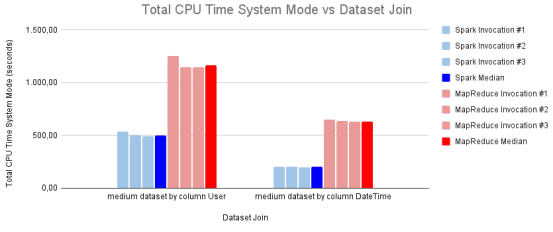


Figura 80: Temps de CPU en el *System Mode* vs. *Join* del conjunt de dades [Font: elaboració pròpia]

En la Figura 80 veiem que MapReduce utilitza un 169 % més temps la CPU en el mode Sistema.

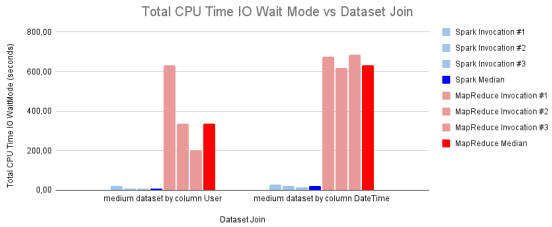


Figura 81: Temps de CPU en el *IO Wait Mode* vs. *Join* del conjunt de dades
[Font: elaboració pròpia]

En la Figura 81 veiem com MapReduce està utilitzant un 3.376 % més la CPU en mode *IO Wait* que Spark. Aquesta gràfica seguiria la mateixa justificació que la Figura 69.

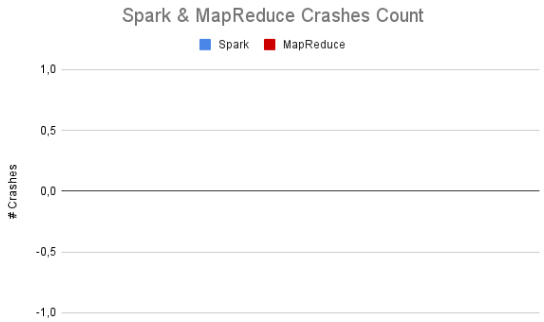


Figura 82: Nombre de fallades a cada *framework*
[Font: elaboració pròpia]

En la Figura 82 es pot veure que no ha sorgit cap error durant l'execució d'aquest experiment.

5.2.6 Exploració del límit mínim de funcionament

Objectiu

L'objectiu d'aquest experiment és explorar el límit mínim que té cada sistema de memòria RAM per a poder començar a executar les operacions del *pipeline*.

Aquest límit ens hauria de demostrar quin dels dos sistemes funcionaria millor en un entorn amb memòria restringida i, per això, limitarem la memòria dels executors. Aquest experiment ens aportaria informació de quin sistema s'adaptaria millor a un clúster que fa un ús més reduït de la memòria RAM. Recordem que les instàncies amb una quantitat de memòria més elevada impliquen un cost econòmic també més elevat, sobretot si estem utilitzant els serveis del núvol, com per exemple Amazon Web Services o Google Cloud Platform.

Configuració

El *pipeline* que executarem per a aquest experiment és el *pipeline* de *join* gran.

Per a aquest experiment definirem una mètrica booleana que ens indicarà si el *Job* ha acabat o no. Explorarem el valor d'aquesta mètrica de forma absoluta comparant-la amb la quantitat de memòria RAM configurada per a ser utilitzada pels *frameworks* i de forma relativa a la grandària de la sortida.

Per a aquest experiment, executarem el *pipeline* sobre el conjunt de dades mitjà per la columna User, on una columna fa *join* amb 6.495 files i el resultat de la sortida té una grandària aproximada de 4,9 TiB.

Per a fer els càlculs, hem creat un excel, en GitHub anomenat "Cluster's Memory Configurations Calculation", on variant el valor de la memòria RAM total, podem obtenir els diferents valors de les configuracions, al igual que vam fer en les Taules 12 a 14 i en la configuració de Hadoop.

Resultat

En aquest apartat, mostrarem els resultats de l'experiment.

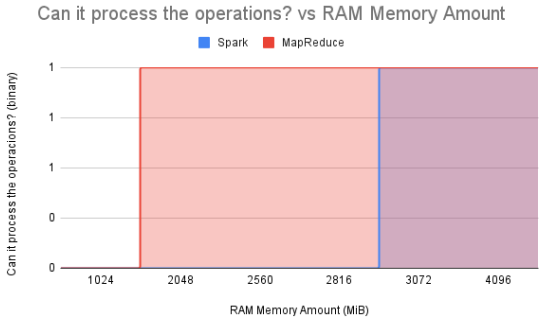


Figura 83: Exploració del llindar de memòria RAM mínim requerit per a que funcioni cada *framework*
[Font: elaboració pròpia]

En la Figura 83 observem que el llindar mínim del *framework* MapReduce està entre 1 GiB de memòria RAM i 2 GiB de memòria RAM. El llindar per a Spark es troba entre els 2816 MiB i 3072 MiB.

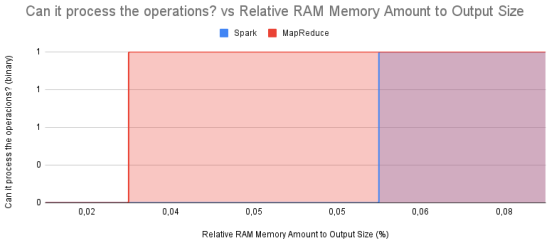


Figura 84: Exploració del llindar de memòria RAM mínim requerit per a que funcioni cada *framework* relatiu a la grandària de la sortida
[Font: elaboració pròpia]

Observem en la Figura 84 que tot i que MapReduce té un llindar de memòria menor que Spark, si mirem el valor relatiu a la sortida de l'operació, veiem que els percentatges són molt similars. Cal recalcar que no seria del tot correcte tampoc comparar la quantitat de

memòria RAM amb la sortida, encara que entre les diferents operacions generen resultats intermedis que són més grans que l'entrada. Aleshores, les sortides computaran certs resultats intermedis que faran que aquesta relació sigui més correcta comparar-la amb la sortida que no pas l'entrada de l'operació.

5.2.7 Exploració de la correlació entre memòria *swap* utilitzada i el temps d'invocació

Objectiu

L'objectiu d'aquest experiment és veure si existeix alguna correlació entre la memòria *swap* o d'intercanvi utilitzada i el temps necessari per a realitzar l'invocació de les operacions del *pipeline*.

Cal recalcar que aquest experiment no requereix l'execució de cap operació, sinó que es recolliran les dades d'utilització de a memòria *swap* dels experiments anteriors, com també el temps d'invocació i es mostraran de forma conjunta per a analitzar la correlació.

Aquest experiment és important perquè la utilització de la memòria *swap* implica l'escriptura de dades al disc, ja que aquestes no caben a la memòria RAM, i pot ser especialment interessant en els casos on s'està analitzant un conjunt de dades gran i aquest no cap a la memòria. Recordem que Spark treballa amb els conjunt de dades repartits a memòria, mentre que MapReduce els llegeix directament del disc. Aquest experiment analitzaria més bé si Spark es veuria afectat pel fet que l'ús de la memòria RAM és elevat i, les dades podrien no caber a memòria i veuríem quant de bé es gestionarien els recursos en comparació amb el sistema MapReduce. Aleshores, l'objectiu a mostrar és que el MapReduce és molt més eficient gestionant la memòria que es diu an alguns cercles.

Configuració

Per a aquest experiment, reaprofitarem els resultats de tots els experiments anteriors.

Definirem les següents mètriques per a veure com es comporten els dos sistemes:

- **Exploració de la correlació entre l'utilització de memòria *swap*.** Aquesta mètrica ens donarà informació de si quanta més memòria *swap* s'estigui utilitzant, el *framework* Spark s'assemblarà més a MapReduce. És a dir, quanta més memòria d'intercanvi s'estigui utilitzant, més s'assemblaran els temps d'invocació dels dos *frameworks* per a l'execució de les operacions.

- **Exploració de la correlació entre l'utilització de memòria *swap*.** Aquesta mètrica ens servirà com a base de comparació amb les dades obtingudes de la mètrica anterior.

- **Exploració de la correlació entre l'utilització de memòria *swap*.** Aquesta mètrica ens reflectirà de forma conjunta als dos *frameworks* si la quantitat de memòria d'intercanvi utilitzada realment té algun impacte en el temps d'invocació de les operacions.

Per a calcular la correlació, hem agafat de tots els experiments anteriorment explicats els temps d'invocació (Invocation time in seconds) i la memòria d'intercanvi utilitzada (Average swap utilization in MiB) per a cada *framework* i les hem posat en dues taules, una per a cada *framework*. Vam ordenar la taula per la columna Average swap utilization in MiB i hem creat una simple gràfica de l'eix Y (Invocation time in seconds) vs. eix X (Average swap utilization in MiB) per a veure la correlació per a cada taula. Posteriorment, hem fet una altra taula, fusionant les dues taules anteriors en una, és a dir posant una taula al final de l'altra, hem ordenat un altre cop per la columna Average swap utilization in MiB i ham creat una gràfica de la mateixa forma que hem explicat anteriorment.

Resultats

En aquest apartat, mostrarem els resultats per cada una de les mètriques abans presentades.

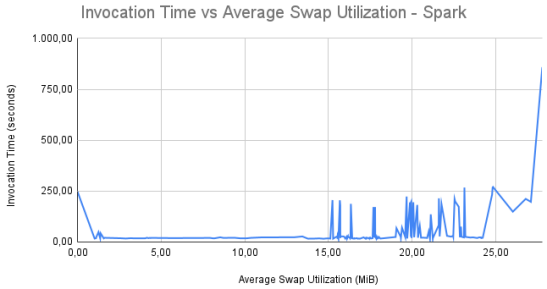


Figura 85: Exploració de la correlació entre l'utilització de memòria *swap* vs. temps d'invocació per a Spark
[Font: elaboració pròpia]

En la Figura 85 observem que hi ha un pic al final en la utilització de la memòria *swap*, on el temps d'invocació és més alt. Tot i així, no observem cap correlació entre el temps d'invocació i l'utilització de la memòria *swap*.

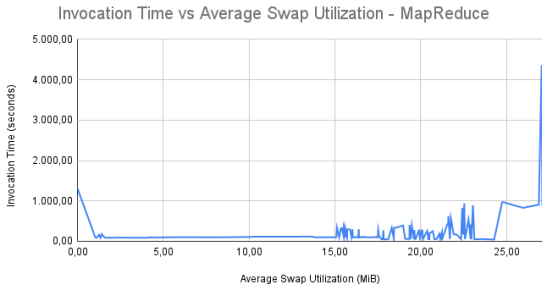


Figura 86: Exploració de la correlació entre l'utilització de memòria *swap* vs. temps d'invocació per a MapReduce
[Font: elaboració pròpia]

En la Figura 86 observem el mateix que en la Figura 85, on no podem concloure que hi existeixi una correlació.

Finalment, en la Figura 87 veiem de forma conjunta als dos *frameworks* que no hi ha cap correlació amb el temps d'invocació.

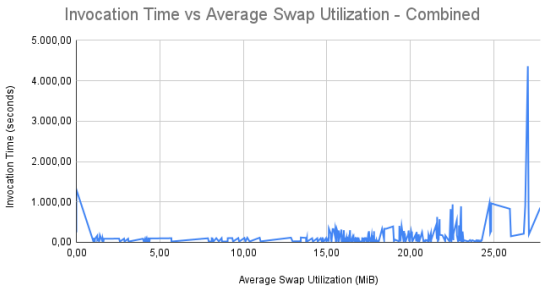


Figura 87: Exploració de la correlació entre l'utilització de memòria *swap* vs. temps d'invocació dels dos *frameworks*
[Font: elaboració pròpia]

5.3 Reproductibilitat i Transparència

Un aspecte fonamental d'aquest tipus de treball és la transparència i la capacitat de reproduir tots els passos per a poder validar que hem arribat a les mateixes conclusions que altres *stakeholders*. Per a això, hem posat tot el codi tant del *Datasets Profiler* com del *Testbed* al GitHub. Els links dels repositoris els trobem en [82] i [83]. Es pot trobar més informació en el capítol D dels apèndixs.

Tota aquesta informació serveix per a fer transparent el projecte i, si algun dels *stakeholders* vol executar els mateixos experiments en un altre clúster o simplement només vol comprovar que les dades són correctes, pot executar el *profiling* dels conjunts de dades. També pot generar les entrades del *Testbed* i executar tots els experiments. Si s'utilitza una mateixa configuració del clúster que nosaltres, podria arribar a les mateixes conclusions.

5.4 Conclusió

En conclusió, observem que Spark millora en molts aspectes a MapReduce, utilitzant la configuració explicada en la secció 5.1. Podem concloure que en quant a temps d'invocació, en tots els experiments i conjunts de dades, Spark va ser més ràpid que MapReduce, per tant, si l'aplicació que estem desenvolupant és molt important que s'executi en un temps molt ràpid, aleshores Spark és l'opció a elegir.

En quant a ús de la memòria RAM, per a les operacions de Selecció i Projecció i la combinació de les dues operacions, no vam veure una gran diferència per part dels dos sistemes per als conjunts de dades petit i mitjà, encara que sí que vam veure la diferència en el conjunt de dades gran, on Spark utilitzava més memòria RAM que MapReduce. En quant a l'operació de *join*, observem que Spark utilitza només un 10 % més memòria RAM. Per a l'operació de la *join* gran, aquest percentatge d'utilització augmenta a un 49 %. Aquest percentatge no expressa ordres de magnitud, encara que hem de destacar el fet que la memòria RAM és un recurs que els proveïdors de serveis al núvol cobren per quantitat, és a dir a més quantitat, més car serà el clúster. Per tant, si volem reduir costos i el temps d'execució que incrementa MapReduce es podria veure compensat econòmicament pel cost de la memòria RAM i l'aplicació no requereix que el sistema tingui un temps d'execució molt ràpid, aleshores podria ser una alternativa a considerar. Tot i que per ser realistes, en els últims anys, el

cost de la memòria RAM es va abaratir enormement i, aquesta és una de les raons per la qual es va crear Spark, perquè la memòria RAM va passar de ser un recurs escàs a un recurs suficientment barat i abundant.

En quant a l'ús del disc local, MapReduce l'utilitza per a intercanviar les dades entre les diferents fases de Map, Shuffle i Reduce. Tot i així, les operacions que no requereixin de Reducers, no escriuran o llegiran en aquest espai. Spark hem vist que no utilitza aquest espai per a la lectura, a menys que el conjunt de dades no càpigui a memòria, que haurà de començar a fer un ús més intensiu del disc.

La memòria d'intercanvi, vam observar en tots els experiments que no tenia massa correlació amb el fet que estàvem utilitzant un sistema o l'altre. Això pot ser conseqüència del fet que tampoc hem pogut saturar el clúster amb un conjunt de dades tant gran que no hi capigués. Cal destacar que en total, els nodes executors disposaven de l'ordre de 70 GiB de memòria RAM com a màxim. Per a restringir l'ús de la memòria RAM, vam modificar els paràmetres dels *frameworks*, per a trobar el llindar mínim de funcionament, però en la resta dels experiments, no es va necessitar utilitzar memòria d'intercanvi.

Vam veure també a través dels experiments que Spark fa un ús de la xarxa que és ordres de magnitud més eficient que en el cas de MapReduce. Això és per l'algorisme de compressió que Spark porta activat per defecte, el LZ4, que permet reduir la quantitat de dades a enviar per la xarxa. Això permet a la seva vegada reduir la possibilitat de saturar la xarxa i, per tant, permetre elevar el llindar de quan es satura aquesta. En alguns casos específics, vam observar que MapReduce era molt lleugerament millor per als conjunts de dades grans, tot i que no consideraríem un 1 % de millora com un factor decisiu per a seleccionar MapReduce en front de Spark.

MapReduce també fa un ús més intensiu del processador en tots els modes, en el mode Usuari i en el mode Sistema. En el mode *IO Write* vam observar que si s'han d'intercanviar dades entre els Mappers i Reducers, aleshores el temps de CPU en el mode *IO Write* creix fins a ordres de magnitud més que en el cas de Spark, com va passa en l'experiment de la *join* gran.

Respecte a la tolerància a fallades, Spark i MapReduce no van fallar per culpa de falta d'espai a la memòria RAM o per errors interns del *framework*. Tot i que en el cas de Spark, els usuaris, utilitzant Spark SQL hereten un dels problemes ja existents amb el SQL relacional, que era la Injecció de SQL. Aquest problema MapReduce no el té ja que no hem de definir una consulta, sinó que hem de crear nosaltres mateixos el codi per a fer l'operació. Vam detectar també dos errors molt aïllats per a MapReduce, encara que l'error de Spark mencionat amb anterioritat es va repetir al llarg de molts experiments.

Respecte al llindar mínim de funcionament, vam observar que de forma relativa al conjunt de dades de sortida, el percentatge mínim de funcionament era molt baix per als dos sistemes.

Hem de destacar que Spark és molt més fàcil de programar i de crear fluxos de processament de dades que no pas MapReduce. MapReduce ens obliga a definir codi una i altra vegada per a gestionar els *jobs*. Spark ho manega tot per si mateix, sense que l'usuari es doni compte de què està fent el *framework*, fent el codi més mantenible que MapReduce.

Finalment, tenir en compte que no hem analitzat conjunts de dades molt grans, de l'ordre de TiB o PiB per falta de temps i recursos. Per a tenir una conclusió més completa que la que ja tenim, hauríem d'haver utilitzat conjunts de dades més grans que aquests, tot i així, cal recalcar que tot el codi del *Testbed* i del *Datasets Profiler* és públic i els resultats de la nostra anàlisi també són públics en els repositoris mencionats anteriorment. Per tant, els *stakeholders* poden comprovar la correctesa de les nostres conclusions reexecutant tots els experiments un clúster amb les mateixes característiques o també poden fer més experiments per a provar altres casos d'ús.

Capítol 6

Planificació temporal

6.1 Descripció de les tasques

6.1.1 Introducció

Es va començar a treballar activament en el TFG el dia 14 de gener de 2021. La data prevista de finalització és el 28 de juny del mateix any, amb la lectura del TFG; tot i que la data exacta de la presentació no està concretada, agafarem el primer dia del període de lectura del TFG. Es treballarà activament durant 6 hores diàries de dilluns a divendres en el treball; així que des de l'inici del treball fins al final, hi ha 117 dies i, per tant, 702 hores de treball en les que s'hauran de distribuir les tasques perquè hi hagi un bon rendiment en el desenvolupament de la tesi.

6.1.2 Tasques

En aquest apartat mostrarem les diverses tasques que es realitzaran al llarg del projecte. A continuació, veurem els grups de tasques, les tasques individuals i una descripció d'aquestes:

[GP] Gestió del projecte

- **[GP1] Definir l'abast.** Consisteix a donar una contextualització del projecte, una justificació del perquè es fa el projecte, definir quin serà l'abast del projecte i explicar la metodologia a seguir.
- **[GP2] Definir la planificació temporal.** Es farà una descripció de les tasques a seguir i una distribució d'aquestes en el temps.
- **[GP3] Fer el pressupost.** Es farà una estimació dels costos del projecte.
- **[GP4] Fer l'informe de sostenibilitat.** S'analitzarà la sostenibilitat del projecte que s'està desenvolupant.
- **[GP5] Fer la documentació per a la fita de seguiment.** Es revaloraran els aspectes definits a la fita inicial i es consideraran els canvis en els diversos aspectes que hi hagi hagut.
- **[GP6] Fer la documentació per a la fita final.** Consisteix a fer informes durant cada setmana per a poder condensar en petits documents la informació que es presen-

tarà a la reunió. Durant la fase final del TFG, s'agafaran aquests informes i s'inclouran en un únic document per a formar la memòria del TFG.

- **[GP7] Preparar la presentació del projecte.** Consisteix a realitzar i assajar la presentació dels aspectes més importants del projecte abans de fer la lectura del TFG.
- **[GP8] Reunions setmanals.** És una tasca setmanal que dura aproximadament dues hores i és una reunió entre el director del TFG i jo mateix per a presentar les tasques fetes durant la setmana, treure conclusions i prendre decisions per a les següents etapes segons els resultats obtinguts.

[FM] Familiarització amb els *frameworks*

- **[FM1] Familiarització amb els *frameworks*.** Es miraran materials dels cursos de CBDE (Conceptes per a Bases de Dades Especialitzades) del GEI (Grau en Enginyeria Informàtica) i BDM (*Big Data Management*) del Màster en *Data Science* per a repassar el funcionament de MapReduce i aprendre com funciona Spark, i fer exercicis per a familiaritzar-se amb els dos *frameworks*.

[DP] Creació del *datasets profiler*

- **[DP1] Definició de l'arquitectura.** Consisteix a definir l'arquitectura del programa que fa un perfil estadístic dels conjunts dades de tal forma que compleixi amb els requisits funcionals i no funcionals.
- **[DP2] Implementació del programa.** S'implementarà el programa per a poder analitzar el contingut dels *datasets* i poder tenir una millor visió de quins escollir.

[SD] Seleccionar els *datasets* a utilitzar

- **[SD1] Recerca de *datasets*.** Es buscaran *datasets* a internet de diverses grandàries.
- **[SD2] Definir les entrades.** Definir les entrades per al *datasets profiler*.
- **[SD3] Execució del *datasets profiler*.** Consisteix a fer l'execució del programa amb les entrades pròpiament dites.
- **[SD4] Classificació dels *datasets*.** Consisteix a fer una classificació dels *datasets* anteriorment analitzats en petits, mitjans, grans i molt grans, segons els perfils estadístics obtinguts.

[TB] Creació del *Testbed*

- **[TB1] Definició de l'arquitectura.** Consisteix a definir l'arquitectura del *Testbed* de tal forma que compleixi amb els requisits funcionals i no funcionals.
- **[TB2] Adaptació del *datasets profiler*.** S'ha d'adaptar aquest programa perquè pugui treballar juntament amb el *Testbed*. La sortida en forma d'estadístiques del programa que fa un perfil estadístic del *dataset* i el conjunt de dades transformat, s'utilitzaran com a entrada per al *Testbed*.
- **[TB3] Implementació del programa.** S'implementarà el programa que ens permetrà executar els experiments.

[IC] Investigació del funcionament del clúster

- **[IC1] Investigar com utilitzar el *datasets profiler* i el *Testbed* en el clúster.** Investigar com treballar amb les màquines virtuals i dissenyar una forma per a moure el *datasets profiler* i el *Testbed* al clúster.
- **[IC2] Investigar com utilitzar la *Memory Storage Support* en HDFS.** Consisteix a veure com activar l'opció *Memory Storage Support* en HDFS en el clúster seguint la documentació.

[SX] Realització dels experiments

- **[SX1] Disseny dels experiments.** Consisteix a dissenyar els experiments que es portaran a terme amb operacions de l'àlgebra relacional.
- **[SX2] Execució dels experiments al clúster i recollida dels resultats.** Consisteix a executar els conjunts d'experiments dissenyats en la tasca anterior.
- **[SX3] Consolidació dels resultats i extracció de conclusions.** Consisteix a revisar els resultats obtinguts, validar-los i extreure conclusions sobre aquests.

[MX] Complementació dels experiments

- **[MX1] Disseny dels experiments.** Consisteix a dissenyar els experiments complementaris que es portaran a terme. Aquests inclouen experiments sobre operacions iteratives i/o execució dels experiments amb la característica de *Memory Storage Support* activada.
- **[MX2] Preparació del clúster/*Testbed* per a executar els experiments.** Consisteix a aplicar sobre el clúster la configuració necessària per a executar l'experiment, o bé afegir les funcionalitats necessàries en el *Testbed*.
- **[MX3] Execució dels experiments al clúster i recollida dels resultats.** Consisteix a executar els conjunts d'experiments dissenyats en la tasca **MX1**.
- **[MX4] Consolidació dels resultats i extracció de conclusions.** Consisteix a revisar els resultats obtinguts, validar-los i extreure conclusions sobre aquests.

Cal destacar que tot i que és possible que trobem programes que siguin molt similars o que ens puguin ajudar per a la implementació del *datasets profiler* i del *Testbed*, nosaltres implementarem els dos programes des de zero, per a poder tenir un major control sobre els paràmetres a l'hora de fer els experiments.

6.1.3 Recursos

Recursos humans

Seré jo mateix, com a investigador [**RHI**], encarregat de dur a terme el projecte, i estaré supervisat pel director del projecte [**RHD**]. Addicionalment, hi ha el tutor de GEP [**RHT**], que corregirà una part de la gestió del projecte.

Recursos materials

Per a la realització satisfactòria d'aquest projecte, es necessitaran els següents materials:

- **[RMM] Materials i exercicis de les assignatures de CBDE i BDM sobre els *frameworks*.** Aquests seran imprescindibles per a adquirir ràpidament un enteniment del funcionament dels dos *frameworks*.
- **[RMX] TeXstudio.** És un editor de LaTeX que utilitzarem per a produir la documentació.
- **[RMC] Clúster d'ordinadors.** Serà imprescindible per a poder realitzar els diferents experiments. Utilitzarem un màxim de 5 instàncies en tot moment. Cada instància té 8 nuclis, 32 GiB de memòria RAM i una capacitat d'1 TB de disc.
- **[RMP] Portàtil.** Serà necessari per a produir la diferent documentació necessària, pel desenvolupament dels diferents programes i també per la comunicació amb el director del TFG.
- **[RMO] Ordinador per a executar el *datasets profiler*.** Els *datasets* tenen una grandària considerable, per tant, s'utilitzarà un ordinador que té 32 GiB de RAM i un processador Intel® Core™ i7-7820X de 8 nuclis i 16 fils.
- **[RMH] Git i GitHub.** Aquestes eines s'utilitzaran per al control de versions del codi.
- **[RMY] IDE de Python.** Utilitzarem PyCharm per al desenvolupament del *datasets profiler*.
- **[RMJ] IDE de Java.** Utilitzarem IntelliJ per al desenvolupament del *Testbed*.

6.2 Estimacions i Gantt

A la Taula 15, s'han resumit totes les tasques. Cal destacar que la fase de Complementació dels experiments (**MX**) està planificada per a dues execucions. La raó per la qual la tasca **MX3** és de 50 hores és perquè podrem reaprofitar una part del processament que s'haurà fet en la tasca **SX2**.

La Figura 88 representa el diagrama de Gantt; hem acolorit les tasques segons la iteració en la qual començaven, exceptuant les tasques corresponents a la Gestió del Projecte. Cada franja de color és una setmana i representa una iteració. Cal destacar que no hem inclòs de forma explícita les reunions amb el director del TFG encara que es realitzaran sempre al final de cada iteració, és a dir, es farà al final de cada franja de color. Els grups de tasques es representen amb una franja negra i la seva durada és la durada de totes les subtasques sumades; serveix per a veure la durada de cada una de les fases del projecte.

6.3 Gestió del risc: plans alternatius i obstacles

És possible que durant la realització del projecte, hi apareguin alguns riscos que afectin negativament el desenvolupament d'aquest. En aquesta secció, presentarem els riscos potencials, com afecten el projecte i com els podrem mitigar.

- **Data d'entrega fixada.** El fet que la data per a acabar el TFG té un termini fix i, per la seva naturalesa d'investigació, fan que sigui probable que hi hagi algunes desviacions respecte a la planificació.
  - **Impacte:** Alt.

  - **Solució proposada:** Si la desviació és petita, augmentarem el nombre d'hores dedicades diàriament, en cas contrari, prescindirem de la fase Complementació dels experiments i això ens estalviarà al voltant de 100 hores de feina que no són fonamentals per a l'entrega del TFG.
- ***Bugs.*** És molt probable que el codi que es realitzi no es comporti segons les intencions del programador [84]. Aquest és un problema inherent del *software*, per la qual cosa, l'haurem de tenir en compte. Aquest risc, podria dificultar la realització del *software* i, conseqüentment, endarrerir l'entrega del TFG.
  - **Impacte:** Mitjà.
  - **Solució proposada:** Les mesures que prendrem són que si és necessari, s'utilitzarà més temps per a poder fer els tests necessaris d'integritat i poder trobar els *bugs*. Per tant, s'aplicaran les mateixes mitigacions que en el primer risc.
- **Corrupció de les dades.** La font d'aquest risc és una fallada en el portàtil on es treballa. Això comportaria la pèrdua del codi i documentació redactats, i ens pot fer no arribar a temps per a la data d'entrega del TFG.
  - **Impacte:** Baix.
  - **Solució proposada:** Una mitigació d'aquest risc és utilitzar Google Drive per a emmagatzemar una còpia de seguretat dels arxius sobre els quals es treballen; d'aquest forma, el risc pràcticament desapareix. Tot i així, si aquest risc esdevé realitat, es necessitarà més temps per a poder reescriure les parts perdudes. Per tant, s'hauran de treballar més de 6 hores diàries i, també els caps de setmana per a reescriure els documents perduts.
- **Complexitat dels *frameworks*.** La complexitat associada a cada un dels *frameworks* de processament de dades que estarem comparant és el generador d'aquest risc.
  - **Impacte:** Baix.
  - **Solució proposada:** Si fos necessari, es dedicaria més temps per a fer exercicis i repassar el material, i es retallarien el nombre d'experiments complementaris que es realitzaran.
- **Capacitat computacional.** El processament de dades molt grans no té cap sentit si no disposem d'un clúster d'ordinadors que ens permeti fer el processament de dades en grans volums. Per tant, el generador d'aquest risc és la possibilitat de no disposar de cap clúster per a realitzar els experiments.
  - **Impacte:** Mot baix.
  - **Solució proposada:** Aquest risc el mitigarem utilitzant el clúster que està disponible per a la UPC en el grup de recerca per a fer Treballs de Final de Grau. Tot i així, si aquest clúster no està disponible, utilitzarem en canvi, el mateix ordinador utilitzat per a executar el *datasets profiler*.

6.4 Desviacions de la planificació inicial

Encara que inicialment l'únic objectiu del projecte era fer una comparativa per a veure en quins casos MapReduce és millor que Spark, quan vam començar el treball, la definició que teníem de *Testbed* era d'un conjunt de scripts per a poder fer les execucions dels experiments. Això implicaria molta repetibilitat dels codis i, una major propensió als errors humans. Per

tant, això es traduïa en una menor rigorositat en el nostre estudi. Vam decidir que aportaria més valor crear un sistema unificat, que estigués dissenyat *ad hoc* per a fer l'execució dels experiments d'una forma rigorosa i amb una mínima intervenció d'un operari. Aleshores, vam decidir dedicar més temps a la tasca d'implementació del *Testbed*.

Conseqüentment, això va afectar a la planificació ja que va ocupar molt més temps de desenvolupament del previst, concretament vam estar implementant el *Testbed* al voltant de 400 hores, treballant una mitjana de 8 hores diàries, durant 7 dies a la setmana. Vam decidir permetre una desviació de dues setmanes en la nostra planificació i la cancel·lació de diverses tasques no imprescindibles, a causa d'aquest increment d'hores, es va necessitar també incrementar el nombre d'hores que es dedicava diàriament al desenvolupament del projecte.

També vam prendre la decisió de fer un codi més nét i senzill perquè sigui fàcil utilitzar per a altres investigadors que vulguin reaprofitar el nostre projecte, perquè reduïa la quantitat de *bugs* del nostre codi i, per tant, ens permetia una millor rigorositat dels experiments. Una clara avantatge addicional és que a la planificació inicial, vam suposar que l'experimentació comportaria 50 hores de treball per part de l'investigador, encara que no és l'investigador el que estarà executant manualment els experiments, sinó que els experiments s'organitzaran en *batches* que comportaran desenes d'hores de computació i no requeriran cap intervenció. Això va permetre desenvolupar gran part de la documentació de la memòria final, mentre s'estaven executant els experiments.

Les tasques cancel·lades són totes les del grup de tasques anomenat **[MX] Complementació dels experiments** i la tasca **[IC2] Investigar com utilitzar la Memory Storage Support en HDFS**. Respecte la resta de tasques, no hi ha hagut cap desviació a destacar.

Cal recalcar també que tot i que al començament vam pensar que els resultats de l'investigació es podrien a utilitzar per a millorar el contingut dels cursos de bases de dades específiques del grau i del màster, finalment ens hem replantejat publicar els resultats a causa de la seva rigorositat, perquè tingui una major visibilitat.

ID	Tasca	Hores	Dependències	Recursos
GP	**Gestió del Projecte**	**224h**	**-**	**RHI, RHD, RHT, RMX, RMP**
GP1	Definir l'abast	25h	-	RHI, RHD, RHT, RMX, RMP
GP2	Definir la planificació temporal	10h	GP1	RHI, RHD, RHT, RMX, RMP
GP3	Fer el pressupost	10h	GP2	RHI, RHD, RHT, RMX, RMP
GP4	Fer l'informe de sostenibilitat	5h	GP2	RHI, RHD, RMX, RMP
GP5	Fer la documentació per a la fita de seguiment	30h	GP4	RHI, RHD, RMX, RMP
GP6	Fer la documentació per a la fita final	60h	GP5	RHI, RHD, RMX, RMP
GP7	Preparar la presentació del projecte	40h	GP6	RHI, RHD, RMP
GP8	Reunions setmanals	44h	-	RHI, RHD, RMX, RMP
FM	**Familiarització amb els *frameworks***	**40h**	**-**	**RHI, RMM, RMP**
FM1	Familiarització amb els *frameworks*	40h	-	RHI, RMM, RMP
DP	**Creació del *datasets profiler***	**60h**	**-**	**RHI, RMP, RMH, RMY**
DP1	Definició de l'arquitectura	10h	FM	RHI, RMP
DP2	Implementació del programa	50h	DP1	RHI, RMP, RMH, RMY
SD	**Seleccionar els *datasets* a utilitzar**	**63h**	**-**	**RHI, RMX, RMP, RMO**
SD1	Recerca de *datasets*	10h	-	RHI, RMP
SD2	Definir les entrades	3h	SD1, DP2	RHI, RMP
SD3	Execució del *datasets profiler*	40h	SD2	RHI, RMP, RMO
SD4	Classificació dels *datasets*	10h	SD3	RHI, RMX, RMP
TB	**Creació del *Testbed***	**75h**	**-**	**RHI, RMP, RMH, RMY, RMJ**
TB1	Definició de l'arquitectura	10h	DP2	RHI, RMP
TB2	Adaptació del *datasets profiler*	15h	TB1	RHI, RMP, RMH, RMY
TB3	Implementació del programa	50h	TB2	RHI, RMP, RMH, RMJ
IC	**Investigació del funcionament del clúster**	**60h**	**-**	**RHI, RMC, RMP, RMH**
IC1	Investigar com utilitzar el *datasets profiler* i el *Testbed* en el clúster	30h	TB3	RHI, RMC, RMP, RMH
IC2	Investigar com utilitzar la *Memory Storage Support* en HDFS	30h	-	RHI, RMC, RMP, RMH
SX	**Realització dels experiments**	**80h**	**-**	**RHI, RMX, RMC, RMP, RMH**
SX1	Disseny dels experiments	10h	IC1, SD4	RHI, RMP
SX2	Execució dels experiments al clúster i recollida dels resultats	50h	SX1	RHI, RMC, RMP, RMH
SX3	Consolidació dels resultats i extracció de conclusions	20h	SX2	RHI, RMX, RMC, RMP, RMH
MX	**Complementació dels experiments**	**100h**	**-**	**RHI, RMX, RMC, RMP, RMH**
MX1	Disseny dels experiments	10h	SX3	RHI, RMP
MX2	Preparació del clúster/*Testbed* per a executar els experiments	20h	MX1, IC2	RHI, RMC, RMP, RMH, RMJ
MX3	Execució dels experiments al clúster i recollida dels resultats	50h	MX2	RHI, RMC, RMP, RMH
MX4	Consolidació dels resultats i extracció de conclusions	20h	MX3	RHI, RMX, RMC, RMP, RMH
	Total	**702h**		

Taula 15: Resum de les tasques
[Font: elaboració pròpia]

Figura 88: Diagrama de Gantt excloent les reunions setmanals amb el director del TFG [Font: elaboració pròpia]

Capítol 7

Gestió econòmica

En aquest capítol comentarem les depeses del nostre projecte i les classificarem segons despeses en recursos humans, en *hardware*, en *software* i en despeses generals. Farem una estimació dels costs i, finalment exposarem les eines que utilitzarem per a controlar el pressupost.

7.1 Identificació dels costos

7.1.1 Cost del personal

Per a analitzar correctament el cost del personal, primer haurem de definir els rols necessaris, el cost per hora i quines activitats durà a terme cada rol. El cost del personal total s'obté de fer la suma del cost del personal per a cada activitat. El cost del personal per a una activitat s'obté de fer la suma per a tots els rols involucrats en l'activitat de la multiplicació del sou d'un rol per hora per les hores treballades per a dur a terme l'activitat. Hi ha exactament 6 rols que estaran involucrats en l'execució d'aquest projecte:

- **Cap de Projecte (CP)**. Executa les tasques de gestió del projecte.
- **Arquitecte de *Software* Júnior (AS)**. Desenvolupa l'arquitectura del programa que fa un *profiling* dels *datasets* i del *Testbed*.
- **Programador Júnior (P)**. Implementa el codi per al programa que fa un *profiling* dels *datasets* i del *Testbed*.
- **Investigador Júnior (I)**. Estudia els dos sistemes i dissenya i porta a terme els diferents experiments.
- **Analista Júnior (A)**. Extreu conclusions dels resultats dels experiments.
- ***Tester* Júnior (T)**. Fa una consolidació dels experiments i comprova que els resultats siguin correctes.

Cal destacar que el rol de **Cap de Projecte (CP)** involucra tant al director del TFG com a mi mateix, mentre que la resta dels rols els assumiré jo mateix. El cost per hora dels diferents rols es pot trobar a la Taula 16:

Rol	Salari brut anual	Salari brut per hora	Cost del rol per hora
Cap de Projecte [86]	55.163€	27,47€	35,71€
Arquitecte de *Software* Júnior [87]	48.923€	24,36 €	31,67€
Programador Júnior [88]	23.490€	11,70€	15,21€
Investigador Júnior [89]	25.630€	12,76€	16,59€
Analista Júnior [90]	27.220€	13,56€	17,63€
Tester Júnior [91]	19.146€	9,53€	12,39€

Taula 16: Salari brut per rols anual, per hora i cost del rol per hora
[Font: elaboració pròpia basada en sous per perfils de Glassdor [92]]

Tot i que els sous a la Taula 16 poden semblar més baixos que en realitat, cal tenir en compte que encara no he estat exposat al món laboral a Barcelona. Per tant, he utilitzat la pàgina web Glassdor [92] per a obtenir els sous anuals dels rols a la ciutat de Barcelona i poder utilitzar-los en l'estimació del pressupost.

Cal mencionar que la Taula 16 té una columna que ens indica el cost del rol per hora després d'aplicar el cost a pagar a la Seguretat Social que una organització ha de pagar per tenir contractat un rol; en aquest cas agafem el factor d'1,3, sobre el sou brut per hora. Per a convertir del cost anual al cost per hora hem utilitzat l'equació (7.1):

$$\text{cost}\ \frac{€}{\text{h}} = \text{cost}\ \frac{€}{\text{any}} \cdot \frac{1}{8}\ \frac{\text{dia}}{\text{h}} \cdot \frac{1}{251}\ \frac{\text{any}}{\text{dies laborals}} \tag{7.1}$$

Ara, tenint aquests costos per hora i per rol, podem analitzar per a cada tasca del Gantt, quantes hores són necessàries per a cada rol i multiplicant aquesta quantitat d'hores pel cost per hora, podrem obtenir el CPA (Cost del Personal per Activitat), tal com podem veure a la Taula 17.

ID	Tasca	Total Hores	Hores						Cost
			CP	AS	P	I	A	T	
GP1	Definir l'abast	25h	25h	-	-	-	-	-	892,75€
GP2	Definir la planificació temporal	10h	10h	-	-	-	-	-	357,10€
GP3	Fer el pressupost	10h	10h	-	-	-	-	-	357,10€
GP4	Fer l'informe de sostenibilitat	5h	5h	-	-	-	-	-	178,55€
GP5	Fer la documentació per a la fita de seguiment	30h	30h	-	-	-	-	-	1.071,30€
GP6	Fer la documentació per a la fita final	60h	60h	-	-	-	-	-	2.142.60€
GP7	Preparar la presentació del projecte	40h	40h	-	-	-	-	-	1.428,40€
GP8	Reunions setmanals	44h[1]	44h	-	-	-	-	-	1.571,24€
FM1	Familiarització amb els *frameworks*	40h	-	5h	10h	25h	-	-	725,20€
DP1	Definició de l'arquitectura	10h	-	10h	-	-	-	-	316,70€
DP2	Implementació del programa	50h	-	-	50h	-	-	-	760,50€
SD1	Recerca de *datasets*	10h	-	-	-	10h	-	-	165,90€
SD2	Definir les entrades	3h	-	-	-	3h	-	-	49,77€
SD3	Execució del *datasets profiler*	40h	-	-	-	40h	-	-	663,60€
SD4	Classificació dels *datasets*	10h	-	-	-	-	10h	-	176,30€
TB1	Definició de l'arquitectura	10h	-	10h	-	-	-	-	316,70€
TB2	Adaptació del *datasets profiler*	15h	-	5h	10h	-	-	-	310,45€
TB3	Implementació del programa	50h	-	-	50h	-	-	-	760,50€
IC1	Investigar com utilitzar el *datasets profiler* i el *Testbed* en el clúster	30h	-	5h	5h	20h	-	-	566,20€
IC2	Investigar com utilitzar la *Memory Storage Support* en HDFS	30h	-	5h	5h	20h	-	-	566,20€
SX1	Disseny dels experiments	10h	-	-	-	10h	-	-	165,90€
SX2	Execució dels experiments al clúster i recollida dels resultats	50h	-	-	-	50h	-	-	829,50€
SX3	Consolidació dels resultats i extracció de conclusions	20h	-	-	-	-	15h	5h	326,40€
MX1	Disseny dels experiments	10h	-	-	-	10h	-	-	165,90€
MX2	Preparació del clúster/*Testbed* per a executar els experiments	20h	-	5h	15h	-	-	-	386,50€
MX3	Execució dels experiments al clúster i recollida dels resultats	50h	-	-	-	50h	-	-	829,50€
MX4	Consolidació dels resultats i extracció de conclusions	20h	-	-	-	-	15h	5h	326,40€
	Total	**702h**	224h	45h	145h	238h	40h	10h	**16.407,16€**

Taula 17: Costos estimats per rols i tasques
[Font: elaboració pròpia]

7.1.2 Costos generals

Amortitzacions

Per a descriure el cost econòmic del projecte, haurem de tenir en compte l'amortització dels recursos tant *software*, com *hardware* que utilitzem. Cal tenir en compte, per a les dues amortitzacions, que treballaré 6 hores diàries durant 5 dies a la setmana, i durant un total de 117 dies.

Software

La majoria del programari que utilitzarem serà *Open Source*, per tant el seu ús serà gratuït, encara que utilitzarem els IDEs PyCharm i IntelliJ. Cal destacar que per a aquests *softwares*, utilitzarem la versió Ultimate. Per als estudiants universitaris s'ofereix un pla que permet utilitzar de forma gratuïta ambdós programes, per tant, el cost total del *software* serà de **0€**.

Hardware

Treballaré en total 117 dies en el portàtil. L'ordinador per a executar el *datasets profiler*, l'utilitzaré només durant 7 dies, mentre que el clúster d'ordinadors, l'estaré utilitzant durant

[1]Es calcula de la següent forma: duració d'una reunió (2h) x nombre de reunions (22)

un total de 37 dies. Hisenda permet amortitzar el *hardware* (HW) en 4 anys. Per tant, l'equació per a calcular l'amortització es pot observar a l'equació (7.2):

$$\text{Amortització} = \text{cost HW} \cdot \frac{\text{temps utilitzat}}{\text{període d'amortització}} = \text{cost HW (€)} \cdot \frac{\text{temps}}{4 \text{ anys} \cdot 251 \ \frac{\text{dies laborals}}{\text{any}}} \tag{7.2}$$

A la Taula 18 podem observar l'amortització del *hardware*. Cal notar que el clúster és un cost variable, ja que anirem canviant el nombre d'instàncies que utilitzarem, amb un màxim de 5 instàncies en tot moment.

Hardware	**Cost inicial**	**Ús del material**	**Amortització**
Portàtil Asus R5120VX0 [85]	759€	117 dies	88,45€
Ordinador per a analitzar els *datasets* [38]	1.701,07€	7 dies	11,86€
Clúster amb 5 instàncies [93]	6,85€/dia	37 dies	253,45€
Total	-	-	**353,76€**

Taula 18: Cost de l'amortització del *hardware*
[Font: elaboració pròpia]

Costos indirectes

Desenvoluparé des de casa tot el TFG i les reunions seran totes en línia. Per tant, cal considerar també les despeses necessàries per a portar a terme aquest projecte treballant des de casa. Considerarem diversos costos indirectes, com l'electricitat, l'internet i el lloguer d'un pis a Barcelona.

- **Cost de l'electricitat.** El preu del kWh és 0,1143€ [94] i aproximadament el consum d'un ordinador portàtil en funcionament és de 50W [95]. Per tant, tenim que 0,1143€/kWh · 0,05kW · 117 dies · 6h/dia = 4,01€. El consum de l'ordinador utilitzat per a analitzar els *datasets* és d'entre 80W i 150W [95]. Suposarem que el consum és 150W. Aleshores, el cost serà de 0,1143€/kWh · 0,15kW · 7 dies · 6h/dia = 0,72€. En total, el cost de l'electricitat és de **4,73€**.
- **Cost de l'internet.** El cost de l'internet és de 29,90€/mes [96]. Per tant, si calculem el cost de l'internet, durant el temps que l'estiguem utilitzant, aquest és: 29,90€/mes · (1/30) mes/dies · 117 dies · 6h/dia · (1/24) dia/h = **29,15€**.
- **Cost del lloguer.** El preu mitjà per habitació a Barcelona és de 443€/mes [97]. Per tant, el cost serà de: 443€/mes · (1/30) mes/dies · 117 dies · 6h/dia · (1/24) dia/h = **431,93€**.

Costos generals

En la Taula 19 reunirem tots els costos generals que hem analitzat amb anterioritat i calculem el valor total, que es correspon amb el CG o Costos Genèrics.

Tipus de cost general	Cost econòmic
Amortització de *Software*	0€
Amortització de *Hardware*	353,76€
Cost de l'electricitat	4,73€
Cost de l'internet	29,15€
Cost del lloguer	431,93€
Total	**819,57€**

Taula 19: Costos generals del projecte
[Font: elaboració pròpia]

7.2 Estimació dels costos

Per acabar, tindrem en compte el cost de la contingència i dels imprevistos. La contingència és d'un 15% en el sector. Això es tradueix en un cost addicional de (CPA + CG) · 0,15 = (16.407,16€ + 819,57€) · 0,15 = **2.584,01€**. Finalment, a la Taula 20 podem observar els costos associats als imprevistos que poden sorgir durant l'execució del projecte.

Descripció	Temps	Risc	Cost estimat	Cost
Data d'entrega fixada	50h	70%	760,5€	532,35€
Bugs	40h	50%	937,6€	468,8€
Corrupció de les dades	20h	15%	714,2€	107,13€
Complexitat dels *frameworks*	0h	10%	0€	0€
Capacitat computacional	222h	5%	74,55€	3,73€
Total	-	-	-	**1.115,01€**

Taula 20: Costos estimats dels imprevistos del projecte
[Font: elaboració pròpia]

El cost d'un imprevist es calcula multiplicant el seu cost total pel risc que passi. En el cas del risc **data d'entrega fixada**, suposem que s'hauran de treballar 50h addicionals per a completar el TFG. On hi ha més risc és en la feina del programador, ja que el cost d'implementar el codi és difícil de preveure amb molta exactitud. Aleshores, suposem que aquestes 50h són del programador exclusivament. Els ***bugs*** hauran de ser solucionats o bé per l'arquitecte o bé pel programador; per tant, distribuirem 40h entre els dos rols al 50%. Per al risc de la **corrupció de les dades**, el cap de projecte haurà de dedicar 20h per a reescriure les parts perdudes. Per al risc de la **complexitat dels *frameworks***, l'investigador, programador i arquitecte necessitaran més temps, per a poder entendre el funcionament dels *frameworks*. En aquest cas, només es reduirà el nombre d'experiments i es dedicarà més temps a entendre els *frameworks*, ja que és una part fonamental de la comparació. Finalment, en el risc **capacitat computacional**, necessitarem gastar 37 dies més d'energia, que és el temps que es dedicava per a processar les dades en el clúster.

7.2.1 Pressupost final

Finalment, reunim en la Taula 21 el pressupost final a partir de les tasques definides en el capítol 6.

Activitat	**Cost**
CPA	16.407,16€
CG	819,57€
Contingència	2.584,01€
Imprevistos	1.115,01€
Total	**20.925,75€**

Taula 21: Cost total del projecte
[Font: elaboració pròpia]

7.3 Control de gestió

En aquesta secció descriurem com modelarem les potencials desviacions de pressupost. El procediment a seguir és el següent: A mesura que s'executa cada una de les tasques, es va recollint el temps real necessari i els materials necessaris per a completar la tasca. Això ho monitorarem calculant els indicadors de les equacions (7.3) i (7.4):

$$\text{Desviació mà d'obra en consum} = (\text{consum hores estimat} - \text{consum hores real}) \cdot \text{cost estimat} \tag{7.3}$$

$$\text{Desviació matèria primera en consum} = (\text{consum estimat} - \text{consum real}) \cdot \text{cost real} \tag{7.4}$$

A les equacions (7.3) i (7.4) es parla d'una desviació en el cost per eficiència. D'aquesta forma, al final del projecte, podrem identificar on s'ha produït la desviació i la seva quantia.

7.4 Desviacions de la planificació inicial

Com hem explicat prèviament, vam fer canvis a la planificació per a dedicar més temps al desenvolupament del *Testbed*, i això va afectar als costos del projecte, fent més car el projecte del que vam preveure, encara que aquestes desviacions no van augmentar el total cost del projecte perquè en la secció 7.2 es van contemplar uns costos per a poder gestionar els possibles problemes que poguessin sorgir.

Capítol 8

Sostenibilitat i compromís social

8.1 Autoavaluació

Com a reflexió, he vist que el concepte de sostenibilitat té diferents enfocaments que desconeixia, que reflecteixen la forma en la qual una activitat o projecte pot ser sostenible amb el medi ambient; és a dir que no perjudiqui el medi ambient, que sigui sostenible en l'àmbit econòmic, és a dir que el projecte o idea es pugui mantenir en un llarg termini, i ser sostenible en una societat, és a dir, si la societat accepta aquell projecte o idea. Analitzar un projecte abans que aquest sigui implementat en aquestes tres perspectives, dóna un gran avantatge, ja que et diu si aquell projecte no perjudicarà el medi ambient, si et podrà generar un benefici, i si triomfarà en una societat. Molts cops, abans de fer aquest projecte només pensava que la sostenibilitat només tenia a veure amb el medi ambient, tot i que m'he adonat amb el desenvolupament d'aquest treball que és un concepte que transcendeix aquesta perspectiva i, de fet és un aspecte transcendental, que pot condicionar l'èxit o el fracàs del projecte.

Ara bé, ja sabem que la sostenibilitat és un aspecte fonamental, però haurem de quantificar d'alguna forma com contribueix el nostre projecte als punts de vista de la sostenibilitat, és a dir, necessitem una sèrie d'indicadors que en avaluar-los ens indiquin quant de sostenible o no és el nostre projecte. Avaluant els indicadors podrem mantenir sempre una visió objectiva de com està funcionant la sostenibilitat del nostre projecte i ens ajudarà a detectar desviacions i solucionar-les a temps abans que el nostre projecte fracassi. Aquest aspecte és un punt fort que tractem en el nostre projecte, sobretot en l'àmbit econòmic, tal com podem veure a l'hora de definir indicadors per a mesurar les desviacions, en la secció 7.3, com també en l'àmbit ambiental, en la secció 8.2.

Aquesta enquesta també m'ha fet veure que existeixen uns principis deontològics relacionats amb la sostenibilitat, que se centren bàsicament a fer complir uns mínims establerts. Aquesta és una mesura molt important, ja que asseguraria que tots els projectes poguessin tenir un mínim de sostenibilitat. Això permetria fer pensar millor als enginyers, per exemple quan han d'escollir entre dues opcions i una pot implicar gastar més CO_2 que l'altra. O bé que en l'àmbit social, que el projecte estigués acceptat per la gent i que aquesta l'utilitzés perquè els resol una necessitat real.

8.2 Dimensió ambiental

Respecte al PPP: Has estimat l'impacte ambiental que tindrà la realització del projecte? T'has plantejat minimitzar l'impacte, per exemple, reutilitzant recursos?

A causa de la naturalesa del projecte, que es tracta d'una investigació, és difícil fer una estimació acurada de l'impacte ambiental. Tot i així, vam dissenyar la tasca de definir les entrades per al *datasets profiler* de tal forma que per als *datasets* grans, que requerien un ampli poder de processament, o fins i tot, més d'un ordinador per a poder calcular eficientment els estadístics, vam decidir agafar-ne diverses mostres representatives que puguin representar la mostra completa. D'aquesta forma, es redueix considerablement la petjada ecològica d'haver d'executar el programa per als *datasets* complets. Addicionalment, cal destacar que a l'hora de fer un *profiling* dels *datasets*, cada execució del *datasets profiler* mostra el cost d'execució de cada funció per tal de veure on es podrien introduir noves millores d'eficiència en un futur, que conseqüentment contribuiran a reduir la petjada ecològica.

La reutilització de recursos per a aquest projecte, constaria bàsicament de reutilitzar investigacions ja existents o programes que facin el mateix que el *Testbed* o el *datasets profiler*. Encara que vam optar per construir els dos programes des de zero, ja que ens aportaria el màxim de flexibilitat a l'hora de fer les proves.

L'indicador que utilitzarem per a mesurar l'impacte ambiental al llarg del TFG és la quantitat de kWh consumits per l'ordinador que executarà el *datasets profiler*, el clúster que executa el *Testbed* i del portàtil. De tots aquests, es farà un seguiment de les hores que s'estan utilitzant, s'anotarà aquest valor i es calcularà el valor numèric de l'impacte amb l'equació (8.1):

$$\text{Impacte en kWh} = \text{potència elèctrica en kW} \cdot \text{nombre d'hores que s'ha utilitzat un dispositiu} \tag{8.1}$$

El valor d'aquest indicador, el podrem comparar amb el valor estimat que es pot trobar a la secció 7.1.2 i mesurar l'impacte individual de cada dispositiu; sumant tots els impactes dels dispositius, obtindrem l'impacte total del nostre projecte.

Respecte al PPP: Has quantificat l'impacte ambiental de la realització del projecte? Quines mesures has pres per reduir l'impacte? Has quantificat aquesta reducció?

Durant el desenvolupament dels sistemes *Datasets Profiler* i *Testbed*, vaig escollir sempre els algorismes més eficients i vaig tenir sempre al cap la complexitat dels algorismes per a que quan aquests s'executessin no malbaratessin recursos. La quantificació de la reducció es va especificar als apartats d'implementació dels dos sistemes en les seccions 3.3 i 4.3, on es feia una comparació entre les diferents alternatives i el seu cost.

Respecte al PPP: Si fessis de nou el projecte, podries fer-ho amb menys recursos?

Aquest projecte m'ha ajudat a adquirir més coneixements en el desenvolupament de codi per a *frameworks* de processament de dades i saber dissenyar arquitectures complexes, per

tant, crec que sí podria fer-ho amb menys recursos, generalment en quant a temps. Tot i que no totes les parts es podrien fer amb menys recursos, com l'escriptura de la documentació, que implicaria la mateixa inversió de temps.

Respecte a la Vida Útil: Com es resol actualment el problema que vols abordar (estat de l'art)? En què millorarà ambientalment la teva solució respecte a les existents?

Actualment, en l'àmbit ambiental, moltes organitzacions tant públiques com privades, es decanten directament per utilitzar només Spark, ja que es considera una tecnologia més nova, obviant el fet que potser el que necessitin utilitzar per als seus casos d'ús és el *framework* de processament de dades MapReduce. Un exemple d'organització privada que utilitza Spark, per exemple és Yahoo, que l'utlitza per a personalitzar les notícies i per a posar anuncis enfocats a l'usuari [98]. I una organització sense ànims de lucre que està utlitzant un d'aquests sistemes és Global Emancipation Network (GEN), que es dedicada a posar fi al tràfic de persones; utilitza els *Big Data* per a detectar els traficants i a les víctimes [99]. Aquestes organitzacions podran reduir la seva petjada ambiental per a fer la seva activitat més sostenible. Aquesta investigació afectaria els dos tipus d'organitzacions perquè a les grans empreses, un petit estalvi energètic per a processar les dades, pot suposar una gran reducció en la petjada ambiental. Per a les organitzacions sense ànims de lucre, normalment els diners dels quals disposen provenen de donacions, per la qual cosa, molts cops no en tenen suficients per a dedicar a investigacions de quin sistema seria millor per al seu cas d'ús i reduiria la petjada ambiental. Per tant, aquestes podran escollir una opció que els permeti fer un processament més eficient basant-se en els resultats d'aquesta anàlisi.

L'indicador de la petjada ecològica que tindrà el nostre projecte durant tota la seva vida útil es mesurarà de la mateixa forma que es fa a l'apartat anterior, mesurant el nombre d'hores que s'utilitzen l'ordinador que executa el *datasets profiler*, del clúster que executa el *Testbed* i del portàtil i després multiplicant aquest nombre d'hores per la seva potència elèctrica. Finalment, la petjada ecològica de tot el projecte durant la seva vida útil s'obtindria de sumar els valors de les diferents petjades ecològiques individuals dels diferents dispositius.

Respecte la Vida Útil: Quins recursos estimes que s'usaran durant la vida útil del projecte? Quin serà l'impacte ambiental d'aquests recursos?

És difícil estimar quins recurs s s'usaran durant la vida útil del projecte per la seva naturalesa d'investigació, ja que es depèn directament de la quantitat d'organitzacions utilitzin els resultats d'aquest projecte.

Respecte la Vida Útil: El projecte permetrà reduir l'ús d'altres recursos? Globalment, l'ús del projecte millorarà o empitjorarà la petjada ecològica?

Sí, si els *stakeholders* del projecte apliquen els resultats d'aquest projecte per a fer més eficients els seus sistemes, aquest projecte pot ajudar a millorar la petjada ecològica. Per tant, globalment, l'ús d'aquest projecte millorarà la petjada ecològica ja que busca els punts on cada eina treballa millor.

Respecte els Riscos: Podrien produir-se escenaris que fessin augmentar la petjada ecològica del projecte?

No es podrien produir escenaris que fessin augmentar la petjada ecològica del projecte per la naturalesa d'investigació del projecte. Tot i així, està en mans dels enginyers que facin les implementacions a les organitzacions el fet d'utilitzar aquesta memòria adequadament per a extreure el màxim de benefici dels resultats obtinguts.

8.3 Dimensió econòmica

Respecte al PPP: Has estimat el cost de la realització del projecte (recursos humans i materials)?

En el capítol 7 es va fer una descripció acurada dels recursos humans i materials necessaris, com també els indicadors que ens permetran ser conscients de les possibles desviacions en el pressupost.

Respecte al PPP: Has quantificat el cost (recursos humans i materials) de la realització del projecte? Quines decisions has pres per reduir el cost? Has quantificat aquest estalvi?

El cost previst i el cost final totals han sigut iguals als estimats inicialment en quant a recursos humans i materials, amb certes desviacions que es van justificar a la secció 6.4. La decisió més important per a reduir el cost va ser la implementació del *Testbed*, que tot i que ens va portar més temps a l'hora de desenvolupar-lo, vam aconseguir compensar aquest cost.

Respecte al PPP: S'ha ajustat el cost previst al cost final? Has justificat les diferències (lliçons apreses)?

El cost previst i el cost final totals han sigut iguals perquè les desviacions van ser cobertes per la part del pressupost dedicada a imprevistos i contingències. Les desviacions es van justificar més en profunditat a la secció 6.4.

Respecte a la Vida Útil: Com es resolen actualment els aspectes de costos del problema que vols abordar (estat de l'art)? En què millorarà econòmicament la teva solució respecte a les existents?

Actualment, el problema radica en el fet que les organitzacions tendeixen més a seguir una moda o el màrqueting anunciat per una altra organització que vol vendre el seu producte, sense prestar atenció en si aquella solució és en realitat bona segons la forma en la qual volen utilitzar el sistema. Per tant, tal com hem dit, en alguns casos fins i tot, la implantació d'un nou sistema pot arribar a causar un malbaratament dels recursos. Austin YMCA [99] és una organització sense ànim de lucre dedicada a construir una comunitat més unida i connectar a les persones [100]. Aquesta organització ha fet una *partnership* amb MVP Audience per a incrementar les donacions. Aleshores, veiem que en el cas de les organitzacions sense ànim de lucre, el seu finançament prové majoritàriament de subvencions i donacions. D'aquesta forma, no tenen suficients diners per a poder fer investigacions exhaustives per a veure quin

dels sistemes els pot ser més útil i es veuen obligades a seleccionar aquell sistema que sembli millor segons el màrqueting. La mateixa lògica s'aplica per a empreses emergents com per exemple Looker [101], que és una empresa que dóna solucions de *Big Data*. La solució que nosaltres oferim no és més que un repertori comprovat empíricament dels casos d'ús en els que els dos sistemes es comportarien de forma objectiva un millor que l'altre. D'aquesta forma els enginyers de les organitzacions podran prendre decisions més informades, per tal de causar el màxim estalvi econòmic a l'organització on s'apliquen els resultats d'aquest projecte.

Respecte a la Vida Útil: Quin cost estimes que tindrà el projecte durant la seva vida útil? Es podria reduir aquest cost per fer-ho més viable?

És difícil fer aquesta estimació perquè depèn directament de qui utilitzarà aquest resultat. Si els nostres *stakeholders* utilitzen aquest projecte per als seus sistemes, l'ús dels resultats d'aquest projecte els podria ajudar a obtenir guanys depenent a la grandària de l'empresa que utilitzi aquestes eines i, en quina mesura les utilitzen. Per tant, veiem que és difícil fer aquesta estimació.

Respecte a la Vida Útil: S'ha tingut en compte el cost dels ajustos / actualitzacions/ reparacions durant la vida útil del projecte?

En aquest projecte, hem fet les dues implementacions dels sistemes de forma molt modular i canviable, de tal forma que els canvis o actualitzacions fossin senzills de realitzar, encara que no tenia massa sentit calcular aquests costos perquè els sistemes desenvolupats són utilitzats més per a fins d'investigació, per tant, no tenen una vida útil com els sistemes en producció.

Respecte als Riscos: Podrien produir-se escenaris que perjudiquen la viabilitat del projecte?

En cas que hi sorgeixin nous *frameworks* de processament de dades, es podria donar el cas que els resultats d'aquest projecte ja no fossin útils per a les empreses, encara que sí serien útils per als desenvolupadors, que veurien en quins casos una eina és millor que l'altra i podrien afegir aquestes modificacions a les noves eines que apareguin. En aquest escenari, el *Testbed* que hem dissenyat es podria adaptar, creant nous adaptadors per a que funcioni amb nous *frameworks* de processament de dades.

8.4 Dimensió social

Respecte al PPP: Què creus que t'aportarà en l'àmbit personal la realització del projecte?

Crec que aquest projecte m'enriquirà per diversos motius: primer, m'ajudarà a obtenir una visió estratègica per a executar més eficientment els projectes, tant per a organitzar-los, determinar el seu abast, o veure quan són fiables. Finalment, en l'àmbit tècnic, adquiriré els coneixements del funcionament de dos dels *frameworks* més populars del mercat existents actualment, que són unes habilitats molt ben reconegudes en el món laboral, com també la

forma en la qual s'hauria de fer una investigació, ja que és el primer cop que faig un projecte gran d'investigació.

Respecte al PPP: La realització d'aquest projecte ha implicat reflexions significatives a nivell personal, professional o ètic de les persones que han intervingut?

La realització d'aquest projecte m'ha fet veure la importància que tenen les dades en el món actual i la necessitat de crear eines com Spark i MapReduce que solucionen el problema d'haver de processar grans quantitats de dades amb sistemes distribuïts. Aquest projecte em va servir també per a veure el documental "A social dilemma", on vaig veure com coses que no donem importància en el dia a dia, com per exemple fer els *likes* a les fotografies de les xarxes socials, o les nostres accions en les pàgines web, poden ser fons importants d'informació que les empreses grans poden explotar i aprofitar-se d'aquestes.

Respecte a la Vida Útil: Com es resol actualment el problema que vols abordar (estat de l'art)? En què millorarà socialment (qualitat de vida) la teva solució respecte a les existents?

Actualment, moltes organitzacions es decanten per l'opció més nova. Per tant, la qualitat dels sistemes, podria millorar en aquest sentit si s'agafés el sistema adequat per al cas d'ús que es consideri. Per a determinar com afectarà aquesta solució en la qualitat de vida de les persones, cal recalcar que hi ha diversos tipus d'organitzacions aprofitant-se dels resultats d'aquesta investigació. En primer lloc, hi ha les organitzacions sense ànim de lucre o cooperatives, com per exemple Som Energia [102], que és una cooperativa de consum d'energia verda sense ànims de lucre. Se centren en la comercialització i producció d'energia d'origen renovable. Un altre exemple és Creu Roja [103], que utilitza els *Big Data* per a salvar i millorar vides. Aquestes organitzacions fan un ús dels *Big Data* que resulta beneficiós per a la societat. En l'altra cara de la moneda, hi ha les organitzacions amb ànim de lucre o empreses privades. En especial, cal fer menció a com s'estan tractant avui dia les dades per grans empreses com Google i Facebook, tal com es pot veure al documental el Dilema Social [104], on es relata la cara obscura de les xarxes socials i mostra les addicions que poden provocar les xarxes socials i els impactes negatius que pot provocar en les persones i comunitats a conseqüència de les estratègies que prenen les grans empreses per a manipular emocions i comportaments i mantenir connectats als usuaris [28]. Aquest documental recalca la falta d'eticitat d'aquestes grans empreses i com fan tot el possible per a lucrar-se dels seus usuaris, sense pensar en com s'està danyant la societat. Finalment, podem veure que la nostra solució no necessàriament fa un bé a la societat, tot i així estem tractant d'investigar quina eina en quins casos d'ús és millor que l'altra, independentment de com s'utilitzi aquesta, ja que com hem exposat amb anterioritat, depenent d'en quines mans estigui aquesta eina pot fer el bé o el mal a la societat.

Respecte a la Vida Útil: Existeix una necessitat real del projecte?

Aquest projecte no és una necessitat real més enllà de donar una oportunitat a les organitzacions d'aprofitar un estudi de diversos casos d'ús on es pot observar en quins escenaris un sistema és millor que l'altre i permetre així escollir entre els dos. D'aquesta forma s'es-

talvia temps a les organitzacions, ja que aquestes no hauran de fer el mateix estudi que nosaltres estem fent. Tot i que no és un projecte imprescindible, té un impacte molt gran, ja que una mala decisió en la selecció d'un *framework* respecte a l'altre, pot deteriorar el rendiment dels sistemes que utilitzin aquests *frameworks*.

Respecte a la Vida Útil: Qui es beneficiarà de l'ús del projecte? Hi ha algun col·lectiu que es pot veure perjudicat pel projecte? En quina mesura?

Es poden veure en la secció 1.1.4 totes les parts implicades, que es veuran de forma directa o indirecta beneficiades per la realització d'aquest projecte. Directament, aquest projecte no perjudica a cap col·lectiu, encara que com hem detallat a la capítol 2, sí és possible que el seu desenvolupament afecti negativament a algun col·lectiu per com algunes empreses poden utilitzar en el seu benefici les dades, per tant sí afectaria indirectament.

Respecte a la Vida Útil: En quina mesura soluciona el projecte el problema plantejat inicialment?

En aquest projecte hem creat els sistemes necessaris per a poder executar experiments sobre els dos *frameworks* de processament de dades i veure en quins casos d'ús cada sistema funcionaria millor, que són justament els dos objectius que ens vam proposar. El problema inicial està solucionat perquè hem pogut proporcionar una comparació seguint els criteris que vam establir.

Respecte als Riscos: Podrien produir-se escenaris que fessin que el projecte fos perjudicial per a algun segment particular de la població?

En el capítol 2 es menciona l'ús poc ètic que es fa per part d'algunes empreses que aprofiten aquestes les dades en el seu benefici i això pot ser en alguns casos perjudicial per a la població, encara que directament l'aplicació del projecte no provocaria cap escenari perjudicial per si mateix.

Respecte als Riscos: Podria crear el projecte algun tipus de dependència que deixés als usuaris en posició de debilitat?

El projecte en sí no pot crear cap tipus de dependència. Tot i així, tal com vam comentar en el capítol 2, es pot donar el cas que empreses grans utilitzin aquestes eines de processament de dades en el seu favor i, per això poden fer que els usuaris seus puguin crear algun tipus de dependència. Tot i així, tal com vam recalcar abans, no és possible que es creés algun tipus de dependència directament.

Capítol 9

Conclusions

9.1 Treball Futur

En quant al treball futur d'aquest projecte, principalment volíem destacar un component a afegir al *Testbed*, que permet fer optimitzacions abans d'executar les operacions tant per a Spark com per a MapReduce. Aquestes optimitzacions serien, en el cas de Spark, afegir caching per a les operacions que es reutilitzen per a evitar una reexecució del *pipeline*. Vam pensar primer que aquest aspecte seria rellevant perquè no sabíem al començament si Spark funcionaria correctament amb el conjunt de dades gran i creiem que MapReduce podria millorar els resultats de Spark, encara que en la realitat, això no va passar, com podem veure en els resultats dels experiments. Tot i així, si es volen utilitzar *pipelines* més complexes i/o conjunt de dades més grans, es recomanaria implementar aquesta millora.

En el cas de MapReduce, la millora que volíem implementar era la fusió de l'operació projecció amb l'operació de selecció. Sembla que aquesta fusió no doni una gran millora, encara que en la realitat, al poder executar qualsevol codi en els Mappers, es podria fer aquesta optimització per a evitar l'execució de dos *jobs* i executar-ne només un. Aquesta millora vam considerar que no afectaria molt ja que finalment, observem en els experiments que MapReduce és superat per Spark amb la configuració del nostre clúster fins i tot per a les operacions més senzilles, que són la selecció i projecció.

Per falta de temps, vam reduir la quantitat d'operacions que vam implementar; inicialment, volíem fer experiments també sobre operacions com per exemple diferència de conjunts de dades, intersecció i producte cartesià. Aquestes operacions no les vam afegir al *Testbed*, encara que en una extensió futura del projecte, es podrien implementar i afegir fàcilment per l'estructura que té el *Testbed*.

Una altra millora que no vam tenir temps per implementar era estendre el *Testbed* per a executar *pipelines* més complexos, on es podria experimentar amb execucions d'una mateixa operació de selecció i projecció múltiples cops i en diferents llocs del *pipeline*. Tot i així, vam optar per poder tenir una versió més petita i menys complexa dels *pipelines* per a poder tenir els experiments més rellevants per a provar les coses més bàsiques.

Volíem destacar també que dels experiments que es poden executar amb la implementació

actual del *Testbed*, només vam executar un subconjunt per falta de recursos com són el temps o el cost econòmic del clúster. Tot i així, volíem destacar que hem deixat el codi públic per si algun *stakeholder* vol executar altes experiments.

De la part d'experimentació, quedaria pendent el fet de provar el comportament dels dos sistemes amb un conjunt de dades molt gran, com per exemple Edgar, que el vam analitzar en el Capítol A dels apèndixs.

Un altre treball de millora seria continuar amb els experiments sobre les operacions iteratives o execució dels experiments amb la característica de *Memory Storage Support* activada, que vam decidir cancel·lar per a dedicar més temps a la implementació del *Testbed*.

Un altre punt important, que ens ajudaria a guanyar més rigorositat és l'ús de tests unitaris i d'integració en el *Testbed*, ja que d'aquesta forma podríem validar que no hi hagi cap *bug* en el codi. Tot i així, vam comprovar manualment que totes les parts del *Testbed* funcionessin correctament abans de fer l'execució dels experiments. Tot i així, caldria automatitzar aquest procés per a que cada cop que s'insereixi una millora al codi, es pugui comprovar que la resta del codi es manté igual.

Com a treball futur, el *Testbed* es pot estendre també per a fer altres comparatives entre altres *frameworks* o bases de dades relacionals, on es podria aprofitar el fet que les operacions no són traduïdes a un codi específic del *framework* fins just abans que s'invoquin.

Finalment, cal aclarir que els resultats presents en aquest projecte tenen un valor acadèmic perquè poden afegir més informació a assignatures de *Big Data* basant-se en els resultats obtinguts.

9.2 Conclusions del projecte

En conclusió, podem afirmar que s'han complert els objectius d'aquest treball proposats en la secció 1.4.1. Addicionalment, també vam complir amb les competències específiques marcades per al projecte. Volem destacar que vam fer un manteniment i avaluació de sistemes i serveix *software* complexos com són Spark i MapReduce per mitjà d'una experimentació (CES 1.1). Per a automatitzar el procés d'extracció dels perfils estadístics que vam utilitzar per a seleccionar quins conjunts dades d'entrada havíem d'utilitzar i per a automatitzar l'experimentació, vam desenvolupar i mantenir dues aplicacions distribuïdes amb suport de xarxa, que són el *Testbed* i el *Datasets Profiler* (CES 1.4). En aquest projecte hem demostrar la comprensió de la gestió i govern dels sistemes *software* desenvolupats una mica ja que no hem aprofundit en com s'haurien de mantenir el *Testbed* i el *Datasets Profiler* al llarg termini, encara que sí que vam fer un disseny de l'arquitectura d'aquests que permetés una futura fàcil extensió (CES 1.9). Els conjunts de dades utilitzats per als experiments, es van processar i transformar per mitjà d'un procés de *parsing* a través del *Datasets Profiler* (CES 1.5). Aquests conjunts de dades es podrien considerar com a "bases de dades" si estiguéssim parlant en el món relacional, encara que per a les bases de dades específiques, ens hauríem de referir a conjunts de dades. En el seu conjunt, podem afirmar hem tractat en profunditat la competència transversal d'administració de bases de dades no relacionals ja que hem hagut de configurar els sistemes i també desenvolupar codi que pogués ser utilitzat per aquests

(CES 1.6).

En aquest projecte, vam veure que Spark és millor en tots els aspectes, de forma que té una visió més global que MapReduce a l'hora d'executar els experiments i fa una gestió dels recursos més eficient. Saber quina operació s'està executant, permet utilitzar un algorisme correcte i eficient que permeti fer una execució també eficient de l'operació. Spark té implementats un repertori d'algorismes àmpliament provats i optimitzats, mentre que MapReduce no els té i deixa en mans del programador tota la feina de recerca dels algorismes i optimitzacions. Conèixer quina operació d'està executant, permet a Spark fer optimitzacions, com en els sistemes relacionals, a diferència de MapReduce, on totes les optimitzacions les ha d'aplicar el programador en els Mappers i Reducers. En els últims anys, Spark ha tingut una evolució notable, on la implementació s'ha fet cada cop més eficient; era aquest un dels motius pels quals avui dia era necessari fer una comparativa per a veure com ha afectat l'evolució de Spark i veure si realment és necessari seguir utilitzant MapReduce o no. Per les conclusions extretes, vam veure que l'evolució de Spark en els últims anys ha sigut tant forta que ara mateix, amb els usos normals, no és rentable utilitzar MapReduce pels problemes i pels volums de dades de grandàries relatives amb els que hem experimentat. Recordem, però que les conclusions que vam extreure dels dos sistemes es basaven exclusivament en els experiments i configuració del clúster explicats, encara que existeix la possibilitat que amb conjunts de dades molt més grans que els provats, MapReduce pugui tenir una avantatge respecte a Spark.

La meva conclusió de treballar amb aquest tipus de sistemes *software* és que són *frameworks* diferents, encara que segueixen una mateixa idea. Spark està dissenyat per a ser més senzill a nivell de codi que MapReduce, encara que el seu funcionament intern és molt complex en comparació amb MapReduce. En el seu torn, MapReduce és un sistema molt fàcil d'entendre, però treballar amb aquest a nivell de codi, requereix un gran esforç i una bona aplicació dels principis SOLID per a evitar que el codi degeneri ràpidament i sigui difícil de mantenir. També cal destacar que estem utilitzant *frameworks* i aquests funcionen aproximadament com les llibreries d'un programa. Si volem que la nostra aplicació que estem desenvolupant ens permeti en un futur canviar de de *framework*, hem de desacoblar el màxim possible aquest sistema de la nostra implementació, tal com vam fer en l'arquitectura del *Testbed*.

Un tema important a destacar, durant la realització del projecte és que ens va portar una gran quantitat de temps fer el *porting* del *Testbed* i del *Datasets Profiler* al clúster; tots aquests problemes fan que arribem a la conclusió que encara avui dia no hi ha una forma fàcil de crear un sistema que funcioni en local i de forma distribuïda sense massa esforç. Això vol dir que actualment, les organitzacions han de fer un esforç gran si volen fer un desenvolupament del codi en local i posteriorment desplegar-lo en un clúster. Hi ha moltes variables que han de ser considerades a l'hora de desenvolupar un *software* que tenen que veure amb on estan ubicades les dades i també la configuració que s'ha de fer dels *frameworks*. Aquests detalls fan que sigui molt més difícil fer un *porting* sense problemes al clúster, a diferència de com s'anuncien aquest tipus de sistemes. Hem de dir en el seu favor que sí oculten la major part de la complexitat de treballar en un sistema distribuït, com per exemple el *scheduling* dels *jobs*, l'execució distribuïda, etc., però no el suficientment com per

fer un desenvolupament igual que en un entorn local.

Cal recalcar que el procés de fer la migració a clústers ja preconfigurats per proveïdors de serveis al *Cloud*, com per exemple Amazon Web Services (AWS) o Google Cloud Platform (GCP), es converteix en una tasca molt més senzilla que si hem de fer nosaltres tota la configuració dels servidors que tinguem en les instal·lacions, també anomenats *on-premises servers*.

Durant el desenvolupament dels diferents sistemes en aquest treball, vam trobar que en el món del *Big Data*, acostumava a haver-hi una falta de tests unitaris que permetin validar en tot moment que l'execució és correcta. Aquests tests són molt utilitzats en els sistemes en producció, però no s'apliquen tant en els sistemes del món del *Big Data*. Cal destacar que nosaltres no vam dedicar temps a la creació de tests unitaris per a aquest projecte perquè vam considerar que al no estar en producció aquests sistemes, no era necessari. Tot i així, és un aspecte d'enginyeria de *software* que es pot tenir en compte per a donar més rigorositat als resultats obtinguts.

Un altre aspecte que vam poder observar és que en el món del *Big Data* primer es selecciona un *framework* de processament de dades i després es procedeix a fer operacions utilitzant aquest sistema. Cal destacar que des del punt de vista d'un enginyer de *software*, aquest comportament és molt dolent respecte els principis SOLID, ja que bàsicament ens estem acoblant amb un detall d'implementació. Els *frameworks* de processament de dades és simplement un detall d'implementació, que s'ha de crear de forma desacoblada de la resta de la lògica de negoci i s'han de posar en adaptadors, com vam fer en el desenvolupament del *Testbed*. També ens hem de mostrar especialment crítics en la decisió de disseny dels *frameworks*, que fomenten aquests mals hàbits d'acoblar el codi Spark o MapReduce amb el codi que conté la lògica de negoci. Tota la documentació que hem trobat, no estableix que la demostració que s'està fent s'hagi de posar en adaptadors ni tampoc es mostra com fer-lo, de tal forma que un lector podria interpretar que la millor manera d'utilitzar els *frameworks* és copiant i pegant els exemples de la documentació i adaptar-los lleugerament.

Cal destacar també que hi ha hagut un interès meu pel món del *Big Data*, on veia que les organitzacions donaven molt valor a les dades i hi havia un ús molt gran de *buzz words* al voltant d'aquest món que eren difícils de comprendre. En realitzar l'assignatura de Coneixements de Bases de Dades Especialitzades i haver complementat aquells coneixements amb aquest Treball de Final de Grau, puc concloure que vaig aprendre molt respecte al funcionament dels sistemes distribuïts i també he adquirit habilitats de carca d'informació i anàlisi d'aquesta que seran molt útils en el món laboral. Tot aquest interès també es va veure reflectit en la gran quantitat d'hores dedicades, que són clarament molt per sobre del que s'hauria de dedicar normalment a un TFG.

Un altre aspecte a destacar d'aquest treball és que hem creat un *Testbed* amb una arquitectura que permet una gran flexibilitat i que es pot estendre fàcilment per a poder tractar *pipelines* molt complexos entre Spark i MapReduce o també es pot estendre fàcilment per a fer comparacions entre altres sistemes; només caldria implementar els adaptadors de les operacions. La intenció d'aquest sistema és utilitzar-lo posteriorment per a fer una recerca més avançada per part dels nostres *stakeholders*, que disposen ara d'una eina per a fer execucions

dels experiment d'una manera més fàcil.

Finalment, hem de destacar el fet que hem crear un format de configuració, els *pipelines* que s'utilitza com a entrada del *Testbed* en Json. Aquest format és molt senzill d'entendre i es podria reaprofitar per a altres sistemes. Aquest és un llenguatge de configuració orientat a *flows* d'alt nivell i que permet abstreure'ns d'escriure Spark SQL o codi específic de MapReduce per a un experiment en concret. Els altres llenguatges de configuració existents actualment, com per exemple Pig, són de molt baix nivell. Actualment, no hi ha cap altre llenguatge que supleixi el que estem fent. A més, estem obrint la porta a que ara es faci una eina visual per a que arrossegant capsetes i configurant mínimament, es pugui crear una configuració d'un *pipeline* i aquesta es pugui executar tot automàticament.

Bibliografia

[1] "DTIM — UPC." [En línia]. Disponible a: https://www.essi.upc.edu/dtim/

[2] "¿Cuántos datos se producen en un minuto?" [En línia]. Disponible a: https://business-intelligence.grupobit.net/blog/cuantos-datos-se-producen-en-un-minuto

[3] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, *The End of an Architectural Era (It's Time for a Complete Rewrite)*, 2007. [En línia]. Disponible a: http://www.vertica.com

[4] D. Reinsel, J. Gantz, and J. Rydning, "Data Age 2025: The Evolution of Data to Life-Critical Don't Focus on Big Data; Focus on the Data That's Big Sponsored by Seagate The Evolution of Data to Life-Critical Don't Focus on Big Data; Focus on the Data That's Big," Tech. Rep., 2017. [En línia]. Disponible a: www.idc.com

[5] "Building a data economy - Brochure — Shaping Europe's digital future." [En línia]. Disponible a: https://ec.europa.eu/digital-single-market/en/news/building-data-economy-brochure

[6] "A comparison of data processing frameworks – Kapernikov." [En línia]. Disponible a: https://kapernikov.com/a-comparison-of-data-processing-frameworks/

[7] "File system - Wikipedia." [En línia]. Disponible a: https://en.wikipedia.org/wiki/File_system

[8] "What is HDFS? Apache Hadoop Distributed File System — IBM." [En línia]. Disponible a: https://www.ibm.com/analytics/hadoop/hdfs

[9] "Apache Spark™ - Unified Analytics Engine for Big Data." [En línia]. Disponible a: https://spark.apache.org/

[10] "¿Qué es Hadoop MapReduce? Introducción — Aprender BIG DATA." [En línia]. Disponible a: https://aprenderbigdata.com/hadoop-mapreduce/

[11] "Map Reduce and Stream Processing - DZone Big Data." [En línia]. Disponible a: https://dzone.com/articles/map-reduce-and-stream

[12] "Spark wins Daytona Gray Sort 100TB Benchmark — Apache Spark." [En línia]. Disponible a: https://spark.apache.org/news/spark-wins-daytona-gray-sort-100tb-benchmark.html

[13] L. Liu, “PERFORMANCE COMPARISON BY RUNNING BENCHMARKS ON HADOOP, SPARK, AND HAMR,” Tech. Rep.

[14] “Unidades de grabación magnética asistida por calor (HAMR) — Seagate España.” [En línia]. Disponible a: https://www.seagate.com/es/es/innovation/hamr/#

[15] “Hadoop vs Spark: A Head to Head Comparison in 2021 [Updated].” [En línia]. Disponible a: https://hackr.io/blog/hadoop-vs-spark

[16] H. Benbrahim, H. Hachimi, and A. Amine, “Comparison between Hadoop and Spark,” Tech. Rep.

[17] “Apache Spark — HG Insights.” [En línia]. Disponible a: https://discovery.hgdata.com/product/apache-spark

[18] “Apache Hadoop — HG Insights.” [En línia]. Disponible a: https://discovery.hgdata.com/product/apache-hadoop

[19] “A comparison of data processing frameworks – Kapernikov.” [En línia]. Disponible a: https://kapernikov.com/a-comparison-of-data-processing-frameworks/

[20] “Apache Spark VS Hadoop Map Reduce — OpenWebinars.” [En línia]. Disponible a: https://openwebinars.net/blog/apache-spark-vs-hadoop-map-reduce/

[21] “Principios SOLID con ejemplos.” [En línia]. Disponible a: https://enmilocalfunciona.io/principios-solid/

[22] “Metodología Kanban — Kanban Tool.” [En línia]. Disponible a: https://kanbantool.com/es/metodologia-kanban

[23] “Git - git-rebase Documentation.” [En línia]. Disponible a: https://git-scm.com/docs/git-rebase

[24] “BOE.es - BOE-A-1999-23750 Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal.” [En línia]. Disponible a: https://www.boe.es/eli/es/lo/1999/12/13/15/con

[25] “¿Qué es la LOPD o Protección de datos?¿Para qué sirve? — DCD.” [En línia]. Disponible a: https://www.dcd.es/que-es-la-lopd-proteccion-de-datos-para-que-sirve/

[26] “La protección de datos en la UE — Comisión Europea.” [En línia]. Disponible a: https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_es

[27] “LOPD vs GDPR principales cambios a tener en cuenta — Cad&Lan.” [En línia]. Disponible a: https://www.cadlan.com/noticias/lopd-vs-gdpr-principales-cambios-a-tener-en-cuenta/

[28] “”El dilema de las redes sociales”de Netflix: 5 secretos de los dueños de las redes para engancharnos y manipularnos, según el documental - BBC News Mundo.” [En línia]. Disponible a: https://www.bbc.com/mundo/noticias-54385775

[29] "What is Emergent Architecture? - Simplicable." [En línia]. Disponible a: https://simplicable.com/new/emergent-architecture

[30] "Agile Architecture — the rise of messy, inconsistent and emergent architecture — Hacker Noon." [En línia]. Disponible a: https://hackernoon.com/agile-architecture-the-rise-of-messy-inconsistent-and-emergent-architecture-e6801ab25b61

[31] "SOLID: los 5 principios que te ayudarán a desarrollar software de calidad." [En línia]. Disponible a: https://profile.es/blog/principios-solid-desarrollo-software-calidad/

[32] "Todo sobre la deuda técnica." [En línia]. Disponible a: https://www.javiergarzas.com/2012/11/deuda-tecnica-2.html

[33] "What is Three-Tier Architecture — IBM." [En línia]. Disponible a: https://www.ibm.com/cloud/learn/three-tier-architecture

[34] "Entity—Boundary—Interactor — Entity–Boundary–Interactor 0.1.0-alpha documentation." [En línia]. Disponible a: https://ebi.readthedocs.io/en/latest/

[35] "The Principles of Clean Architecture by Uncle Bob Martin - YouTube." [En línia]. Disponible a: https://www.youtube.com/watch?v=o_TH-Y78tt4

[36] "Spark - Difference between Cache and Persist? — SparkByExamples." [En línia]. Disponible a: https://sparkbyexamples.com/spark/spark-difference-between-cache-and-persist/

[37] "DatasetsProfiler/Ad_click_on_taobao.csv at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/raw_excels/whole_datasets/Ad_click_on_taobao.csv

[38] "HP Z4 G4 3,6 GHz Intel® Core™ Serie X i7-7820X Negro Mini Tower Puesto de Trabajo - Ordenador de sobremesa (3,6 GHz, Intel® Core™ Serie X, 32 GB, 4256 GB, DVD-RW, Windows 10 Pro): Amazon.es: Informática." [En línia]. Disponible a: http://amzn.to/3lbECqQ

[39] "pyspark.sql.DataFrame.describe — PySpark 3.1.2 documentation." [En línia]. Disponible a: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.describe.html

[40] "Error absoluto y error relativo: Qué son y cómo se calculan." [En línia]. Disponible a: https://ekuatio.com/error-absolutos-y-error-relativos-que-son-y-como-se-calculan/

[41] A. Abelló and E. Rodríguez, "Selectivity factor of a Selection (I)," in *Phys. Optim. (Intermediate results, Join)*, p. 14.

[42] "hadoop - Avro vs. Parquet - Stack Overflow." [En línia]. Disponible a: https://stackoverflow.com/questions/28957291/avro-vs-parquet

[43] "What is Memory Swapping? — Enterprise Storage Forum." [En línia]. Disponible a: https://www.enterprisestorageforum.com/hardware/what-is-memory-swapping/

[44] “Different CPU Times: Unix/Linux ‘top’ - DZone Performance.” [En línia]. Disponible a: https://dzone.com/articles/different-cpu-times-unixlinux-top

[45] “linux - How can I monitor disk I/O in a particular directory? - Unix and Linux Stack Exchange.” [En línia]. Disponible a: https://unix.stackexchange.com/questions/9520/how-can-i-monitor-disk-i-o-in-a-particular-directory

[46] “File-based dynamically-allocated hard-disk on Linux - Super User.” [En línia]. Disponible a: https://superuser.com/questions/1096549/file-based-dynamically-allocated-hard-disk-on-linux

[47] “Qué es la relación señal-ruido o SNR en audio y para qué sirve.” [En línia]. Disponible a: https://hardzone.es/reportajes/que-es/relacion-senal-ruido-snr-audio/

[48] “Multimap in C++ Standard Template Library (STL) - GeeksforGeeks.” [En línia]. Disponible a: https://www.geeksforgeeks.org/multimap-associative-containers-the-c-standard-template-library-stl/

[49] “Topological Sort — CodePath Android Cliffnotes.” [En línia]. Disponible a: https://guides.codepath.com/compsci/Topological-Sort

[50] “Breadth First Search or BFS for a Graph - GeeksforGeeks.” [En línia]. Disponible a: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[51] “Tree Traversals (Inorder, Preorder and Postorder) - GeeksforGeeks.” [En línia]. Disponible a: https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

[52] “Stack safe recursion in Java - Manning.” [En línia]. Disponible a: https://freecontent.manning.com/stack-safe-recursion-in-java/

[53] “Read and Write Avro files using Spark DataFrame — SparkByExamples.” [En línia]. Disponible a: https://sparkbyexamples.com/spark/read-write-avro-file-spark-dataframe/#write-avro-partition

[54] “Topological Sorting - GeeksforGeeks.” [En línia]. Disponible a: https://www.geeksforgeeks.org/topological-sorting/

[55] “Methods of Search.” [En línia]. Disponible a: http://www.cse.unsw.edu.au/$\sim$billw/Justsearch.html

[56] “Apache Parquet.” [En línia]. Disponible a: https://parquet.apache.org/documentation/latest/

[57] “Parquet three encodings means? - Stack Overflow.” [En línia]. Disponible a: https://stackoverflow.com/questions/52068926/parquet-three-encodings-means

[58] “Composition operation - Learning Functional Programming in Go.” [En línia]. Disponible a: https://subscription.packtpub.com/book/application_development/9781787281394/9/ch09lvl1sec63/composition-operation

[59] “Prometheus - Monitoring system and time series database.” [En línia]. Disponible a: https://prometheus.io/

[60] “An introduction to Prometheus metrics and performance monitoring — Enable Sysadmin.” [En línia]. Disponible a: https://www.redhat.com/sysadmin/introduction-prometheus-metrics-and-performance-monitoring

[61] “Time Series Database (TSDB) Explained — InfluxDB — InfluxData.” [En línia]. Disponible a: https://www.influxdata.com/time-series-database/

[62] “prometheus/node exporter: Exporter for machine metrics.” [En línia]. Disponible a: https://github.com/prometheus/node_exporter

[63] “Querying basics — Prometheus.” [En línia]. Disponible a: https://prometheus.io/docs/prometheus/latest/querying/basics/

[64] “Uso del comando iostat con ejemplos en linux.” [En línia]. Disponible a: https://www.sololinux.es/uso-del-comando-iostat-con-ejemplos-en-linux/

[65] “Issue with Sheet name having more than 31 characters - Aspose.Cells Product Family - Free Support Forum - aspose.com.” [En línia]. Disponible a: https://forum.aspose.com/t/issue-with-sheet-name-having-more-than-31-characters/85128

[66] “PySpark Documentation — PySpark 3.1.2 documentation.” [En línia]. Disponible a: https://spark.apache.org/docs/latest/api/python/

[67] “Writing An Hadoop MapReduce Program In Python.” [En línia]. Disponible a: https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

[68] “Python vs Java: What’s The Difference? – BMC Software — Blogs.” [En línia]. Disponible a: https://www.bmc.com/blogs/python-vs-java/

[69] “El compilador JIT - Documentación de IBM.” [En línia]. Disponible a: https://www.ibm.com/docs/es/sdk-java-technology/8?topic=reference-jit-compiler

[70] “¿Qué es Hadoop Yarn? Introducción — Aprender BIG DATA desde cero.” [En línia]. Disponible a: https://aprenderbigdata.com/hadoop-yarn/

[71] “The Java 8 Stream API Tutorial — Baeldung.” [En línia]. Disponible a: https://www.baeldung.com/java-8-streams

[72] “Functional Programming in Java — Baeldung.” [En línia]. Disponible a: https://www.baeldung.com/java-functional-programming

[73] “What is a Spring Bean? — Baeldung.” [En línia]. Disponible a: https://www.baeldung.com/spring-bean

[74] “java - Why use/develop Guice, when You have Spring and Dagger? - Stack Overflow.” [En línia]. Disponible a: https://stackoverflow.com/questions/39688830/why-use-develop-guice-when-you-have-spring-and-dagger

[75] “Significado de Experimento (Qué es, Concepto y Definición) - Significados.” [En línia]. Disponible a: https://www.significados.com/experimento/

[76] J. D. Ullman, “Experiments as Research Validation-Have We Gone too Far?” Tech. Rep., 2013. [En línia]. Disponible a: http://feedproxy.google.com/$\sim$r/daniel-lemire/atom/$\sim$3/4bht7t0oFZc/

[77] “/rdlab – Engaged with your research.” [En línia]. Disponible a: https://rdlab.cs.upc.edu/

[78] “Apache Hadoop 3.1.2 – Hadoop: YARN Resource Configuration.” [En línia]. Disponible a: https://hadoop.apache.org/docs/r3.1.2/hadoop-yarn/hadoop-yarn-site/ResourceModel.html

[79] “DTIM — UPC.” [En línia]. Disponible a: https://www.essi.upc.edu/dtim/blog/post/tips-and-tricks-on-writing-experiments

[80] “Configuration - Spark 2.3.0 Documentation.” [En línia]. Disponible a: https://spark.apache.org/docs/2.3.0/configuration.html

[81] “SQL Injection.” [En línia]. Disponible a: https://www.w3schools.com/sql/sql_injection.asp

[82] “georgeboc/DatasetsProfiler: Project used to profile a dataset.” [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler

[83] “georgeboc/Testbed: Project used to execute multiple experiments to compare Spark vs MapReduce.” [En línia]. Disponible a: https://github.com/georgeboc/Testbed

[84] “¿Qué es un bug?” [En línia]. Disponible a: https://www.abc.es/tecnologia/consultorio/20150226/abci--201502252129.html?ref=https:%2F%2Fwww.qwant.com%2F

[85] “Asus R510VX-DM528T Intel Core i7-7700HQ/8GB/1TB/GTX950M/15.6 PcComponentes.com.” [En línia]. Disponible a: http://bit.ly/2PXBUtn

[86] “Sueldo: Software Manager en Barcelona — Glassdoor.” [En línia]. Disponible a: https://www.glassdoor.es/Sueldos/barcelona-software-manager-sueldo-SRCH_IL.0,9_IM1015_KO10,26.htm?clickSource=searchBtn

[87] “Sueldo: Junior Software Architect en Barcelona — Glassdoor.” [En línia]. Disponible a: https://www.glassdoor.es/Sueldos/barcelona-junior-software-architect-sueldo-SRCH_IL.0,9_IM1015_KO10,35.htm?clickSource=searchBtn

[88] “Sueldo: Junior Software Engineer en Barcelona — Glassdoor.” [En línia]. Disponible a: https://www.glassdoor.es/Salaries/barcelona-junior-software-engineer-salary-SRCH_IL.0,9_IM1015_KO10,34.htm?countryRedirect=true

[89] "Sueldo: Junior Researcher en Barcelona — Glassdoor." [En línia]. Disponible a: https://www.glassdoor.es/Sueldos/barcelona-junior-researcher-sueldo-SRCH_IL.0,9_IM1015_KO10,27.htm?clickSource=searchBtn

[90] "Sueldo: Junior Analyst en Barcelona — Glassdoor." [En línia]. Disponible a: https://www.glassdoor.es/Sueldos/barcelona-junior-analyst-sueldo-SRCH_IL.0,9_IM1015_KO10,24.htm?clickSource=searchBtn

[91] "Sueldo: Junior QA Tester en Barcelona — Glassdoor." [En línia]. Disponible a: https://www.glassdoor.es/Sueldos/barcelona-junior-qa-tester-sueldo-SRCH_IL.0,9_IM1015_KO10,26.htm?clickSource=searchBtn

[92] "Sueldos de la empresa — Glassdoor.es." [En línia]. Disponible a: https://www.glassdoor.es/Sueldos/index.htm

[93] "AWS Pricing Calculator." [En línia]. Disponible a: https://calculator.aws/#/createCalculator/EC2

[94] "Precio kWh España 2021: precio diario, evolución y comparativa." [En línia]. Disponible a: https://tarifaluzhora.es/info/precio-kwh

[95] "How Much Energy Do My Appliances Use? INFOGRAPHIC." [En línia]. Disponible a: https://smarterbusiness.co.uk/blogs/how-much-energy-do-my-appliances-use-infographic/

[96] "Movistar: Internet, Móvil, TV y ¡Ofertas exclusivas!" [En línia]. Disponible a: https://www.movistar.es/

[97] "La oferta de habitaciones en alquiler se dispara en un 20Noticias — Marketing — TicPymes." [En línia]. Disponible a: https://www.ticpymes.es/marketing/noticias/1121816049304/oferta-de-habitaciones-alquiler-se-dispara-20-espana.1.html

[98] "Top 5 Apache Spark Use Cases." [En línia]. Disponible a: https://www.dezyre.com/article/top-5-apache-spark-use-cases/271

[99] "Big Data For Nonprofits — Data Analytics For Nonprofits — PostFunnel." [En línia]. Disponible a: https://postfunnel.com/how-big-data-helps-nonprofits-thrive/

[100] "Overview & Mission — YMCA of Austin." [En línia]. Disponible a: https://www.austinymca.org/about

[101] "Apache Spark Overview — Looker." [En línia]. Disponible a: https://looker.com/databases/apache-spark

[102] "Som Energia — La Cooperativa de Energia Verde." [En línia]. Disponible a: https://www.somenergia.coop/es/

[103] "Así utilizan el 'big data' Cruz Roja o MSF para salvar y mejorar vidas." [En línia]. Disponible a: https://www.expansion.com/economia-digital/2019/04/20/5cbb4461ca47416f088b45d3.html

[104] J. Orlowski, "The Social Dilemma," 2020. [En línia]. Disponible a: https://www.netflix.com/es-en/title/81254224

[105] "¿Qué es un log? – Ankama." [En línia]. Disponible a: https://support.ankama.com/hc/es/articles/203790076--Qu{\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{e\global\mathchardef\accent@spacefactor\spacefactor}\let\begingroup\def{}\endgroup\relax\let\ignorespaces\relax\accent19e\egroup\spacefactor\accent@spacefactor}-es-un-log-

[106] "Qué es un log, para qué sirve y algunos de los principales tipos de log." [En línia]. Disponible a: https://pandorafms.com/blog/es/logs/

[107] "DevOps Observability, Analysis and Insight — Splunk." [En línia]. Disponible a: https://www.splunk.com/en_us/devops.html

[108] "¿Qué es los Amazon CloudWatch Logs? - Amazon CloudWatch Logs." [En línia]. Disponible a: https://docs.aws.amazon.com/es_es/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html

[109] "logpai/loghub: A large collection of system log datasets for AI-powered log analytics." [En línia]. Disponible a: https://github.com/logpai/loghub

[110] "Kaggle: Your Home for Data Science." [En línia]. Disponible a: https://www.kaggle.com/

[111] "SEC.gov — EDGAR Log File Data Set." [En línia]. Disponible a: https://www.sec.gov/dera/data/edgar-log-file-data-set.html

[112] "Exploratory Data Analysis Tutorial: Data Profiling - DataCamp." [En línia]. Disponible a: https://www.datacamp.com/community/tutorials/python-data-profiling

[113] "Descriptive Statistics Definition." [En línia]. Disponible a: https://www.investopedia.com/terms/d/descriptive_statistics.asp

[114] "Third Normal Form (3NF) - GeeksforGeeks." [En línia]. Disponible a: https://www.geeksforgeeks.org/third-normal-form-3nf/

[115] "Calculadora en línea: Entropía de Shannon." [En línia]. Disponible a: https://es.planetcalc.com/2476/

[116] "Why is Entropy maximised when the probability distribution is uniform? - Cross Validated." [En línia]. Disponible a: https://stats.stackexchange.com/questions/66108/why-is-entropy-maximised-when-the-probability-distribution-is-uniform

[117] "DatasetsProfiler/datasets_profiling_information at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/tree/main/datasets_profiling_information

[118] "DatasetsProfiler/datasets_profiling_information/parameters at main · georgeboc/-DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/tree/main/datasets_profiling_information/parameters

[119] "DatasetsProfiler/datasets_profiling_information/raw_excels at main · georgeboc/-DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/tree/main/datasets_profiling_information/raw_excels

[120] "Ad Display/Click Data on Taobao.com — Kaggle." [En línia]. Disponible a: https://www.kaggle.com/pavansanagapati/ad-displayclick-data-on-taobaocom

[121] "DatasetsProfiler/Ad_click_on_taobao.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Ad_click_on_taobao.ods

[122] "DatasetsProfiler/Ad_click_on_taobao_Ad_feature.ods at main · georgeboc/-DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Ad_click_on_taobao_Ad_feature.ods

[123] "DatasetsProfiler/Ad_click_on_taobao_User_profile.ods at main · georgeboc/-DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Ad_click_on_taobao_User_profile.ods

[124] "loghub/Andriod at master · logpai/loghub." [En línia]. Disponible a: https://github.com/logpai/loghub/tree/master/Andriod

[125] "DatasetsProfiler/Android.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Android.ods

[126] "loghub/BGL at master · logpai/loghub." [En línia]. Disponible a: https://github.com/logpai/loghub/tree/master/BGL

[127] "DatasetsProfiler/BGL.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/BGL.ods

[128] "DatasetsProfiler/Edgar_samples.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Edgar_samples.ods

[129] "loghub/HDFS at master · logpai/loghub." [En línia]. Disponible a: https://github.com/logpai/loghub/tree/master/HDFS

[130] "DatasetsProfiler/HDFS_1.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/HDFS_1.ods

[131] "DatasetsProfiler/HDFS_2.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/HDFS_2.ods

[132] "MOOC Dataset — Kaggle." [En línia]. Disponible a: https://www.kaggle.com/samyakjhaveri/mooc-final

[133] "DatasetsProfiler/MOOC.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/MOOC.ods

[134] "Enhanced Reader." [En línia]. Disponible a: moz-extension://45a607ec-6f06-46ca-bd0f-41b5d7d17e1f/enhanced-reader.html?openApp&pdf=https%3A%2F%2Fs3.amazonaws.com%2Fstorage.citizensforethics.org%2Fwp-content%2Fuploads%2F2017%2F04%2F08153657%2F2019-1-7-40-JA.pdf

[135] "Obama Visitor Logs — Kaggle." [En línia]. Disponible a: https://www.kaggle.com/jayrav13/obama-visitor-logs?select=whlogs_4.csv

[136] "DatasetsProfiler/Obama_visitor_logs.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Obama_visitor_logs.ods

[137] "Recommender Click Logs- Sowiport — Kaggle." [En línia]. Disponible a: https://www.kaggle.com/sohier/recommender-click-logs-sowiport?select=full-data.csv

[138] "DatasetsProfiler/Recommender_click_logs_sowiport.ods at main · georgeboc/-DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Recommender_click_logs_sowiport.ods

[139] "Seattle COBAN Logs — Kaggle." [En línia]. Disponible a: https://www.kaggle.com/city-of-seattle/seattle-coban-logs?select=coban-logs.csv

[140] "DatasetsProfiler/Seattle_coban_logs.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Seattle_coban_logs.ods

[141] "loghub/Spark at master · logpai/loghub." [En línia]. Disponible a: https://github.com/logpai/loghub/tree/master/Spark

[142] "DatasetsProfiler/Spark.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Spark.ods

[143] "loghub/Thunderbird at master · logpai/loghub." [En línia]. Disponible a: https://github.com/logpai/loghub/tree/master/Thunderbird

[144] "DatasetsProfiler/Thunderbird.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Thunderbird.ods

[145] "Ubuntu Dialogue Corpus — Kaggle." [En línia]. Disponible a: https://www.kaggle.com/rtatman/ubuntu-dialogue-corpus?select=toc.csv

[146] "DatasetsProfiler/Ubuntu_dialogue_corpus.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Ubuntu_dialogue_corpus.ods

[147] "User Logs V2 — Kaggle." [En línia]. Disponible a: https://www.kaggle.com/jameschartouni/user-logs-v2

[148] "DatasetsProfiler/User_logs_v2.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/User_logs_v2.ods

[149] "loghub/Windows at master · logpai/loghub." [En línia]. Disponible a: https://github.com/logpai/loghub/tree/master/Windows

[150] "DatasetsProfiler/Windows.ods at main · georgeboc/DatasetsProfiler." [En línia]. Disponible a: https://github.com/georgeboc/DatasetsProfiler/blob/main/datasets_profiling_information/processed_excels/Windows.ods

[151] "hadoop - How to check the distributed data over hdfs - Stack Overflow." [En línia]. Disponible a: https://stackoverflow.com/questions/16966680/how-to-check-the-distributed-data-over-hdfs

[152] "java - Apache Spark – using spark-submit throws a NoSuchMethodError - Stack Overflow." [En línia]. Disponible a: https://stackoverflow.com/questions/42398720/apache-spark-using-spark-submit-throws-a-nosuchmethoderror

[153] "Testbed/SelectSparkOperation.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/spark/SelectSparkOperation.java

[154] "Testbed/ProjectSparkOperation.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/spark/ProjectSparkOperation.java

[155] "Testbed/UnionSparkOperation.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/spark/UnionSparkOperation.java

[156] "Spark DataFrame Union and UnionAll — SparkByExamples." [En línia]. Disponible a: https://sparkbyexamples.com/spark/spark-dataframe-union-and-union-all/

[157] "Testbed/AggregateSparkOperation.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/spark/AggregateSparkOperation.java

[158] "Testbed/GroupbySparkOperation.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/spark/GroupbySparkOperation.java

[159] "Testbed/JoinSparkOperation.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/spark/JoinSparkOperation.java

[160] "Testbed/SelectJar.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/mapReduce/select/SelectJar.java

[161] "Testbed/ProjectJar.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/mapReduce/project/ProjectJar.java

[162] A. Abelló and O. Romero, "MapReduce," p. 15, 2019.

[163] "Testbed/UnionJar.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/mapReduce/union/UnionJar.java

[164] "Testbed/SumAggregateJar.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/mapReduce/sumAggregator/SumAggregateJar.java

[165] "Testbed/GroupByJar.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/mapReduce/groupBy/GroupByJar.java

[166] "Testbed/JoinJar.java at main · georgeboc/Testbed." [En línia]. Disponible a: https://github.com/georgeboc/Testbed/blob/main/src/main/java/com/testbed/boundary/invocations/frameworks/mapReduce/join/JoinJar.java

Apèndixs

Apèndix A

Selecció dels *Datasets*

En aquest capítol ens centrarem a explicar el domini que utilitzarem com a entrada dels dos *frameworks* per a poder mesurar el rendiment, el procediment seguit per a seleccionar els *datasets* més rellevants, i finalitzarem amb una classificació que serà la que utilitzarem durant els experiments com a entrades.

A.1 Domini

El domini seleccionat per als *datasets* són els *logs*. Els *logs* o registres, són arxius de text en els quals consten de forma cronològica els esdeveniments que han afectat un sistema informàtic, sigui un programa, aplicació, servidor, etc., i el conjunt de canvis que aquests hagin generat sobre el sistema [105].

Aquests són una eina imprescindible per als sistemes en producció, ja que ens permet saber en tot moment què està fent el nostre sistema i, ens permet identificar *bugs* que no estiguin coberts per tests unitaris o d'integritat, mentre el sistema s'estigui executant. Els *logs* també són importants tant des del punt de vista de la seguretat, com de l'anàlisi de l'ús del sistema o del rendiment [106].

Una característica fonamental dels *logs* és que aquests no estan estructurats d'una forma universal, és a dir, que no tots segueixen la mateixa forma, encara que sí que segueixen uns determinats patrons. Tant és així que els *logs* a vegades són considerats l'aliment preferit de les tècniques de *Machine Learning* per a detectar patrons que ajudin a prendre decisions [106].

Actualment, existeixen diferents tecnologies que serveixen per a gestionar els *logs* de forma centralitzada; la més coneguda mundialment és Splunk [107]. Tanta és la importància dels *logs*, que han sorgit serveis al núvol com per exemple Amazon CloudWatch Logs, que permeten fer la centralització dels *logs* de tots els sistemes, aplicacions i serveis d'AWS que s'utilitzin [108].

Aquests sistemes tenen un fort component de selecció dels *logs*, per a poder trobar i analitzar els errors que hagin sorgit, també hi ha agregacions, joins i altres operacions d'àlgebra relacional que es fan servir per a aquest tipus de dades.

És per aquests motius que hem considerat que els *logs* serien un candidat interessant en la nostra anàlisi dels dos *frameworks*.

A.2 Recerca dels *Datasets*

Per a poder trobar *datasets*, ens hem hagut de centrar en *logs* que es trobaven de forma pública a internet. Cal destacar que si bé els *logs* exposen els errors que hagi ocorregut durant l'execució d'un sistema o aplicació, també poden exposar informació crítica o bé del sistema o de dades personals. Per a evitar exposar dades sensibles a internet, moltes empreses mantenen la confidencialitat dels *logs*, disminuint així la quantitat d'informació que puguem trobar a Internet.

De totes maneres, vam trobar el repositori *log*hub [109] a GitHub amb conjunts de dades de logs, de diverses aplicacions i sistemes. Tots aquests conjunts de dades, no superaven cap els 100 GB. Aleshores, vam optar per continuar cercant per a obtenir un ventall més extens de conjunts de dades i no limitar-nos a una única font. En aquest repositori, tots els *datasets* es troben en format cru, és a dir, de la mateixa forma que es generen, en un arxiu de text, i sense haver sigut processats per a extreure una estructura.

Vam trobar també la pàgina web Kaggle [110], on es poden trobar molts *datasets* que s'utilitzen per a *Machine Learning*. Hi havia diversos conjunts de dades de *logs*, encara que com aquesta pàgina web està dedicada a fer competicions en el camp del *Machine Learning*, els conjunts de dades ja venien amb una estructura predefinida. En aquest lloc web, podem destacar que no hem trobat cap *log* com en el repositori [109], tots els que vam trobar, estaven en format CSV i les etiquetes de tots els camps eren conegudes, al contrari que passava amb els conjunts de dades del repositori, on alguns camps no es podia interpretar correctament la informació que aportaven.

Tots els *datasets* que hem obtingut fins aquest moment, tenien menys de 100 GB de grandària de forma descomprimida. Aleshores, el nostre objectiu, en aquest punt era trobar un conjunt de dades d'una grandària superior a centenars de gigabytes, preferiblement de l'ordre d'un terabyte per a poder utilizar-lo com a conjunt de dades gran. Aleshores, vam trobar el *dataset* Edgar [111], que tenia una mida d'aproximadament 1,4 TB de forma descomprimida, de *logs* generats per un sistema software.

De totes aquestes fonts de dades, vam fer una col·lecció i les vam analitzar de forma sistemàtica per a poder determinar quines serien les més interessants per a la nostra anàlisi. El procediment que hem utilitzat ha sigut crear un sistema *software* que permetés fer un perfil estadístic del conjunt de dades per tal de poder comprar-los entre ells només mirant els perfils dels *datasets*. El sistema *software*, anomenat *Datasets Profiler*, es detalla més en profunditat el seu funcionament i arquitectura en el capítol 3.

A.3 El perfil estadístic

Per a analitzar els conjunts de dades, necessitem una forma d'encarar aquesta anàlisi. Aquesta forma rep el nom de *Exploratory Data Analysis* (EDA), i se centra a descobrir i

resumir un conjunt de dades, explorant l'estructura d'aquest, les seves variables i relacions [112].

Un aspecte d'aquesta anàlisi exploratòria de les dades, és el *Data Profiling*, que consisteix a resumir el conjunt de dades a través d'estadístics descriptius.

Un estadístic descriptiu, segons [113], és un coeficient que ofereix una breu descripció que resumeix un determinat conjunt de dades i pot ser una representació de tota la població o bé una mostra. Els estadístics descriptius estan segmentats en mesures de tendència central i mesures de variabilitat.

Les mesures de tendència central inclouen la mitjana, la mediana, la moda, mentre que les mesures de variabilitat inclouen la desviació estàndard, la variància, les variables mínimes i màximes, d'entre altres.

Aquests tipus d'estadístics ens poden ajudar a entendre les propietats col·lectives dels elements d'un conjunt de dades i ens poden donar una idea de la distribució de probabilitat o la forma de les dades [113].

L'objectiu de fer el perfil de les dades, segons [112], és tenir un enteniment sòlid de les dades per a poder començar posteriorment a fer consultes i visualitzar les dades de diferents formes.

Abans de fer el perfil estadístic, ens hauríem d'informar de l'origen de les dades que estem analitzant, i s'ha de tenir informació de com s'hi ha recollit, com es van processar i si han passat a través de transformacions [112]. El fet de no poder respondre a les anteriors incògnites farà difícil poder extreure conclusions.

Addicionalment, ens hauríem d'assegurar que el conjunt de dades està estructurat d'una forma estandarditzada. Tal com vam comentar anteriorment, per naturalesa, els *logs*, no tenen una estructura predefinida, per la qual cosa, imposarem una estructura a l'hora de parsejar les dades amb el nostre programa *Datasets Profiler*. Aleshores, tindrem les dades estructurades i amb uns tipus determinats, i podrem analitzar-les correctament.

La forma més recomanable de format és la Tercera Forma Normal. Una relació està en Tercera Forma Normal (3FN) [114], si no hi ha una dependència transitiva per a les claus candidates. Addicionalment, si per a cada dependència del tipus $X \rightarrow Y$, on X i Y són atributs, es mantenen les següents condicions:

- X és una clau primària
- Cada element de Y és part d'alguna clau candidata.

Addicionalment, haurem de fer èmfasi que per a complementar el perfil estadístic, inclourem el descriptor estadístic de l'entropia, ja que ens permet quantificar la "quantitat d'informació"que ens aporta una determinada columna d'un conjunt de dades.

Una definició més formal de l'entropia de Shannon, segons [115], és que l'entropia és una mesura de la incertesa en una variable aleatòria. Podríem entendre com a variable aleatòria la columna d'un conjunt de dades que estiguem analitzant. El valor de l'entropia de Shannon

es podria quantificar amb l'equació (A.1), obtenint així la quantitat d'informació continguda en un missatge:

$$H(X) = -\sum_{i=1}^{n} p(x_{\mathrm{i}}) \cdot log_{\mathrm{b}}(p(x_{\mathrm{i}})) \tag{A.1}$$

Això vol dir que l'entropia d'una font X que disposa de n possibles símbols, ve determinada pel sumatori de la probabilitat que cada símbol aparegui, multiplicat pel logaritme de la mateixa probabilitat i s'agafarà el valor negatiu, ja que els valors de la probabilitat són sempre entre 0 i 1. Per tant, el logaritme tindrà un valor negatiu sempre. Per tant, el valor de l'entropia serà sempre positiu.

La interpretació que es fa de l'entropia és la següent: per a una font A, que disposa d'un únic símbol, aleshores, la probabilitat que aquest símbol aparegui serà 1. Aleshores, l'entropia de la font A serà H(A) = 0. En canvi, si tenim una font B, on tenim n diferents símbols equiprobables, aleshores, la probabilitat serà $1/n$ per a tots els símbols i l'entropia serà màxima [116]. Per tant, l'entropia serà més gran, com més desconeixem la font d'informació i això vol dir que la font ens aportarà el màxim d'informació.

En conclusió, aquest estudi del perfil estadístic ens serà l'eina que permetrà comparar els diferents conjunts de dades i ens permetrà establir quins són més interessants que altres per a poder utilitzar-los posteriorment per als nostres experiments.

A.4 Anàlisi dels *Datasets*

Primer, hem agafat els conjunt de dades especificats a repositori de codi, en [117], els hem descarregat i, utilitzant els paràmetres especificats a [118], vam crear els perfils estadístics utilitzant el sistema descrit en el capítol 3 de totes les mostres i dels conjunts de dades sencers. Els resultats de l'execució es mostren en [119].

Segons l'exposat amb anterioritat, anirem analitzant cada conjunt de dades i anirem identificant els diferents factors que ens permetran determinar si el conjunt de dades és interessant o no per als nostres experiments.

Primer, començarem amb una descripció dels *datasets*; seguidament exposarem l'esquema que segueixen – l'esquema especificat en el *dataset* o bé l'esquema que hem inferit nosaltres – i, finalment, farem l'anàlisi dels resultats d'executar el programa que fa un perfil estadístic del *dataset*.

Els *datasets* els hem analitzat prenent mostres primer i, si era possible també hem fet una anàlisi del *dataset* complet. Les mostres són bàsicament analitzar les primeres 1000 files, i anar incrementant exponencialment aquesta mida de la mostra multiplicant per 10 cada cop la quantitat de files analitzades començant per l'inici de conjunt de dades. Això ho hem fet només per als conjunts de dades grans, i no per als conjunts de dades auxiliars, ja que aquests tenien una mida molt més reduïda, i no calia fer un mostreig; per a aquests conjunts de dades, hem fet el càlcul dels descriptors per al conjunt de dades complet.

Cal recalcar que tot i que per a alguns conjunts de dades no calia per a un parseig per a poder deixar fora aquelles files que no seguien un determinat format, nosaltres hem decidit fer passar per aquest procés a tots els conjunts de dades. Això ens permetia verificar que la transformació que s'hagi fet sobre el conjunt de dades era correcta i ens evitava tenir problemes en els resultats de la nostra anàlisi.

En l'anàlisi que mostrarem a continuació, explicarem els descriptors estadístics més rellevants i els posarem entre parèntesis per a poder comparar-los amb els valors existents en els excels.

A.4.1 Ad Display/Click Data on Taobao.com

Descripció

Aquests *logs* s'han obtingut prenent mostres aleatòries d'1.140.000 usuaris del lloc web de Taobao durant 8 dies dels *logs* dels clics als anuncis publicitats. Hi ha 26 milions de files que formen l'arxiu de *log*. Aquest conjunt de dades està acompanyat d'uns altres dos conjunts de dades auxiliars de grandàries molt més reduïdes, que fan join amb el conjunt de dades principal.

Naturalment, aquests *logs* estan parsejats, és a dir transformats, i preprocessats prèviament per a ser posats en un format CSV, és a dir que no els trobem en un format cru. S'han transformat per a poder col·locar el format cru en una taula en CSV amb etiquetes.

Segons [120], si utilitzem els camps User i Timestamp com a clau primària, hi haurà molts valors repetits. Això passa pel fet que el comportament dels diferents tipus de les dades col·leccionades pels diferents departaments que a l'hora d'empaquetar-se tots junts, hi havia petites desviacions. Per exemple, dos mateixos *timestamps* és possible que siguin lleugerament diferents, amb una petita diferència de temps entre els dos.

El conjunt de dades està estructurat d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades principal és d'1,04 GiB. Les altres dues fonts de dades, Ad Feature i User Profile, tenen respectivament 30,64 MiB i 23,95 MiB. Aquesta font s'ha extret de [120].

Esquemes

L'esquema d'aquesta font de dades principal es pot observar a la Taula 22.

L'esquema de la font de dades auxiliar Ad Feature es pot observar a la Taula 23.

L'esquema de la font de dades auxiliar User Profile es pot observar a la Taula 24.

El camp NewUserClassLevel cobreix el comportament pel que fa a compres al llarg dels 22 dies de tots els usuaris en el conjunt de dades principal.

Camp	Tipus	Descripció
User	Integer	Identificador de l'usuari
Timestamp	Timestamp	Instant de temps en el que s'ha creat el *log*
AdGroupId	Integer	Identificador del grup d'anuncis
PID	String	Escenari
Clk	Integer	El valor 1 vol dir clic i el valor 0 val dir que no s'ha fet clic
No Clk	Integer	És el valor de la columna Clk negat

Taula 22: Esquema del conjunt de dades principal de Ad Display/Click Data on Taobao.com [Font: adaptació a partir de [120]]

Camp	Tipus	Descripció
AdGroupId	Integer	Identificador del grup d'anuncis d'un producte
CategoryId	Integer	Identificador de la categoria del producte
CampaignId	Integer	Identificador de la campanya
Customer	Integer	Identificador del client
Brand	Integer	Identificador de la marca del producte
Price	Float	És el preu del producte

Taula 23: Esquema del conjunt de dades auxiliar Ad Feature [Font: adaptació a partir de [120]]

Camp	Tipus	Descripció
UserId	Integer	Identificador de l'usuari
CmsSegId	Integer	Identificador del CmsSeg
CmsGroupId	Integer	Identificador del grup de Cms
FinalGenderCode	Integer	El valor 1 vol dir sexe masculí i el valor 2, sexe femení
AgeLevel	Integer	Identificador del nivell d'edat
PValueLevel	Integer	Grau de consumició: 1 vol dir baix, 2 vol dir mitjà i 3 vol dir alt
ShoppingLevel	Integer	Intensitat de compra: 1 vol dir poc intens, 2 vol dir moderat i 3 vol dir molt intens
Occupation	Integer	És estudiant universitari? 1 vol dir sí i 0 vol dir no
NewUserClassLevel	Integer	Nivell de ciutat

Taula 24: Esquema del conjunt de dades auxiliar User Profile [Font: adaptació a partir de [120]]

Anàlisi

Podem observar en [121] que per a la columna User, amb les primeres mostres, la proporció de files diferents (Count distinct) respecte al nombre de files (Original rows count) és gran, encara que aquesta va disminuint considerablement cada cop que augmentem la mida de la mostra, fins a arribar al *dataset* complet, on podem observar que la proporció és petita.

Veiem que l'entropia (Entropy, Delta time in seconds entropy i Messages entropy) de les columnes User, DateTime i AdGroupId és elevada i això ho podem comprovar si ho mirem tant per a les mostres com per al conjunt de dades complet.

El fet de tenir diverses columnes amb una entropia alta i també dues taules auxiliars en [122, 123] per a fer les operacions de *join*, converteixen aquest conjunt de dades en una candidata forta per als nostres experiments.

A.4.2 Android

Descripció

Aquests *logs* es van generar amb un telèfon mòbil amb el sistema operatiu Android en un laboratori. Conté *logs* propis del sistema operatiu. Els *logs* són crus, és a dir, no han sigut sotmesos a cap transformació ni processament. El sistema operatiu Android els ha generat en un format de text i aquests *logs* són els que nosaltres estem processant.

Aquests *logs* no estan parsejats, ni tampoc tenen etiquetes per a les columnes. Per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és de 183,5 MiB. Aquesta font s'ha extret de [124].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 25.

Camp	Tipus	Descripció
DateTime	Timestamp	Instant en el que s'ha produït el *log*
PID	Integer	Identificador del procés que ha creat el *log*
TID	Integer	Identificador del *thread* que ha creat el *log*
Priority	String	Prioritat del missatge
Tag	String	És una etiqueta que descriu el mòdul del qual prové el *log*
Message	String	És el contingut del *log*. Expressa l'estat del sistema

Taula 25: Esquema del conjunt de dades Android
[Font: elaboració pròpia a partir del conjunt de dades a [124]]

Anàlisi

En els resultats processats a [125], veiem que el camp DateTime no conté cap especificació de l'any en el qual es van recollir les dades, aleshores, es va quedar la data per defecte, que en aquest cas és de l'any 1900.

Observem que les columnes DateTime i Message tenen una entropia (Delta time in seconds entropy i Messages entropy) més gran que la resta de columnes. També es pot observar que la columna DateTime té una quantitat de valors diferents (Count distinct) alta.

A.4.3 BGL

Descripció

BGL és un *dataset* col·leccionat pel sistema supercomputador BlueGene/L en Lawrence Livermore National Labs (LLNL) en Livermore, California. Els *logs* contenen missatges

d'alerta i no alerta identificats per etiquetes de categoria d'alerta. S'ha utilitzat en diversos estudis de parseig de *logs*, detecció d'anomalies i predicció de fallades. No s'ha aplicat cap transformació ni processament sobre el conjunt de dades.

Aquests *logs* no estan parsejats, ni tampoc tenen etiquetes per a les columnes. Per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és de 708,9 MiB. Aquesta font s'ha extret de [126].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 26.

Camp	Tipus	Descripció
IsAlertMessage?	Integer	El valor 1 indica que el *log* és un missatge d'alerta. El valor 0 indica que no ho és
Timestamp	Timestamp	Instant en el que es registra el *log*
Date	Timestamp	Data en la qual es registra el *log*
DateTime	Timestamp	Data i temps en la que es registra el *log*
Node	String	És el node del clúster des del qual es genera el *log*
Node Master	String	És el node màster
Type	String	Indica el tipus de *log*
Component	String	És el component des del qual es genera el *log*
Level	String	Indica el nivell de severitat del *log*
Content	String	És el contingut del *log*. Expressa l'estat del sistema

Taula 26: Esquema del conjunt de dades BGL
[Font: elaboració pròpia a partir del conjunt de dades a [126]]

Anàlisi

Observem en [127] que en aquest *dataset* les columnes que més entropia (Messages entropy i Timestamp entropy) tenen són la de Node, Timestamp i Content. Tot i això, es pot veure clarament que les columnes Type, Component i Level tenen una entropia (Messages entropy) d'al voltant de 0, que vol dir que quasi tots els valors es repeteixen.

A.4.4 Edgar

Descripció

La informació continguda té el format CSV extret d'arxius de *log* d'Apache que graven i emmagatzemen estadístiques d'accés dels usuaris per al lloc web SEC.gov, cobrint un període des del 14 de febrer de 2003 fins al 30 de juny de 2017. A causa de certes limitacions, inclòs l'existència d'arxius perduts o danyats, la informació recollida és possible que no hagi recollit completament tot el tràfic. També és possible l'existència d'imprecisions o d'altres errors introduïts en les dades durant el procés d'extracció d'aquestes. No s'ha aplicat cap transformació ni processament sobre el conjunt de dades.

Aquests *logs* no estan parsejats, és a dir que estan en format cru, però sí que tenen etiquetes per a les columnes. Les etiquetes estan en un arxiu README separat de la resta de dades.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és del voltant d'1,42 TiB. Per aquest motiu, per a aquesta anàlisi, hem utilitzat una mostra de les dades. Aquesta font s'ha extret de [111].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 27.

Camp	Tipus	Descripció
IP	String	És la IP ofuscada del node que ha generat el *log*
DateTime	Timestamp	Data i temps en la que es registra el *log*
Zone	Float	És la zona en la qual es registra el *log*
CIK	Float	És la clau d'índex central de SEC (Central Index Key, CIK) associada amb el document sol·licitat
Accession	String	Indica el nombre de consentiment del document SEC associat amb el document sol·licitat
Doc	String	És el nom de l'arxiu de l'arxiu sol·licitat incloent l'extensió
Code	Float	Indica el codi de l'estatus de l'arxiu de *log* per a la sol·licitud
Size	Float	Indica la grandària del document
IDX	Float	Té el valor d'1 si el sol·licitant ha arribat en la pàgina d'índex d'un conjunt de documents i 0 altrament
NoRefer	Float	Té el valor d'1 si el camp de referent de l'arxiu de *log* està buit i 0 altrament
NoAgent	Float	Té el valor d'1 si el camp de *user agent* de l'arxiu de *log* està buit i 0 altrament
Find	Float	Són valors numèrics de 0 a 10 que es corresponen a si una determinada cadena de text es va trobar en el camp del referent
Crawler	Float	Aquesta variable pren el valor d'1 si el *user agent* s'autoidentifica com un *crawler* de pàgines web o si té un codi d'usuari de 404
Browser	String	És una variable de tres caràcters que identifica el tipus de navegador utilitzat

Taula 27: Esquema del conjunt de dades Edgar
[Font: adaptat a partir de [111]]

Anàlisi

Podem observar en [128] que les columnes amb més entropia (Timestamp entropy i Messages entropy) són DataTime i Accession.

Cal destacar que aquest *dataset* és el més gran de tots, i segurament ens serveix per a la nostra anàlisi per a provar com funcionarien els dos *frameworks* de processament de dades amb volums grans d'informació.

A.4.5 HDFS 1

Descripció

Aquest conjunt de *logs* s'ha generat en un entorn privat de *cloud* amb HDFS utilitzant *workloads* per a *benchmarks*. No s'ha aplicat cap transformació ni processament sobre el conjunt de dades.

Aquests *logs* no estan parsejats, ni tampoc tenen etiquetes per a les columnes. Per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és d'1,5 GiB. Aquesta font s'ha extret de [129].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 28.

Camp	Tipus	Descripció
DateTime	Timestamp	Data i temps en la que es registra el *log*
Pid	Integer	Identificador del procés que ha generat el *log*
Level	String	Marca el nivell de severitat del *log*
Component	String	És el component que ha generat el *log*
Content	String	És el contingut del *log*. Expressa l'estat del sistema

Taula 28: Esquema del conjunt de dades HDFS 1
[Font: elaboració pròpia a partir del conjunt de dades a [129]]

Anàlisi

Per a aquest *dataset*, podem observar en els resultats a [130] que les columnes DateTime, Pid i Content són les que més entropia (Timestamp entropy, Entropy, Messages entropy) tenen. L'únic desavantatge que té aquest conjunt de dades és que té molt poques columnes.

A.4.6 HDFS 2

Descripció

El conjunt de *logs* es va recollir agregant *logs* d'un sistema HDFS en un laboratori en CUHK amb el propòsit d'investigar-los. Els *logs* són agregats en el nivell de node. Cal destacar que tres màquines del clúster han sigut reparades i alguns *logs* s'han perdut. No s'ha aplicat cap transformació ni processament sobre el conjunt de dades.

Aquests *logs* no estan parsejats, ni tampoc tenen etiquetes per a les columnes. Per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és de 17 GiB. Aquesta font s'ha extret de [129].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 29.

Camp	Tipus	Descripció
DateTime	Timestamp	Data i temps en la que es registra el *log*
Level	String	Marca el nivell de severitat del *log*
Component	String	És el component que ha generat el *log*
Content	String	És el contingut del *log*. Expressa l'estat del sistema

Taula 29: Esquema del conjunt de dades HDFS 2
[Font: elaboració pròpia a partir del conjunt de dades a [129]]

Anàlisi

Podem observar en [131] que el percentatge de files perdudes (Lost rows percentage) és alt. També veiem que les columnes que més entropia (Timestamp entropy i Messages entropy) tenen són la primera i l'última, corresponent a DateTime i Content.

Es descarten un 18,31 % de les files a l'hora de considerar el *dataset* complet, això converteix aquest conjunt de dades en el que més dades es descarten en passar pel *Datasets-Profiler*. Per tant, aquesta font de dades no sembla ser massa bona. Les línies descartades són missatges d'error que no segueixen cap format.

A.4.7 MOOC

Descripció

Aquest és un conjunt de dades que consisteix en *logs* d'una pàgina web que té més de 155.000 estudiants aprenent 247 cursos únics a través de cursos MOOC en una universitat xinesa.

Aquests *logs* estan parsejats, és a dir transformats, i preprocessats prèviament per a ser posats en un format CSV, és a dir que no els trobem en un format cru. S'han transformat per a poder col·locar el format cru en una taula en CSV amb etiquetes.

El conjunt de dades està estructurat d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

El *dataset* complet té 6,01 GiB i està dividit en un *dataset* per a entrenar i un *dataset* per a fer tests. Nosaltres hem analitzat només el *dataset* d'entrenament, que té un pes de 4,3 GiB. Aquesta font s'ha extret de [132].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 30.

Anàlisi

Per a aquest *dataset*, podem observar en [133] que només dues columnes tenen l'entropia (Messages entropy) més baixa que la resta: SessionId i Object. També podem observar que té 8 columnes i que les dades venen en un format fàcilment parsejable, en format CSV.

Camp	Tipus	Descripció
EnrollId	Integer	És l'identificador de la matrícula
Username	Integer	Indica el nom d'usuari
CourseId	String	És l'identificador del curs
SessionId	String	És l'identificador de la sessió
Action	String	Indica el tipus d'activitat de l'usuari
Object	String	Indica l'objecte corresponent a l'acció
DateTime	Timestamp	Indica l'instant de temps en el que s'ha registrat el *log*
Truth	Integer	El valor és 1 si l'alumne ha abandonat el curs i 0 altrament

Taula 30: Esquema del conjunt de dades MOOC
[Font: adaptació a partir de [132]]

A.4.8 Obama Visitor Logs

Descripció

Aquest conjunt de dades va ser cedit per la Casa Blanca i consisteix en al voltant de 5,9 milions de *logs* de visitants.

Aquests *logs* estan parsejats, és a dir transformats, i preprocessats prèviament per a ser posats en un format CSV, és a dir que no els trobem en un format cru. S'han transformat per a poder col·locar el format cru en una taula en CSV amb etiquetes. La descripció de les etiquetes no està acompanyada del conjunt de dades; aquesta s'ha extret de [134].

El conjunt de dades està estructurat d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

El *dataset* complet té 1,1 GiB i està dividit en 6 arxius d'1 milió de files cadascun. Aquesta font s'ha extret de [135].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 31.

Anàlisi

Podem observar en [136] que aquest *dataset* té moltes més columnes que la resta. Aquest, en concret, ens permetrà veure com es comportaran els dos *frameworks* de processament de dades per a moltes columnes.

A.4.9 Recommender Click Logs-Sowiport

Descripció

Aquest *dataset* conté 28 milions de recomanacions i parelles de clic/no clic dels usuaris de la biblioteca Sowiport.

Aquests *logs* estan parsejats, és a dir transformats, i preprocessats prèviament per a ser posats en un format CSV, és a dir que no els trobem en un format cru. S'han transformat

Camp	Tipus	Descripció
NAMELAST	String	Són els cognoms del visitant
NAMEFIRST	String	És el primer nom del visitant
NAMEMID	String	És el segon nom del visitant
UIN	String	És el Nombre d'Identificació Únic
BDGNBR	Integer	Nombre d'emblema
TYPE_OF_ACCESS	String	Indica el tipus de reunió (per exemple, visitant, treballador)
TOA	Timestamp	Temps d'arribada
POA	String	Lloc d'arribada
TOD	Timestamp	Temps de sortida
POD	String	Lloc de sortida
APPT_MADE_DATE	Timestamp	Data a la qual es va crear de la reunió
APPT_START_DATE	Timestamp	Data a la qual comença la reunió
APPT_END_DATE	Timestamp	Data a la qual acaba la reunió
APPT_CANCEL_DATE	Timestamp	Data a la qual es cancel·la la reunió
Total_People	Integer	Indica el nombre total de persones en la reunió
LAST_UPDATEDBY	String	Indica la persona que ha fet l'última actualització
POST	Timestamp	Àrea de la localització de la cabina de peatge en arribar
LastEntryDate	Timestamp	Indica la data de l'última modificació de l'entrada
TERMINAL_SUFFIX	String	Identifica l'ordinador en la cabina de peatge
visitee_namelast	String	Són els cognoms de la persona visitada
visitee_namefirst	String	És el nom de la persona visitada
MEETING_LOC	String	És la localització de la reunió
MEETING_ROOM	String	Indica el nombre de l'habitació de la reunió
CALLER_NAME_LAST	String	Indica els cognoms de la persona que ha fet creat la reunió
CALLER_NAME_FIRST	String	Indica el primer nom de la persona que ha fet la reunió
CALLER_ROOM	String	És el nombre d'habitació de la persona que ha fet la reunió
Description	String	Detalls respecte a la seguretat o naturalesa de l'esdeveniment
RELEASE_DATE	Timestamp	Indica la data de finalització de la visita

Taula 31: Esquema del conjunt de dades Obama Visitor Logs
[Font: adaptació a partir de [135]]

per a poder col·locar el format cru en una taula en CSV amb etiquetes. La descripció de les etiquetes no està acompanyada del conjunt de dades; per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

El conjunt de dades està estructurat d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

El *dataset* complet té 2,3 GiB. Aquesta font s'ha extret de [137].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 32.

Anàlisi

Podem observar en [138] que l'entropia (Entropy i Messages entropy) de les columnes clicked i recommendation_id és alta.

Camp	Tipus	Descripció
recommendation_id	Integer	És l'identificador de la recomanació
recommendation_class	String	És la classe de la recomanació
cbf_feature_type	String	És el tipus de característica CBF
cbf_feature_count	Integer	És el nombre de característiques CBF
category	String	Indica la categoria
clicked	String	Indica si ha estat clicat o no
request_received	Timestamp	Indica l'instant en el qual s'ha rebut la sol·licitud
response_delivered	Timestamp	Indica l'instant en el qual s'ha enviat la resposta
processingTime	Integer	Indica el temps necessari per a processar la sol·licitud

Taula 32: Esquema del conjunt de dades Recommender Click Logs-Sowiport
[Font: elaboració pròpia a partir del conjunt de dades a [137]]

A.4.10 Seattle COBAN Logs

Descripció

Aquest *dataset* mostra l'activitat de vídeos policials gravats en cotxe, incloent-hi el nombre serial de l'oficial, el començament (data i hora) del vídeo, codi d'activitat posterior a la gravació i descripció.

Aquests *logs* estan parsejats, és a dir transformats, i preprocessats prèviament per a ser posats en un format CSV, és a dir que no els trobem en un format cru. S'han transformat per a poder col·locar el format cru en una taula en CSV amb etiquetes. Aquest conjunt de dades està acompanyat de la descripció de les etiquetes.

El conjunt de dades està estructurat d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

El *dataset* complet té 3,81 GiB. Aquesta font s'ha extret de [139].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 33.

Camp	Tipus	Descripció
fileName	String	És el nom de l'arxiu de la gravació en vídeo
logTime	Timestamp	És l'instant en el qual ha ocorregut l'activitat
userID	String	El valor pot ser el Nombre Serial de l'Oficial que ha completat l'activitat o System si l'activitat ha estat completada per un procés automàtic.
actCode	String	És el codi de l'activitat
ID	Integer	És un identificador numèric
sent	Integer	Indica si s'ha enviat

Taula 33: Esquema del conjunt de dades Seattle COBAN Logs
[Font: elaboració pròpia a partir del conjunt de dades a [139]]

Anàlisi

Podem observar en [140] que les columnes amb més entropia (Messages entropy, Timestamp entropy i Entropy) són fileName, logTime i ID.

A.4.11 Spark

Descripció

El conjunt de *logs* es va recollir agregant *logs* d'un sistema Spark en un laboratori en CUHK amb el propòsit d'investigar-los. Els *logs* són agregats en el nivell de màquina. Cal destacar que tres màquines del clúster han sigut reparades i alguns *logs* s'han perdut. No s'ha aplicat cap transformació ni processament sobre el conjunt de dades.

Aquests *logs* no estan parsejats, ni tampoc tenen etiquetes per a les columnes. Per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és de 2,7 GiB. Aquesta font s'ha extret de [141].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 34.

Camp	Tipus	Descripció
DateTime	Timestamp	Instant en el que es registra el *log*
Level	String	Indica el nivell de severitat del *log*
Component	String	És el component des del qual es genera el *log*
Content	String	És el contingut del *log*. Expressa l'estat del sistema

Taula 34: Esquema del conjunt de dades Spark
[Font: elaboració pròpia a partir del conjunt de dades a [141]]

Anàlisi

A partir dels resultats, que els podem observar a [142], podem destacar que aquest *dataset* té un percentatge alt de files que s'han filtrat en el parsejador (Lost rows percentage) i, per tant, no s'han processat. També cal destacar que l'entropia (Timestamp entropy i Messages entropy) de les columnes DateTime i Content són les més elevades.

Cal destacar que aquest conjunt de dades té un percentatge alt de files descartades a l'hora d'analitzar el *dataset* complet. El percentatge de files descartades és de 17,87 % de files. Aquest és el segon valor més alt de nombre de files descartades, per tant, tampoc sembla ser un conjunt de dades massa bo.

A.4.12 Thunderbird

Descripció

Thunderbird és una font de dades de *logs* recollida per un supercomputador amb Thunderbird en Sandia National Labs (SNL) en Albuquerque. Els *logs* contenen missatges d'alerta

i no alerta. No s'ha aplicat cap transformació ni processament sobre el conjunt de dades.

Aquests *logs* no estan parsejats, ni tampoc tenen etiquetes per a les columnes. Per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és de 29,6 GiB. Aquesta font s'ha extret de [143].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 35.

Camp	Tipus	Descripció
IsAlertMessage?	Integer	Indica si el missatge és una alerta
Timestamp	Timestamp	Instant en el que es registra el *log*
Date	Timestamp	Data en la qual es registra el *log*
Node	String	Node que emet el *log*
DateTime	Timestamp	Instant en el que es registra el *log*
URI	String	Identifica d'on prové el *log*
Content	String	És el contingut del *log*. Expressa l'estat del sistema

Taula 35: Esquema del conjunt de dades Thunderbird
[Font: elaboració pròpia a partir del conjunt de dades a [143]]

Anàlisi

Per a aquest *dataset*, observem en els resultats a [144] que les columnes que més entropia (Timestamp entropy, Timestamp entropy i Messages Entropy) tenen són Timestamp, DateTime i Content. Cal destacar que aquesta font de dades és gran i podria ser una molt bona candidata per a una font de dades gran.

A.4.13 Ubuntu Dialogue Corpus

Descripció

Aquest *dataset* consisteix en quasi un milió de conversacions entre dues persones extret dels *logs* de xat d'Ubuntu, utilitzats per a rebre suport tècnic per a diversos problemes relacionats amb Ubuntu. Les conversacions tenen una mitjana de 8 torns cadascun, amb un mínim de 3 torns. Totes les conversacions són portades a terme de forma textual (sense àudio).

Aquests *logs* estan parsejats, és a dir transformats, i preprocessats prèviament per a ser posats en un format CSV, és a dir que no els trobem en un format cru. S'han transformat per a poder col·locar el format cru en una taula en CSV amb etiquetes. Aquest conjunt de dades està acompanyat de la descripció de les etiquetes.

El conjunt de dades està estructurat d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

El *dataset* complet té 2,8 GiB. Aquesta font s'ha extret de [145].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 36.

Camp	Tipus	Descripció
folder	Integer	És el nom del directori del qual prové un diàleg. Cada arxiu conté diàlegs d'un directori
dialogueId	String	Un nombre identificador per a un diàleg específic. Els identificadors de diàlegs es reutilitzen entre directoris
date	Timestamp	És l'instant en el qual la línia de diàleg es va enviar
from	String	És l'usuari que va enviar la línia de diàleg
to	String	És l'usuari al qual es contestava. En el primer torn del diàleg, aquest camp està buit
text	String	És el text del torn del diàleg

Taula 36: Esquema del conjunt de dades Ubuntu Dialogue Corpus
[Font: elaboració pròpia a partir del conjunt de dades a [145]]

Anàlisi

Podem observar en [146] que aquesta font de dades que totes les columnes tenen una entropia elevada (si mirem les files Entropy, Messages entropy, Timestamp entropy), exceptuant la primera, anomenada folder.

A.4.14 User Logs V2

Descripció

Aquest *dataset* no proveeix una descripció dels seus continguts, encara que es pot deduir per títol que conté *logs* sobre usuaris.

Aquests *logs* estan parsejats, és a dir transformats, i preprocessats prèviament per a ser posats en un format CSV, és a dir que no els trobem en un format cru. S'han transformat per a poder col·locar el format cru en una taula en CSV amb etiquetes. La descripció de les etiquetes no està acompanyada del conjunt de dades; per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

El conjunt de dades està estructurat d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

El *dataset* complet té 1,4 GiB. Aquesta font s'ha extret de [147].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 37.

Camp	Tipus	Descripció
msno	String	És un identificador del *log*
date	Timestamp	És la data en la qual s'ha creat el *log*
num_25	Integer	-
num_50	Integer	-
num_75	Integer	-
num_985	Integer	-
num_100	Integer	-
num_unq	Integer	-
total_secs	Float	És el nombre total de segons

Taula 37: Esquema del conjunt de dades User Logs V2
[Font: elaboració pròpia a partir del conjunt de dades a [147]]

Anàlisi

Podem observar en [148] que en aquesta font de dades la columna que més entropia (Entropy) té és total_secs. Tot i això, aquesta no sembla ser una bona font de dades, ja que no es coneix la procedència exacta de les dades. Per tant, no es pot saber correctament ni el significat de les etiquetes, ni si les dades s'han extret de forma correcta.

A.4.15 Windows

Descripció

Aquesta font de dades de *logs* es va recollir agregant *logs* d'un ordinador de laboratori executant Windows 7. Els *logs* originals estaven situats en C:\Windows*logs*\CBS. CBS (Component Based Servicing) és una arquitectura de components en Windows, que funciona per actualitzar al nivell de paquets les instal·lacions del sistema operatiu.

Aquests *logs* no estan parsejats, ni tampoc tenen etiquetes per a les columnes. Per aquest motiu, hem assignat el nom als camps de forma arbitrària, pensant en quin seria el significat real dels camps i que semblava més adequat mirant una mostra del *dataset*.

Per tant, vam estructurar amb el *Datasets Profiler* el conjunt de dades d'una forma estandarditzada i seguint la Tercera Forma Normal (3FN). En resum, cada variable forma una columna i conté valors, cada observació forma una fila i tota l'observació forma una única taula.

La grandària de la font de dades és de 26,1 GiB i comprenen un període de més de 226 dies. Aquesta font s'ha extret de [149].

Esquema

L'esquema d'aquesta font de dades es pot observar a la Taula 38.

Camp	Tipus	Descripció
DateTime	Timestamp	Instant en el que es registra el *log*
Level	String	Indica el nivell de severitat del *log*
Component	String	Indica el component des del qual s'ha generat el *log*
Content	String	És el contingut del *log*. Expressa l'estat del sistema

Taula 38: Esquema del conjunt de dades Windows
[Font: elaboració pròpia a partir del conjunt de dades a [149]]

Anàlisi

Podem observar en [150] que les columnes tenen una entropia (si mirem les files Timestamp entropy i Messages entropy) al voltant de 10 bits, encara que la columna Component és la que menys en té. Aquest *dataset* podria ser interessant per tal de veure com afectaria el clúster un volum de dades grans.

A.5 Classificació dels *Datasets*

Un cop analitzades les fonts de dades, ja les podem classificar en un rànquing per a veure quines utilitzarem per a la comparativa dels dos *frameworks* de processament de dades.

En aquesta classificació hi ha un fet a remarcar. La mida que hem anat mostrant fins ara és la mida real dels *logs*, és a dir, sense comprimir. Aquest fet és important considerar-lo, ja que els *logs*, per naturalesa, acostumen a seguir uns determinats patrons que fan que la compressió sigui molt eficient, obtenint ràtios de compressió molt grans.

També farem menció de la mida real del conjunt de dades en format Parquet. Aquesta és important perquè és diferent de la mida real del conjunt de dades i normalment és més petita. Aquesta serà l'entrada del programa que desenvoluparem per a fer els experiments.

Cal recalcar el fet que en el *Datasets Profiler*, vam deshabilitar la compressió dels conjunts de dades per a ser independents de l'algorisme de compressió utilitzat, no generar cap esbiaix i enfocar-nos només en el rendiment dels *frameworks* de processament de dades, encara que tot i això, al transformar el conjunt de dades cru al format Parquet, la mida final ha sigut més petita que la del conjunt de dades cru.

Addicionalment, tindrem també en compte la quantitat de files reals que es processaran a l'arxiu Parquet, ja que com hem mencionat amb anterioritat, per a alguns conjunts de dades ha calgut fer una tasca de parseig i, aquesta va eliminar aquelles files que no seguien un determinat patró.

Una dada important és que quan aquests conjunts de dades s'han posat a HDFS, tant en format cru com en el format parsejat en un arxiu Parquet, vam observar que la distribució que va fer HDFS és la mateixa entre les tres màquines. És a dir, que cada bloc de cada arxiu es replica tres cops en tres màquines diferents i, en el clúster només tenim tres nodes que participen en emmagatzemar les dades de HDFS, aleshores, a totes les màquines tenim exactament la mateixa informació. Això vol dir que els conjunts de dades no estan fragmentats, tot i així, ho hem fet per a maximitzar el paral·lelisme ja que Spark podrà decidir on

es processarà cada fragment.

Aquesta data l'hem obtingut seguint el mètode indicat a [151], on executant una comanda, es poden veure tots els blocs dels arxius i en quin node estan emmagatzemats.

A.5.1 Font de dades petita (181 MiB)

Podríem utilitzar la font de dades **Obama Visitor Logs**, per la gran quantitat de columnes de la que disposa.

La mida final de l'arxiu Parquet amb el conjunt de dades parsejat, que servirà d'entrada per als nostres experiments és 181,0 MiB. La grandària que ocupa en total entre totes les rèpliques, tenint un factor de replicació de 3 a HDFS és 543,1 MiB.

La quantitat de columnes descartades són 7.799, que forma un 0,13% de files descartades, d'un total de 5.914.773 files, considerant el conjunt de dades complet. Podem observar que aquesta porció és raonable per a poder considerar que aquest conjunt de dades és bo.

A.5.2 Font de dades mitjana (719,6 MiB)

Emprarem **Ad Display/Click Data on Taobao.com** perquè es presenta juntament amb altres dues taules i ens permetrà analitzar més a fons les joins.

La mida de l'arxiu Parquet amb el conjunt de dades parsejat, que servirà d'entrada per als nostres experiments és 719,6 MiB. La grandària que ocupa en total entre totes les rèpliques, tenint un factor de replicació de 3 a HDFS és 2,1 GiB.

La quantitat de columnes descartades és exactament una, que es correspon a la capçalera que hi ha al començament del conjunt de dades, d'un total de 26.557.962 files, considerant el conjunt de dades complet.

A.5.3 Font de dades gran (19,6 GiB)

Thunderbird ha demostrat ser una candidata robusta, a causa de la seva grandària.

La mida de l'arxiu Parquet amb el conjunt de dades parsejat, que servirà d'entrada per als nostres experiments és 19,6 GiB. La mida que ocupa en total entre totes les rèpliques, tenint un factor de replicació de 3 a HDFS és 58,9 GiB.

La quantitat de columnes descartades és 1.238 files, que suposa un 0,0006 % de files descartades, d'un total de 211.212.192 files, considerant el conjunt de dades complet.

Apèndix B

Adaptació del *Testbed* i *Datasets Profiler* per a executar-se al clúster

En aquest capítol exposarem per als dos *frameworks* Spark i MapReduce, quines han sigut les dificultats més importants a l'hora de fer la migració des d'un entorn local, a un entorn en un clúster. Classificarem els problemes per tipus i també per quantitat de temps dedicat.

Molts d'aquests errors estan associats al fet que nosaltres estem administrant els nostres propis servidors. Molts d'aquests problemes de configuració serien inexistents si estiguéssim utilitzant un sistema en un entorn al núvol, proveït per empreses com Amazon Web Services o Google Cloud Platform. Aquests proveïdors proporcionen serveis gestionats i pre-configurats per a que es els desenvolupadors no s'hagin de preocupar de la difícil configuració d'aquests sistemes.

B.0.1 Canvi del sistema d'arxius a HDFS

Independentment de si s'està utilitzant el *framework* MapReduce o Spark, s'ha de fer un canvi en el projecte i si el volem executar en un clúster. Si estem executant el programa en un clúster, aleshores, tots els arxius hauran d'estar en HDFS. D'aquesta forma evitem fer que el *driver* hagi d'enviar individualment tots els arxius a tots elx executors quan es prepara l'execució. També, fa més eficient la execució, ja que en el cas de Spark, les operacions amb *narrow depepndencies*, s'executen de forma local al node. Hi ha una interfície dissenyada específicament per a permetre tant referir-se als arxius locals com als arxius en HDFS. Un petit problema d'aquesta interfície és que ofereix un repertori d'operacions a realitzar més restringit que si tractéssim els fitxers directament. El temps dedicat a resoldre aquest problema van ser 6 hores.

B.0.2 Tuneig del clúster

Aquest va ser un error que no ens va costar molt de trobar, però ens va portar temps arribar a una bona configuració per a solucionar-ho. Per defecte, els dos *frameworks* utilitzen uns valors molt baixos d'utilització dels recursos i no permeten aprofitar tot el clúster. Per tant, hem hagut d'explorar quin seria el valor més correcte i llegir molta documentació i

exemples de configuració abans d'arribar a uns valors de configuració de la memòria RAM i dels nuclis definitiva. El temps dedicat a resoldre aquest problema van ser 4 hores.

B.0.3 Ports

Es necessita que els ports estiguin oberts per a poder accedir a les pàgines web que contenen les consoles tant per a Hadoop MapReduce, com per a Spark. D'aquesta forma es pot accedir de forma visual a informació del clúster i al seu estat. El temps dedicat a resoldre aquest problema van ser 1 hora.

B.0.4 Errors en la configuració de Yarn

Si a l'hora de definir els paràmetres de configuració en qualsevol arxiu de configuració de Yarn, aleshores aquest sistema no podrà parsejar els arxius de configuració i conseqüentment no podrà assignar un executor a cap dels *jobs*, quedant-se aquests encolats per a ser executats indefinidament. El temps dedicat a resoldre aquest problema van ser 1 hora.

B.0.5 Propagació de la Configuració de Yarn

Si volem canviar la configuració de Yarn, ho hem de fer en un node, copiar a la resta de nodes i al *master* la nova configuració i reiniciar el *framework* Spark i MapReduce. Si no es propaga la configuració a través de tots els nodes, es seguirà utilitzant la configuració vella. El temps dedicat a resoldre aquest problema van ser 5 minuts.

B.0.6 Spark

Dependències

Quan executes amb el *wrapper* spark-submit el JAR, aleshores, és com si el JAR que li passem és una dependència. Spark li proporciona les dependències en temps d'execució (MapReduce també ho fa). Un problema que pot ocórrer és el que es descriu a [152], on podem observar que la dependència de Guava es requereix en el nostre projecte amb la versió 30.1.1-jre, mentre que Spark utilitza la versió 14. Cal recalcar que en el moment de fer la invocació, Spark li passa al programa la dependència de Guava amb la versió 14. Nosaltres utilitzem la estructura de dades de Graf per als plans lògic i físic, per la qual cosa, aquesta llibreria era imprescindible. Aleshores, el que hem decidit fer és passa a substituït el *plugin* de maven assembly, que utilitzàvem per a crear el Uber JAR, i hem utilitzat el *plugin* del maven shade. A continuació, vam procedir a fer un *shade* de la dependència de Guava. Quan es creava el JAR, es canviava a tot arreu automàticament, en el JAR, les referències a Guava, per a que apuntessin a la versió "Shaded". El temps dedicat a resoldre aquest problema van ser 10 hores.

Ús obligatori de wrapper

Spark està dissenyat de forma que s'hagi d'utilitzar sempre un *wrapper* al voltant del JAR per a poder fer una execució al clúster o en local. Per a la execució en local, he posat a dintre del JAR les dependències necessàries per a poder fer l'execució sense el *wrapper*. En canvi, per a la versió en clúster, va ser molt difícil reutilitzar el mateix codi i no utilitzar

el *wrapper*. Hi havia una tendència a rebutjar els valors "yarn". Tot el temps donava error de parseig d'aquest valor. Per tant, vam utilitzar el *wrapper*. El temps dedicat a resoldre aquest problema van ser 6 hores.

Migració amb una única comanda

No va ser totalment fàcil adaptar el JAR que funcionava amb Spark en local per al clúster de forma immediata, ja que hi havia un problema en la especificació del *master*. Resulta que si en Java, el *master* estava especificat amb la URL del *master*, aleshores no funcionava, el Job es quedava en ACCEPTED fins que saltava una excepció de *timeout*. Això es va arreglar en el moment en el que es va especificar tant en *wrapper* com en el programa Java que el *master* és Yarn. El temps dedicat a resoldre aquest problema van ser 2 hores.

Rellevància dels errors

Els errors en Spark no són desxifrables si fem una execució amb el "deploy-modeçlúster ja que no es veu l'execució. En canvi, si utilitzem el "deploy-modeçlient, que fa una execució del *driver* local a on es realitza la invocació del spark-submit, aleshores, podrem veure perfectament el missatge d'error i podrem debugar correctament el sistema. El temps dedicat a resoldre aquest problema van ser 1 hora.

B.0.7 MapReduce

Funcionament en local

Per a que funcioni en local el MapReduce, s'han d'incloure un conjunt molt específic de dependències. Si es posen dependències de més el codi simplement falla, amb un missatge que sembla no relacionat i fa que es perdi molt temps buscant la solució. El temps dedicat a resoldre aquest problema van ser 10 hores.

Rellevància dels errors

Molts cops ha sigut difícil de trobar els errors, ja que estan en diversos arxius i, fins i tot, molts cops es va basar en assaig i error. Aquesta forma de debugar és terrible ja que no se sap si s'està arreglant l'error o si s'està introduint codi de més. Cal recalcar que aquests error, molts cops quan surt i són visibles, tant en local com en mode clúster són gairebé intel·ligibles. Les solucions a cada problema són molt diverses i no se sap mai si s'està solucionant exactament el problema o no. El temps dedicat a resoldre aquest problema van ser 5 hores.

Opcions a Yarn

Cal recalcar el fet que també va ser necessari afegir algunes opcions addicionals a Yarn per a que acabi de funcionar. Vam provar que executés el programa de *wordcount*. Vam afegir solucions als *bugs* fins que finalment, vam fer que funcionés aquest exemple en MapReduce. El temps dedicat a resoldre aquest problema van ser 5 hores.

Problema de amb el reconeixement dels workers

Aquest problema vé del fet que el *master* sembla no saber com comunicar-se amb els nodes quan s'està fent una execució en mode clúster. El missatge d'error és intel·ligible i fa referència a una excepció de connexió. Es va solucionar aquest problema simplificant l'arxiu /etc/hosts. Es van suprimir les entrades que redirigien al nom del host al *localhost*. Eliminant aquelles entrades, es va aconseguir que es pogués executar el programa de *wordcount*. El temps dedicat a resoldre aquest problema van ser 2 hores.

Empaquetació dels Mappers i Reducers

Quan es fa una execució en local, no és necessari fer servir la rutina "setJarByClass"del Job per a assignar una classe que empaqueti el Mapper i Reducer. Si no fem la crida a aquella funció, el codi segueix funcionant. En canvi, si estem fent una execució en el clúster, MapRecude fallarà sigilosament a l'hora de fer la preparació dels JARs per a enviar-los als Mappers i Reducers. Aleshores, vam decidir empaquetar els Mappers i Reducers en una classe que els empaqueta juntament. Aleshores, utilitzem la crida a la subrutina per a configurar el JARByClass. El temps dedicat a resoldre aquest problema van ser 2 hores.

B.0.8 Errors en la configuració de MapReduce

Si a l'hora de definir els paràmetres de configuració en qualsevol arxiu de configuració de MapReduce, que segueix un format XML similar al de la configuració de Yarn. Si hi ha algun problema de parseig o si algun paràmetre no està correctament definit, aleshores MapReduce no funcionarà. El temps dedicat a resoldre aquest problema van ser 2 hores.

B.0.9 Propagació de la Configuració de MapReduce

Si volem canviar la configuració de MapReduce, al igual que Yarn, ho hem de fer en un node, copiar a la resta de nodes i al *master* la nova configuració i reiniciar el *framework* Spark i MapReduce. Si no es propaga la configuració a través de tots els nodes, es seguirà utilitzant la configuració vella. Això en canvi no passa amb Spark, que permet definir la configuració a l'hora d'invocar el *wrapper*. El temps dedicat a resoldre aquest problema van ser 1 hora.

B.1 Conclusions

Spark té menys errors i, la suma total dels temps per dos sistemes, si sumem totes les hores dels dos *frameworks*, és més petita per a Spark.

Apèndix C

Operacions d'Àlgebra Relacional

En aquest capítol ens centrarem en explicar les principals operacions relacionals que tractarem en aquest projecte i les analitzarem per a veure com es podrien implementar en els diferents *frameworks*.

C.1 Operacions en Spark

Abans d'explicar com s'implementarien les diferents operacions en Spark, primer fem una explicació general i en alt nivell del seu funcionament. Quan un usuari fa una invocació a través d'un llenguatge de programació determinat, aquest es comunica amb el node anomenat *driver*, que és la part del *framework* encarregada de registrar la crida. Hi ha dos tipus de crida o invocació que es poden fer: transformacions i accions. Si es fa una transformació, la seva execució no és imminent, sinó que es defereix fins que es fa la invocació d'una acció. Una acció consisteix en una operació que requereix que el resultat de totes les operacions anteriors s'hagi computat.

Per a decidir quines operacions s'han d'executar, el *driver* organitza en un graf acíclic dirigit (*Directed Acyclic Graph*, DAG) totes les operacions i observa les dependències que existeixen. Un cop ha analitzat quines operacions s'han d'executar, envia ordres als *workers* i comunica quins conjunts de dades d'entrada i de sortida han d'utilitzar. Els conjunts de dades es troben a memòria dels nodes executors, així que tots els nodes executors processaran el fragment del conjunt de dades que tingui a la seva memòria local si l'operació no necessita intercanviar dades amb els altres nodes (*narrow dependency*), o bé fa un processament determinat i després intercanvia dades amb els altres nodes si és necessari (*wide dependency*). Aquestes dependències són pròpies de cada operació i serà l'objecte d'anàlisi en els pròxims apartats.

C.2 Com es mapejen les principals operacions relacionals amb Spark?

Analitzarem en aquest apartat com s'implementarien les principals operacions relacionals amb Spark.

```sql
SELECT *
FROM bank
WHERE age < 20
```

Fragment C.1: Clàusula de selecció en SQL
[Font: elaboració pròpia]

C.2.1 Selecció

Aquesta operació consisteix en seleccionar d'una taula només aquelles files que satisfacin un predicat. El predicat normalment fa referència a una o més columnes. Aquesta operació redueix el nombre de files de sortida o el deixa igual que a l'entrada. Per exemple, en SQL tindríem la clàusula que podem observar en el Fragment C.1.

Aquesta operació es tradueix a Spark amb l'operació de *filter* i es pot veure un exemple de la seva utilització en el repositori del *Testbed*, en [153].

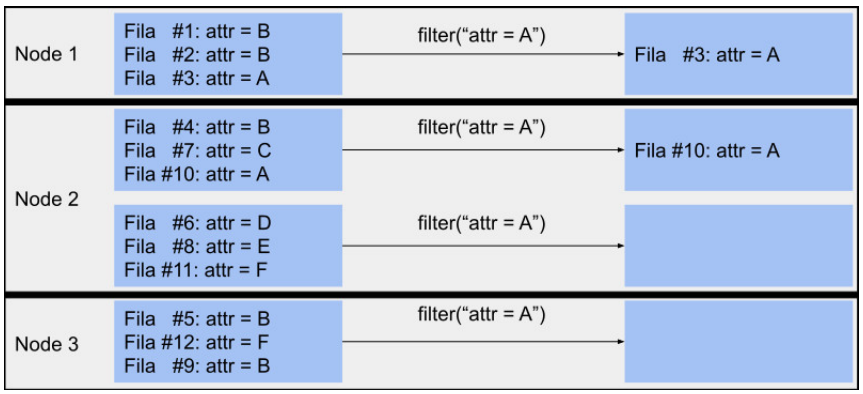


Figura 89: Selecció en Spark
[Font: elaboració pròpia]

Observem a la Figura 89 que no hi hauria cap problema en fer-ho localment, per tant, podem executar el mateix filtratge a cada partició individualment. Aleshores, es tracta d'una *narrow dependency*.

C.2.2 Projecció

Aquesta operació consisteix en seleccionar d'una taula només un conjunt especificat de columnes. Aquesta operació redueix el nombre de files de sortida o el deixa igual que a l'entrada.

Per exemple, en SQL tindríem la clàusula que podem observar en el Fragment C.2.

Aquesta operació es tradueix a Spark amb l'operació de *select*.

```sql
SELECT age, job
FROM bank
```

Fragment C.2: Clàusula de projecció en SQL
[Font: elaboració pròpia]

L'operació de la Figura 90 és una *narrow dependency*, ja que si ho pensem de la següent forma:

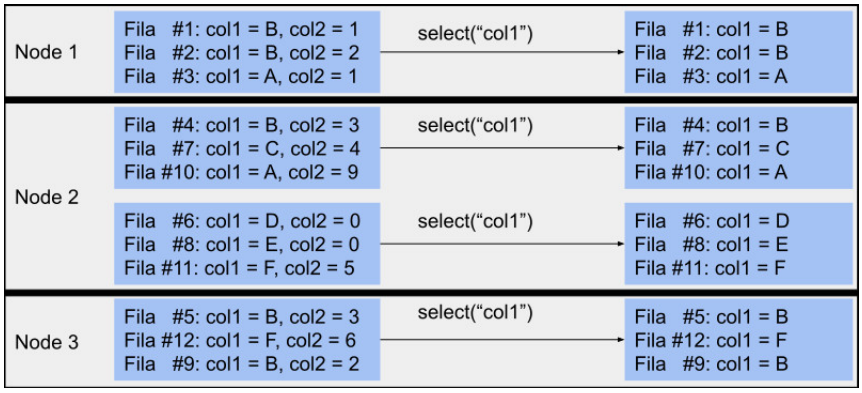


Figura 90: Projecció en Spark
[Font: elaboració pròpia]

Observem que no hi hauria cap problema en fer-ho localment, per tant, podem executar la mateixa projecció a cada partició individualment. Observem un exemple de la seva utilització en [154].

C.2.3 Unió

Aquestes operació consisteix en combinar dos conjunts de dades de tal forma que el resultat de l'operació són aquelles files del primer conjunt més les files del segon conjunt que no estan en el primer conjunt. La cardinalitat del resultat és la el nombre de files del primer conjunt de dades més el nombre de files del segon conjunt de dades menys el nombre de files iguals entre els dos conjunts de dades. Cal notar que aquesta operació elimina els duplicats.

Per exemple, en SQL tindríem la clàusula que podem observar en el Fragment C.3.

Aquesta operació es tradueix a Spark a l'operació d'*union*. Es pot veure un exemple d'utilització en [155]. En aquest exemple, hem utilitzat l'operació d'*union* i després *distinct* perquè *union* a Spark no funciona exactament com el UNION de SQL, sinó més bé com un UNION ALL, on es mantenen els duplicats. Per a esborrar els duplicats, a Spark és necessària l'operació de *distinct* [156].

Aquesta operació implica una *wide dependency*, ja que depèn de més d'una partició per a executar cada operació i, si ho visualitzem com en la Figura 91, veurem que és necessari fer un intercanvi de dades a través de la xarxa.

```
1 SELECT * FROM bank
2 UNION
3 SELECT * FROM bank
```

Fragment C.3: Clàusula d'unió en SQL
[Font: elaboració pròpia]

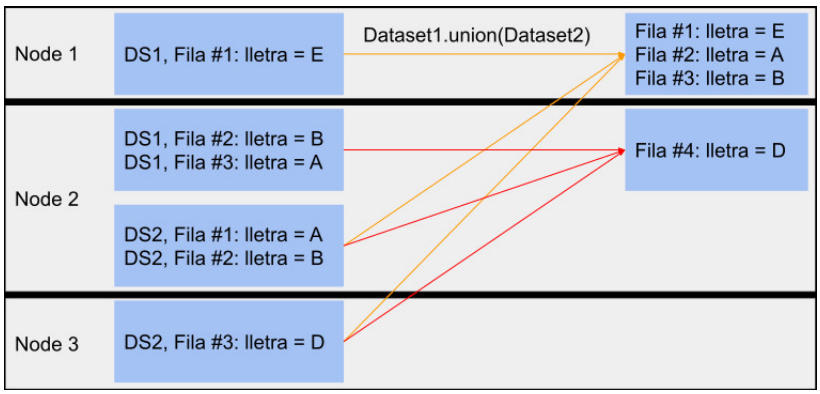


Figura 91: Operacions sobre conjunts en Spark
[Font: elaboració pròpia]

```sql
SELECT AVG(age)
FROM bank
```

Fragment C.4: Clàusula d'agregació en SQL
[Font: elaboració pròpia]

C.2.4 Agregació

Aquesta operació consisteix en executar una operació (per exemple una suma o mitjana) sobre un conjunt de dades. La quantitat de files del conjunt de dades resultant acostuma a disminuir a la sortida.

Per exemple, en SQL tindríem la clàusula que podem observar en el Fragment C.4.

Aquesta operació es tradueix a Spark a l'operació de agg i es pot veure un exemple de la seva utilització a [157].

Aquesta operació és una *wide dependency*, ja que depèn de més d'una partició per a executar l'operació i, si ho visualitzem com en la Figura 92, observarem que és necessari fer un intercanvi de les dades a través de la xarxa.

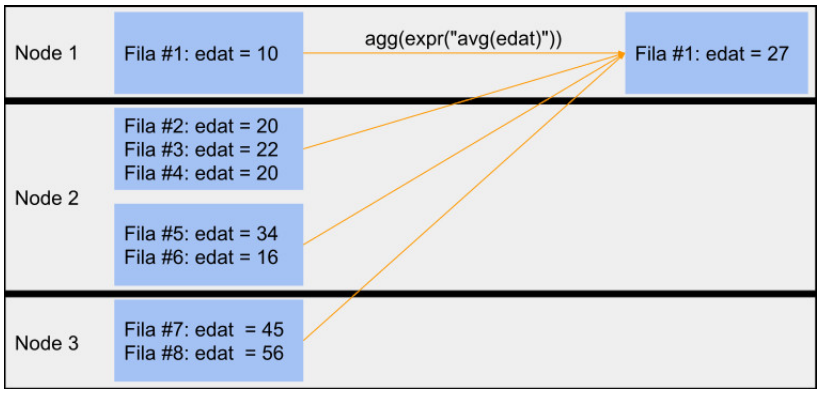


Figura 92: Agregació en Spark
[Font: elaboració pròpia]

```sql
SELECT job
FROM bank
GROUP BY job
```

Fragment C.5: Clàusula d'agrupació en SQL
[Font: elaboració pròpia]

C.2.5 Agrupació

Aquesta operació consisteix en agrupar les files pels valors d'un conjunt d'atributs (més d'un atribut). La cardinalitat de la taula resultant acostuma a disminuir a la sortida ja que estarà formada per tantes tuples com valors diferents tingui el atribut pel qual s'agrupa.

Per exemple, en SQL tindríem la clàusula que podem observar en el Fragment C.5.

Aquesta operació es tradueix a Spark a l'operació de *groupBy*. Es pot observar un exemple d'aquesta operació en [158]. En aquest exemple, hem inclòs una agregació buida al fina de l'agrupació per tal d'obtenir un Dataset¡Row¿. Això és degut a que el groupBy està pensat en aquesta API per fer agrupacions per a agregar dades.

Aquesta operació és una *wide dependency*, ja que depèn de més d'una partició per a executar l'operació i, si ho visualitzem com en la Figura 93, observarem que és necessari fer un intercanvi de dades a través de la xarxa.

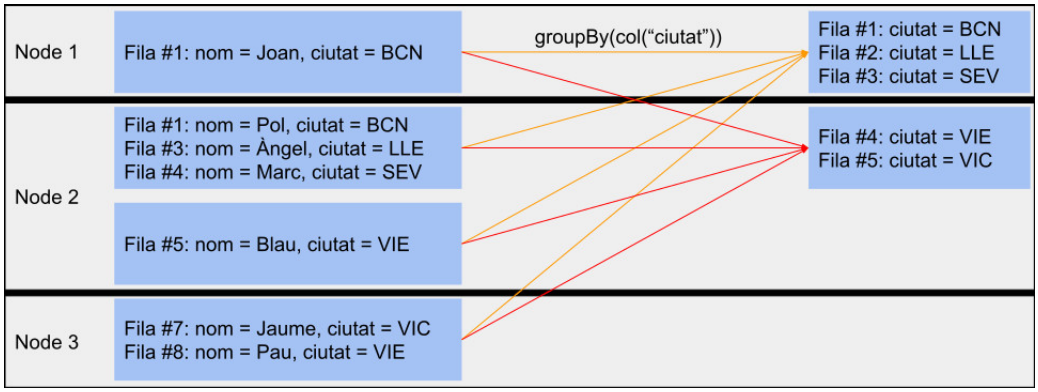


Figura 93: Agrupació en Spark
[Font: elaboració pròpia]

C.2.6 *Join*

Aquesta operació consisteix en ajuntar les files de dos conjunts de dades a través d'un a diversos atributs. La quantitat de files del resultat pot a la sortida ja que podem pensar en la *join* com una operació de producte cartesià amb una operació de selecció, que determina quines files es seleccionen a la sortida.

Per exemple, en SQL tindríem la clàusula que podem observar en el Fragment C.6.

Aquesta operació es tradueix a Spark a l'operació de *join* i es pot observar un exemple d'utilització a [159].

Aquesta operació és una *wide dependency*, ja que depèn de més d'una partició per a

```sql
SELECT b.*, h.*
FROM bank AS b NATURAL INNER JOIN heart AS h
WHERE b.age = h.age
```

Fragment C.6: Clàusula de *join* en SQL
[Font: elaboració pròpia]

executar l'operació i, si ho visualitzem com en la Figura 94, observarem que és necessari fer un intercanvi de dades a través de la xarxa.

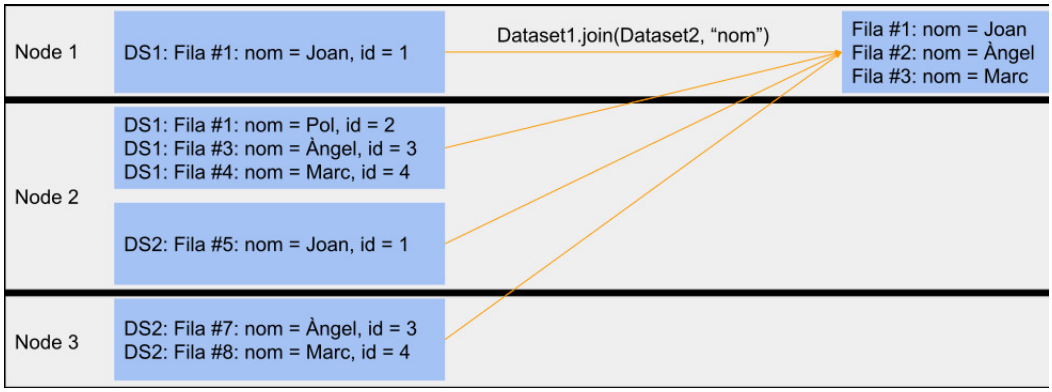


Figura 94: *Join* en Spark
[Font: elaboració pròpia]

C.3 Operacions en MapReduce

Abans d'explicar com s'implementarien les diferents operacions en MapReduce, primer fem una explicació general i en alt nivell del seu funcionament. Per a que un usuari executi una operació en el *framework* MapReduce, ha de crear primer un *Job* des del programa. En aquest Job es defineix la configuració d'aquest, és a dir, el codi dels Mappers, el codi dels Reducers, la quantitat de tasques dels Reducers, etc. Tota aquesta configuració s'enviarà després al *application master* per a que pugui distribuir els codis i pugui orquestrar l'execució. Un cop hagi distribuït els JAR amb l'operació de Map als Mappers i el JAR amb l'operació de Reduce als Reducers, ja es pot començar a fer l'execució. Com els conjunts de dades estaran emmagatzemats en un sistema d'arxius distribuït, aleshores, cada node llegirà el fragment del conjunt de dades que estigui present en el node local. Primer s'executaran els Mappers que mapejaran d'un conjunt de dades a un altre els valoris i, aquests resultats s'ordenaran amb una fase de Shuffle. Posteriorment, s'enviaran al Reducers els resultats ordenats del Mapper i es procedirà a reduir el resultat en un únic valor per Reducer.

C.4 Com es mapejen les principals operacions relacionals amb MapReduce?

Analitzarem en aquest apartat com s'implementarien les principals operacions relacionals amb MapReduce

C.4.1 Selecció

Per a l'execució d'aquesta operació necessitem només una operació de Map que filtri i no deixi passar al conjunt de sortida els valors que no volem. Per a fer més eficient l'execució, deshabilitarem els Reducers, ja que no calen. Un exemple d'implementació es pot trobar a [160].

C.4.2 Projecció

Per a l'execució d'aquesta operació necessitarem, al igual que en l'operació de selecció, un Map que deixi passar les columnes que vulquem al conjunt de dades de sortida. Per a fer més eficient l'execució, deshabilitarem els Reducers, ja que tampoc calem. Un exemple d'implementació es pot trobar a [161].

C.4.3 Unió

Per a l'execució d'aquesta operació necessitarem dos Maps diferents, un combiner i un reducer. Primer de tot, necessitem dos Maps per a poder mapejar de dos conjunts de dades diferents a un únic conjunt comú als dos. Després, s'aplica un combiner, que és com una etapa de reducció que s'aplica localment, és a dir, just després d'aplicar-se el Map, s'aplica una fase de reducció anomenada Combiner i després s'aplica el Shuffle i la fase de Reducció final. Aquesta fase de combinació només es pot aplicar si l'operació és associativa i commutativa [162] i permet reduir la quantitat de dades que es transmeten a través de la xarxa.

En l'operació d'unió, el Combiner rep diverses entrades amb una mateixa clau i les redueix retornant la clau. Això ens serveix per a eliminar els valors repetits localment. En el Reducer, s'eliminen els valors repetits globalment. Un exemple d'implementació el podem trobar a [163].

C.4.4 Agregació

Per a aquesta operació és molt més difícil pel fet que hem de fer específicament un Mapper i Reducer específic per a cada operació. És a dir, per a fer una agregació del tipus suma, aleshores haurem de fer un Mapper que agrupi els valors de les files i el Reducer haurà d'executar l'operació; es pot considerar també l'opció d'incloure un Combiner que optimitzi els càlculs.

Agafarem l'operació de suma com a exemple. Aquesta consisteix en un Map, un Combine i un Reduce. El Map agafa un valor determinat i el mapeja amb una clau determinada i el valor es posarà a la part de valor. La clau serà la mateixa per a totes les files mapejades per a que ens resulti en una única fila. El Combiner farà la suma de dos valors i els escriurà a a sortida; aquest element farà l'agregació a nivell local. Finalment, el Reducer acabarà de fer l'agregació a nivell global.

Es pot observar un exemple de l'agregació a [164].

C.4.5 Agrupació

Per a implementar aquesta operació, necessitarem un Map i un Reduce. El Map agafarà un determinat valor i el posarà com a clau i deixarà buit el valor. Aleshores, quan es faci la fase de Shuffle, els valors s'ordenaran i el Reducer rebrà els diferents valors. El resultat d'aquesta operació serà el mateix que s'obtindria d'executar una operació de DISTINCT relacional sobre el mateix conjunt de dades, ja que no fem una agregació sobre les dades.

Es pot observar un exemple de l'agrupació a [165].

C.4.6 *Join*

Per a aquesta operació necessitarem dos Maps i un Reducer. Cada Map s'encarregarà de mapejar d'un dels conjunt de dades d'entrada a un conjunt comú i es posarà com a clau el valor de la columna per la que fem la *equi join*. Hem d'utilitzar també un tipus de prefix en els valors per a determinar quines files provenen del Map de l'esquerra i quines del Map de la dreta. D'aquesta forma podrem establir com s'executaran els bucles en el Reducer. El Reducer rebrà una clau i diversos valors, on la clau serà el valor de la columna per la que fem la *equi join* i com a valors hi ha les diferents files. Primer, classificarà les files en files provinents del Map de l'esquerra i files provinents del Map de la dreta i farà un producte cartesià de les dades.

Es pot observar un exemple de la *join* a [166].

Apèndix D

Estructura dels repositoris de GitHub

Exposarem en aquest capítol l'estructura als repositoris del GitHub i també posarem una explicació dels diferents arxius.

D.1 Repositori del *Datasets Profiler*

Aquest repositori es troba a [82].

Els continguts d'aquest repositori es mostren a continuació:

- datasets_profiler: conté tot el codi font del *Datasets Profiler*.
- datasets_profiling_information: conté tota la informació dels paràmetres, resultats, resultats processats, conjunts de dades i càlculs fets.
  - cardinality_calculations: conté càlculs i la distribució dels arxius a HDFS.
    - big_join_cardinality.txt: conté els càlculs i el raonament de perquè s'utilitzen només dues columnes per al conjunt de dades mitjà.
    - file_distribution_in_hdfs.txt: conté la distribució de tots els arxius a HDFS (conjunts de dades, conjunts de dades parsejats...).
    - file_distribution_in_hdfs_processed.txt: és el mateix contingut que a file_distribution_in_hdfs.txt però està processat per a que sigui més llegible.
    - join_cardinality.txt: conté els càlculs de la *join*.
    - select_project_cardinality.txt: conté els càlculs de l'operació de selecció i projecció.
  - parameters: conté tots els paràmetres utilitzats com a entrada per al *Datasets Profiler*.
    - parameters_list_get_testbed_input.json: permet obtenir l'entrada del *Testbed*

        * parameters_list_integration_test.json: defineix una sèrie de tests d'integració
        * parameters_list_samples_10e3.json: defineix els conjunts de dades per a crear un perfil estadístic de mostres de mil files de cada conjunt de dades.
        * parameters_list_samples_10e4.json: defineix els conjunts de dades per a crear un perfil estadístic de mostres de deu mil files de cada conjunt de dades.
        * parameters_list_samples_10e5.json: defineix els conjunts de dades per a crear un perfil estadístic de mostres de cent mil files de cada conjunt de dades.
        * parameters_list_samples_10e6.json: defineix els conjunts de dades per a crear un perfil estadístic de mostres d'un milió de files de cada conjunt de dades.
        * parameters_list_samples_10e7.json: defineix els conjunts de dades per a crear un perfil estadístic de mostres de deu milions de files de cada conjunt de dades.
        * parameters_list_samples_10e8.json: defineix els conjunts de dades per a crear un perfil estadístic de mostres de cent milions de files de cada conjunt de dades.
        * parameters_list_whole_datasets.json: defineix els conjunts de dades per a crear un perfil estadístic dels conjunt de dades complets.
    - processed_excels: conté tots els excels processats dels conjunts de dades
    - raw_excels: conté els resultats dels excels sense processar dels conjunts de dades obtinguts com a sortida del *Datasets Profiler*. Segueixen una estructura similar als paràmetres amb els que es van invocar.
- scripts: conté tots els scripts
    - build-wheel.sh: serveix per a crear un arxiu *wheel* del *Datasets Profiler*.
    - calculate_big_join_cardinality.py: permet calcular la cardinalitat de l'operació de la *join* gran.
    - calculate_join_cardinality.py: permet calcular la cardinalitat de l'operació de la *join*.
    - call_statistics_formatter.py: és un script que permet formatar la sortida del *Datasets Profiler* que vam utilitzar per a construir els excels processats dels resultats.
    - check_parsed_dataset.py: aquest script comprova si el conjunt de dades parsejat és correcte.
    - check_tests.sh: aquest script feia al *Datasets Profiler* passar uns tests d'integració i comprovava si els passava tots.
    - run-cluster.sh: aquest script permet executar en el clúster el *Datasets Profiler*
    - run-local.sh: aquest script permet executar en local el *Datasets Profiler*

  - spark-avro_2.12-3.1.1.jar: aquest arxiu és necessari per a que es puguin llegir i escriure arxius amb el format Avro.

- .gitignore: permet ignorar arxius a l'hora de crear *commits* amb git.
- LICENSE: és l'arxiu de llicència
- README.md: és una petita explicació de com funciona el *Datasets Profiler*.
- __main__.py és un arxiu de *bootstrap* per al *Datasets Profiler*.
- setup.py: permet definir les dependències del programa *Datasets Profiler*.

D.2 Repositori del *Testbed*

Aquest repositori es troba a [83]

Els continguts d'aquest repositori es mostren a continuació:

- experiments_information: conté tota la informació dels experiments.
  - pipelines: conté els pipelines que s'han utilitzat com a entrada del *Testbed*. S'han generat automàticament amb els scripts ¡operation_name¿_batch_generator.py, on operation_name és el nom de l'operació.
  - processed_excels: conté tots els resultats del *Testbed* processats.
  - raw_excels: conté els excels crus, directament obtinguts del *Testbed*.
    * full_ram_experiments: és la sortida d'executar els experiments corresponents a tots els pipelines.
    * minimum_ram_memory_threshold_exploration: és la sortida d'executar els experiments corresponents als pipelines de big_join.
  - runner_scripts: són els scripts que si s'executen invocant-los amb bash, executen una sèrie de *pipelines*.
  - utils: conté una eina que permet calcular els valors dels paràmetres del *tuning* del *clúster*.
- README.md: és una petita explicació de com funciona el *Testbed*.
- scripts: conté tots els scripts i *templates*
- templates: conté els templates que s'utilitzen pels scripts en Python per a generar els *runners*.
  - batch_generators_commons.py: conté codi que es reutilitza pels scripts anomenats *_generator.py
  - big_join_batch_generator.py: conté el codi per a generar els *pipelines* i el script *runner* per l'operació de *Join* gran.

  - correctness_demonstration.py: permet demostrar la correctesa del *Tedtbed*.
  - experiments_runner.sh: permet executar tres cops el testbed per a prendre temps i un cop més per a fer l'instrumentació del *pipeline*.
  - install.sh: permet instal·lar el *Testbed* amb totes les seves dependències.
  - join_batch_generator.py: conté el codi per a generar els *pipelines* i el script *runner* per l'operació de *Join*.
  - mount_local_disk_loop_devices.sh: permet muntar el disc *loop* per a poder mesurar les estadístiques d'entrada i sortida del disc local.
  - project_batch:generator.py: conté el codi per a generar els *pipelines* i el script *runner* per l'operació de projecció.
  - select_batch:generator.py: conté el codi per a generar els *pipelines* i el script *runner* per l'operació de selecció.
  - select_project_batch:generator.py: conté el codi per a generar els *pipelines* i el script *runner* per l'operació de selecció i projecció.
  - umount_local_disk_loop_devices.sh: : permet desmuntar el disc *loop* per a poder mesurar les estadístiques d'entrada i sortida del disc local.
- src/main: conté tot el codi font del *Testbed*.
- .gitignore: permet ignorar arxius a l'hora de crear *commits* amb git.
- LICENSE: és l'arxiu de llicència
- README.md: és una petita explicació de com funciona el *Testbed*.
- mvnw: és un arxiu de Maven.
- mvnw: és un arxiu de Maven.
- pom.xml: és un arxiu que conté totes les dependències per a Maven.