Títol: Tècniques per a la traducció d'esquemes relacionals a no relacionals

Volum: 1/1
Alumne: Sergi Nadal Francesch

Director: Xavier Castillejo Nache
Ponent: Alberto Abelló Gamazo
Departament: ESSI
Data: 19/06/2013

DADES DEL PROJECTE

Títol del Projecte:	Tècniques per a la traducció d'esquemes relacionals a no relacionals
Nom de l'estudiant:	Sergi Nadal Francesch
Titulació:	Enginyeria en Informàtica
Crèdits:	37.5
Director:	Xavier Castillejo Nache
Ponent:	Alberto Abelló Gamazo
Departament:	Enginyeria de Serveis i Sistemes d'Informació

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President:	Alberto Abelló Gamazo
Vocal:	Juan José Navarro Guerrero
Secretari:	Oscar Romero Moral

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:
Data:

Vull agrair tant a la meva família com a la Neus la seva paciència, comprensió i el suport donat.

D'altra banda, també vull agrair a l'Alberto Abelló la seva incansable i sempre eficaç ajuda.

Índex

Capítol 1

Esquemes relacionals i no relacionals

La primera part d'aquest capítol fa una mirada enrera en la història de les bases de dades relacionals i la seva evolució, d'aquesta forma ens situarem en el context del projecte. També es veuran les motivacions i es justificarà què ha dut a les organitzacions a fer la transició d'ús d'esquemes relacionals a no relacionals. Finalment es llistaran les característiques més rellevants dels esquemes no relacionals, també com quines avantatges i inconvenients tenen respecte dels seus predecessors.

1.1 Una mirada enrera

Durant la dècada dels 60, a mesura que l'ús d'ordinadors esdevenia més econòmic per a les organitzacions arribaven les bases de dades. Aquestes tenien un objectiu molt simple: recollir un conjunt de dades estructurat per tal que posteriorment pugui ser explotada per qualsevol procés que requereixi d'aquesta informació.

Dos models de base de dades van esdevenir populars: models jeràrquics i models en xarxa. Els models jeràrquics estructuren les dades en forma arborescent o pare-fill.
Les relacions són del tipus *1:N*, això implica que un pare pot tenir un o més fills mentre que un fill tindrà un únic pare.

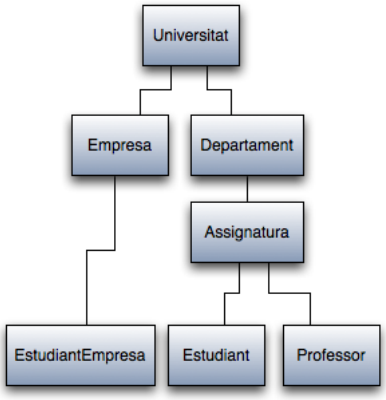


Figura 1.1: Esquema de dades jeràrquic.

- Avantatges

  1. **Simplicitat:** La relació entre capes de l'estructura jeràrquica és conceptualment simple.
  2. **Eficiència:** Representació i recorregut eficient en estructures jeràrquiques.
  3. **Integritat:** Donat que està basat en la relació pare-fill, sempre existeix la relació entre nodes.

- Inconvenients

  1. **Complexitat d'implementació:** tot i ser fàcil de dissenyar, és complex d'implementar degut a l'ús de punters.
  2. **Manca d'independència estructural:** si es fa algun canvi en l'estructura del model, és necessari fer canvis en tots els processos que passin per aquesta capa modificada.
  3. **Cerques:** les operacions de cerca que no segueixen l'estructura jeràrquica són altament ineficients.
  4. **Duplicitat de dades:** per a representar relacions *N:N* s'ha de repetir el valor per tants pares com vulgui tenir.

Pel que fa al model en xarxa, a diferència del jeràrquic, permet modelar relacions *1:N* i *N:N* ja que considera el model com un graf on els nodes són els valors emmagatzemats i les arestes les relacions entre ells.

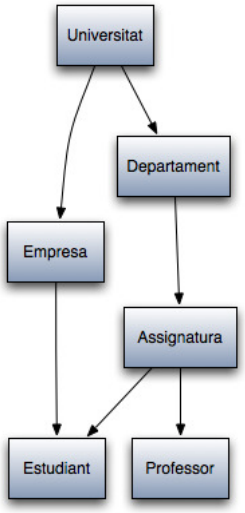


Figura 1.2: Esquema de dades en xarxa.

- Avantatges
  1. **Simplicitat:** al igual que en el model anterior, aquest és conceptualment simple.
  2. **Tipus de relacions:** Millora el model jeràrquic permetent les relacions *N:N*.
  3. **Integritat:** Donat que està basat en la relació pare-fill, sempre existeix la relació entre nodes.
- Inconvenients
  1. **Complexitat:** alt ús de punters.
  2. **Eficiència de la màquina:** una implementació física que funciona eficientment per un tipus de xarxa, pot donar molt mal rendiment per un altre tipus.
  3. **Mantenibilitat:** modificar relacions pot obligar-nos a reorganitzar les dades a nivell físic.
  4. **Falta de robustesa:** una fallada en el sistema pot deixar referències mal assignades que provoqui un estat inconsistent.

Tot i suposar una millora respecte les bases de dades jeràrquiques, les de xarxa seguien sense oferir solucions a les organitzacions que les volien explotar en forma de consultes dinàmiques, això era degut al baix nivell al que

treballaven.

A finals dels anys 60, Edgar F. Codd va presentar "*A Relational Model of Data for Large Shared Data Banks*", en aquest article es proposava un nou model teòric de base de dades. Aquest pretenia solucionar problemàtiques que sorgien amb l'evolució de la tecnologia i la capacitat d'emmagatzemar majors volums de dades, que fins llavors els models de bases de dades existents eren incapaços de resoldre. Es proposava l'ús de conceptes innovadors com ara:

- Concepte de taula amb dades estructurades per tipus (columnes) i tuples o registres (files).
- Independència de dades entre hardware i la implementació tecnològica de l'emmagatzematge
- Regles d'integritat com ara claus primàries o foranes.
- Navegació automàtica entre conjunts de dades.

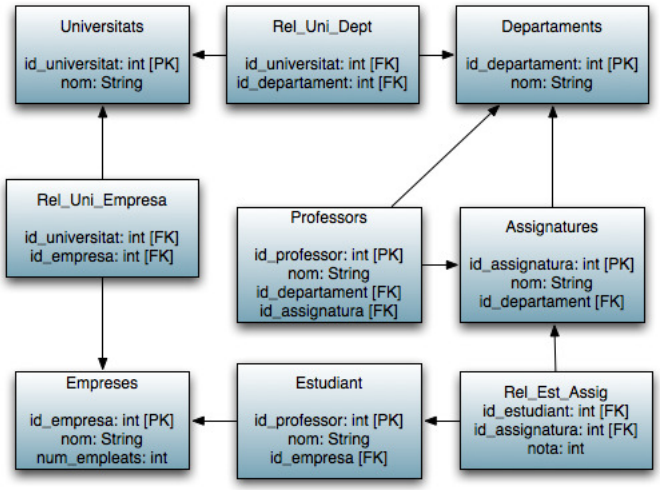


Figura 1.3: Esquema de dades relacional.

Poc temps després es va definir el model E-R el qual permet descriure de forma abstracta una base de dades relacional en termes d'entitats i relacions entre elles. Una entitat és un element que existeix en el domini de la base de dades i és únicament identificable.

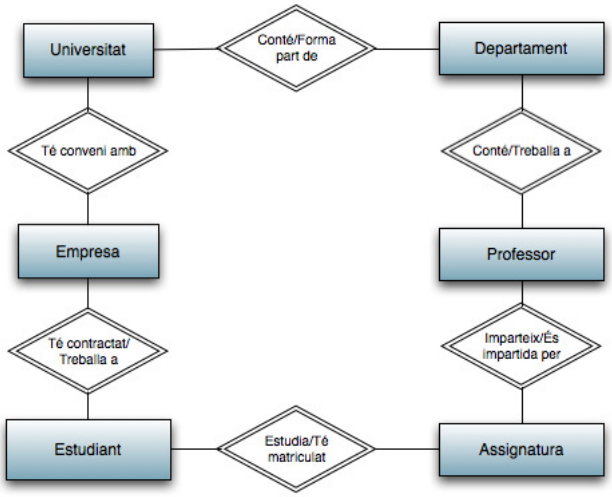


Figura 1.4: Diagrama E-R.

Per tal d'explotar aquestes bases de dades relacionals va sorgir el llenguatge SQL, aquest permet l'explotació d'aquest model relacional en forma de consultes amb certa relació al llenguatge natural, les quals permeten inserir, modificar o eliminar informació d'una base de dades relacional. Aquest fet va causar el boom de les bases de dades relacionals, van sorgir gran quantitat de gestors de bases de dades (SGBDs) els quals es basaven en aquest model relacional, cada nova versió pretenia millorar l'anterior i aconseguir millorar respecte la competència.

```
SELECT Professors.nom
FROM Professors JOIN Departaments on
    Professors.id_departament = Departaments.id_departament
WHERE Departaments.nom = "MA2";
```

Figura 1.5: Consulta SQL.

Tot SGBD que treballi sobre una base de dades relacional, a més a més satisfà quatre propietats englobades sota l'acrònim ACID. L'assoliment d'aquestes propietats ens assegura que les transaccions siguin fiables.

- ***Atomicity***: l'atomicitat d'una transacció ens assegura que al acabar s'han executat i finalitzat tots els passos del que estava composta, o bé no se n'ha executat cap.

- ***Consistency***: la consistència assegura que la base de dades està en un estat consistent al iniciar i al finalitzar qualsevol transacció. Una base de dades està en un estat consistent si es satisfan totes les regles d'integritat de la mateixa.

- ***Isolation***: l'aillament es refereix a que l'execució d'una transacció no afectarà a les altres. Això permet que dues transaccions treballin sobre el mateix subconjunt de dades sense errors.

- ***Durability***: la durabilitat d'una transacció ens assegura que un cop finalitzada correctament aquesta és persistent i no es pot desfer.

Quaranta anys després de la seva creació, el model relacional segueix sent el model per excel·lència en bases de dades, tot i la evolució tecnològica. A dia d'avui, els SGBD incorporen funcionalitats que complementen l'emmagatzematge de dades a part de la satisfacció de les propietats ACID, com ara: seguretat, integritat, recuperació o control de concurrència entre altres. Tots aquests afegits impliquen una càrrega extra de treball, cosa que provoca que per volums de dades molts grans, de l'ordre de petabytes (aproximadament 10^{15}), la seva gestió i explotació es converteix en inviable.

1.2 *Big Data* i *NOSQL*

Actualment les organitzacions no emmagatzemen el mateix tipus ni volum de dades com ho feien als inicis del model relacional, clarament les magnituds de dades són molt majors ja que es calcula que la capacitat d'emmagatzematge de dades mundial és duplicada cada quaranta mesos des dels anys 80. Aquests grans conjunts de dades són coneguts com a *big data*. Aquest terme fa referència a conjunts de dades els quals el seu tamany no permet el seu procés, emmagatzematge ni anàlisi per part dels mètodes convencionals[1].

Els conjunts de dades considerats *big data* compleixen tres característiques principals:

1. **Volum**: alt volum de dades, actualment de l'ordre de petabytes però en evolució a ritme exponencial.

2. **Velocitat**: la velocitat a la que es generen aquestes dades és molt elevada, per exemple dades provinents de sensors.

[1]En aquest projecte per a mètode convencional entenem el model relacional i totes les eines que l'exploten.

3. **Varietat**: actualment un conjunt de dades no té un únic origen, les dades poden arribar des d'origens estructurats (bases de dades relacionals) i de no estructurats (correus electrònics, fulls de càlcul, etc).

El problema rau en que els SGBD que treballen sobre bases de dades relacionals, no són capaços d'oferir l'assoliment d'aquestes característiques. Aquests volen mantenir totes les prestacions comentades prèviament, que al combinar-ho amb grans volums de dades provoca la pèrdua de rendiment o directament la impossibilitat de funcionar. Les dues principals causes de la incompatibilitat de *big data* amb els SGBD relacionals clàssics són:

- **Només són escalables verticalment**: només es pot millorar el *hardware* del servidor on funciona el SGBD, però aquesta millora té un límit físic.
- **No són distribuïbles**: les dades només resideixen en un únic servidor i no es pot compartir amb d'altres.

Val a dir que en els últims anys els fabricants d'SGBDs relacionals han potenciat la distribució i escalabilitat horitzontal dels seus sistemes, això no vol dir però que de moment siguin comparables els amb sistemes *NOSQL*.

Per a solucionar la problemàtica sorgida amb *big data* i la seva incompatibilitat amb SGBD relacionals, ha sorgit el concepte *NOSQL*, el qual consisteix en diferents models de bases de dades no relacionals que satisfan els problemes d'escalabilitat, rendiment, emmagatzematge i distribució. La idea principal de *NOSQL* és que cada cas d'ús necessita un tipus de base de dades concret, mentre que algunes funcionalitats poden ser crítiques, d'altres es poden relaxar o fins i tot arribar a eliminar.
En el gran ecosistema de bases de dades *NOSQL* hi ha quatre grans famílies:

1. **Clau-valor**: també coneguda com a taula *hash*, emmagatzema valors associats a una clau.
    - Característiques:
        - El cost d'accés a dades per clau és òptim: $\Theta(1)$.
        - Manca total d'esquema. L'estructura lògica de la base de dades prové de l'aplicació.
        - No dona bon rendiment per consultes complexes.
        - L'espai ocupat per metadades és mínim.
        - Les dades són emmagatzemades en format binari o BLOB.
    - Exemples:

  - Voldemort
  - Scalaris
  - Tokyo Cabinet
  - Redis
  - MemcacheDB

2. **Documentals**: similar a les clau-valor però no emmagatzema les dades en format BLOB sinó en un tipus estructurat com ara XML o JSON.

   - Característiques:
     - El cost d'accés a dades per clau és òptim: $\Theta(1)$.
     - La base de dades disposa d'una API o un llenguatge de consultes per obtenir els documents.
     - Es poden fer cerques no únicament per clau, a diferència de clau-valor.
     - Modelat de dades comprensible per al desenvolupador.
     - Alt consum de memòria física.
   - Exemples:
     - CouchDB
     - eXist
     - MongoDB
     - RavenDB

3. **Orientades a columna**: en aquest grup tenim dues subcategories les quals es diferencien per la manera d'emmagatzematge.

   (a) **Mapa multidimensional ordenat**: les dades s'emmagatzemen en un mapa multidimensional el qual és accedit per famílies de columnes.

   - Característiques:
     - Un valor s'identifica per taula, familia de columna i *timestamp*.
     - Les dades són separades per famílies de columnes, però cada una d'aquestes pot emmagatzemar múltiples columnes.
     - Escriure una tupla requereix multiples accessos.
     - Permet la paral·lelització d'operacions mitjançant el *framework* MapReduce.

  - Esquema més estàtic que les altres solucions.
  - Facilment escalable horitzontalment.
  - Minimitza l'accés a disc per consultes.
- Exemples:
  - HBase
  - Hypertable
  - Cassandra

(b) **Emmagatzematge per columnes**:

- Característiques:
  - Manté un esquema molt similar al relacional.
  - Les columnes d'una tupla són emmagatzemades per separat.
  - Suporta l'ús del llenguatge SQL.
  - Està optimitzat per les lectures, les actualitzacions són altament costoses.
- Exemples:
  - C-Store
  - Sybase IQ
  - Vertica

4. **Graf**: similar a les bases de dades en xarxa comentades previament, però disposen d'un motor tecnològic molt més avançat i potent.

   - Característiques:
     - Esquema lliure.
     - Les relacions són físicament implementades, les operacions de navegació són senzilles i tenen bon rendiment.
     - Penalitza les operacions d'agregació de dades.
   - Exemples
     - Neo4J
     - Infinite Graph
     - HypergraphDB
     - FlockDB

1.3 No tot són beneficis

Fins ara hem parlat del beneficis i millores que aporta *NOSQL*, però a l'hora de treballar amb sistemes distribuïts els problemes de comunicació i sincronització augmenten, és per això que totes les propietats ACID no és poden satisfer degut a que les dades són dividides a tots els nodes del cluster. El problema és que les propietats ACID obliguen a bloquejar les dades amb que les transaccions treballen, si aquesta dada està distribuïda estem bloquejant múltiples servidors. Aquest fet fa que les aplicacions que requereixen alta disponibilitat de dades perdin rendiment.

L'exemple anterior es pot generalitzar amb el teorema de CAP, aquest diu que hi ha tres requisits bàsics per a aquests tipus de sistemes distribuïts:

- ***Consistency***: similar a la propietat de consistència d'ACID. En aquest cas es refereix a que no es permet més d'un valor per el mateix conjunt de dades al mateix moment, és a dir, tots els nodes disposen de la mateixa dada al mateix moment.
- ***Availability***: el sistema sempre està disponible per rebre sol·licituds, no necessariament la resposta ha de ser satisfactoria.
- ***Partition tolerance***: el sistema ha de ser capaç de seguir funcionant encara que un o més d'un dels nodes fallin.

A més a més, aquest teorema estipula que és impossible disposar dels tres requisits a la vegada, se n'ha de prescindir d'un d'ells. És per això que els diferents productes disponibles de bases de dades no relacionals s'han decantat per eliminar un dels requisits o un altre.
En el cas d'aquest projecte, treballarem sobre **HBase** la qual afavoreix la consistència i la tolerància a particionament sobre la disponibilitat. Això ho fa mitjançant mecanismes de replicació i distribució de les dades, en el cas que la dada demanada no sigui disponible HBase ens farà esperar.

Una altra de les mancances dels esquemes no relacionals és que degut a la seva simplicitat, no podem estar segurs si la estructura aplicada satisfarà els requisits demanats fins ben avançat el procés d'implementació.

Capítol 2

Objectius i desenvolupament

En aquest capítol es detallen els objectius del projecte i a continuació quin procediment es seguirà per assolir-los.

2.1 Objectius

El principal objectiu d'aquest projecte és determinar si existeixen determinades tècniques de conversió d'esquemes relacionals a no relacionals sota determinades condicions. Trobar-les ens permetria automatitzar certs processos de migració de bases de dades relacionals a *NOSQL*. Aquestes conversions han de fer que l'esquema sigui òptim de cara a temps d'execució d'un subconjunt de consultes.

D'altra banda, també es vol avaluar el rendiment d'un sistema relacional contra un de no relacional sota les mateixes condicions d'aplicació.

Lligant amb l'objectiu anterior, es vol aprendre a instal·lar i configurar un SGBD relacional i un de no relacional en una configuració de tipus *cluster*, és a dir, distribuït.

Finalment, l'últim objectiu, el qual el seu grau d'assoliment variarà en funció de l'assoliment dels altres dos, serà la construcció d'una guia per tal que sota unes determinades condicions ens digui quina és la millor conversió de relacional a no relacional.

2.2 Desenvolupament del projecte

Per tal d'assolir els tres primers objectius es treballarà amb MySQL Cluster com a sistema relacional, aquest disposa de mecanismes de distribució i escalabilitat horitzontal a la vegada que manté les propietats ACID. A més a més, és l'únic producte gratuït que ofereix aquestes funcionalitats.
Pel que fa al sistema no relacional, s'usarà HBase, actualment aquest és una de les opcions més robustes i extesa per les bases de dades no relacionals orientades a columna.

Com que ambdós sistemes han de funcionar sota les mateixes condicions es treballarà amb el mateix hardware, concretament 4 màquines (una master i la resta nodes de dades), totes aquestes cedides per la FIB. El sistema operatiu en totes elles serà *Ubuntu server* 32 bits.
Pel que fa a memòria, tan el master com la resta de nodes disposen de 2 GB de RAM.
Pel que fa a espai de disc, aproximadament es disposa de 200 GB en cada node. En total es disposa del voltant de 600 GB per cada tipus de sistema.

Per l'objectiu d'aprenentatge d'instal·lació i configuració, es documentarà el procés seguit pas a pas per a instal·lar ambdós sistemes. També s'adjuntarà a aquesta memòria un llistat dels errors que han sorgit i la seva respectiva solució.

El punt més important són les diferents proves de conversió d'un model relacional a diferents variants no relacionals. Per fer això es treballarà sobre un model en estrella amb quatre dimensions i una taula de fets. S'usa aquest i no un altre perquè es volen recrear les condicions utilitzades en un *data warehouse* (evidentment simplificat) per a ser explotat per una eina de *business intelligence*.
Aquest model es carregarà en MySQL i contindrà dades creades de forma aleatoria seguint diferents distribucions de probabilitat. A continuació, s'executaran un conjunt de consultes SQL i s'avaluarà el temps d'execució. Aquestes consultes tindran en compte diferents factors com ara: factor de selecció, nombre de joins o existència de subconsultes.
Al sistema no relacional es carregaran diferents variants de construcció de l'esquema. S'avaluaran el temps d'execució del mateix conjunt de consultes amb els mateixos volums de dades i es farà una comparació amb els resultats obtinguts en la versió relacional.

Finalment, l'últim objectiu s'aconseguirà a partir de l'anàlisi de les proves

realitzades i les conclusions extretes d'aquestes.

2.3 Planificació

La planificació inicial del projecte el situava com a inici, al setembre del 2012 amb el detall de tasques de la figura 2.1.

Aquesta planificació, va haver de ser modificada degut a les complicacions a l'hora d'instalar els sistemes en màquines virtuals, la clara manca de hardware i la incompatibilitat de dur a terme el desenvolupament del projecte amb altres activitats.
La segona planificació situava el projecte amb inici al febrer de 2013 i final juny de 2013, ja que gairebé s'havia de començar de nou amb les màquines cedides per la FIB. Aquesta segona planificació es detalla en la figura 2.2.

D'altra banda, tant en la planificació inicial com en l'informe previ del projecte, s'inclou la construcció d'una eina de conversió d'esquemes relacionals a no relacionals. En la planificació final no s'ha inclós per la manca de temps.

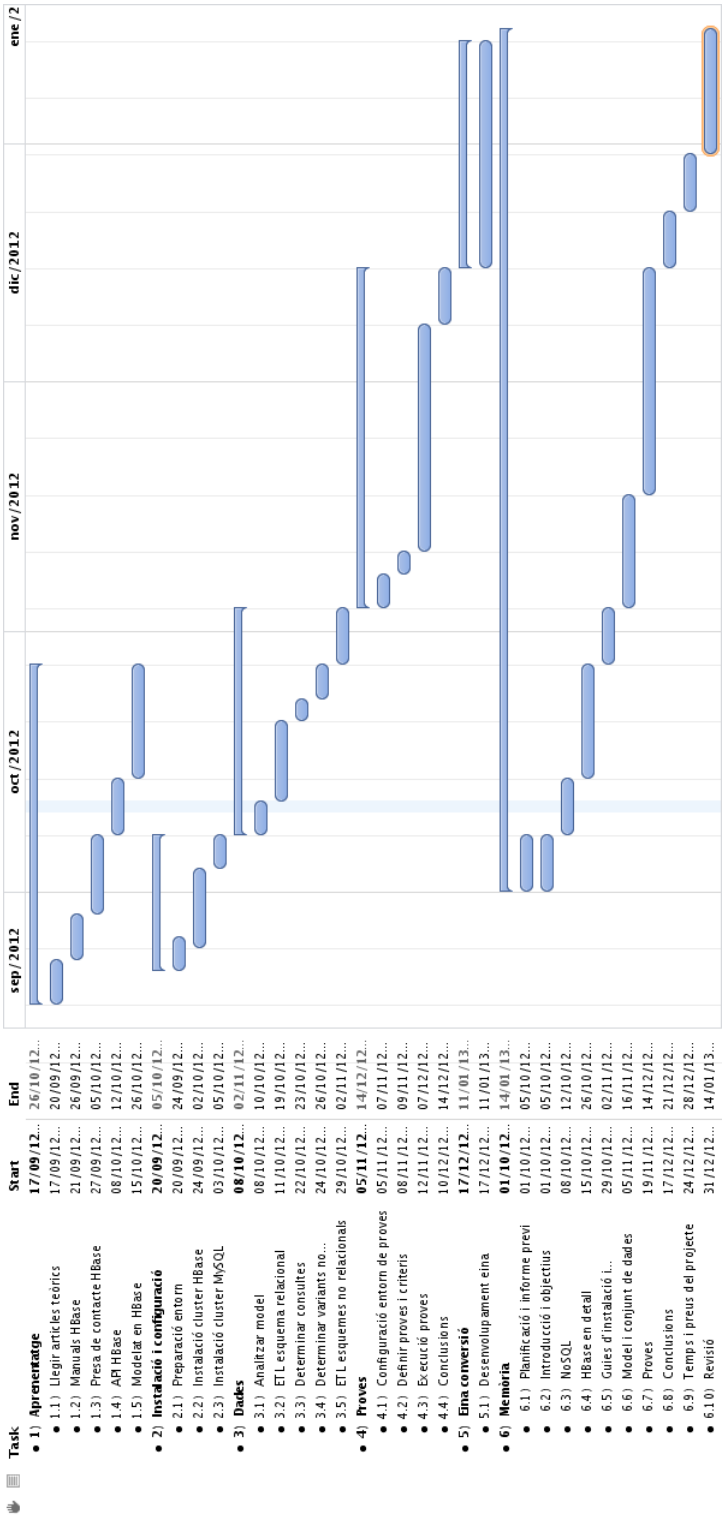


Figura 2.1: Planificació inicial del projecte.

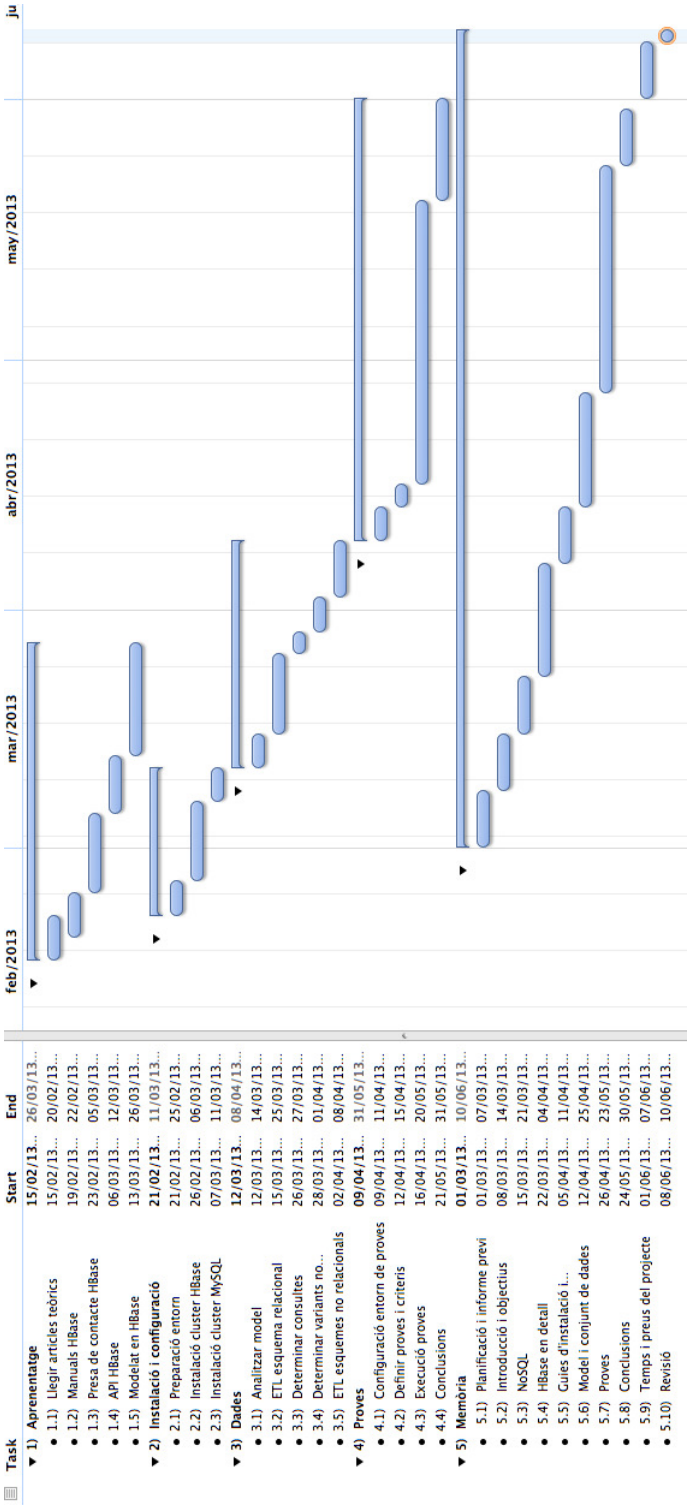


Figura 2.2: Planificació final del projecte.

2.4 Estudi econòmic

Tot el projecte s'ha pogut construir fent ús de software lliure i maquinari cedit per la facultat, així doncs no ha estat necessària la compra de cap llicència ni aparell.

Pel que fa als recursos humans, aquest projecte està pensat per desenvoluparse en 4 mesos a mitja jornada. Per tal de compensar els pics de feina tindrem en compte els 7 dies de la setmana en comptes dels 5 laborables, així doncs ens surten aproximadament unes 500 hores de feina.
El sou d'un estudiant en pràctiques en conveni amb la UPC és de 7'5€/hora, així doncs el cost del projecte resultaria en 3750€.

Capítol 3

Apache HBase: una introducció

En aquest tercer capítol es detallarà el funcionament, arquitectura i base. A continuació, es descriuran els principals components dels que es composa i que també seran utilitzats en el projecte, aquests són: Hadoop, HBase, MapReduce, ZooKeeper i Hive. S'explicarà com interactuen entre ells i quin és el seu rol.

3.1 Què és HBase?

HBase es sustenta sobre quatre pilars principals els quals queden units amb la següent frase dels propis desenvolupadors:

HBase is an **open source**, **distributed**, **sorted map datastore** *modeled after* **Google's Big Table**.

A continuació detallarem quin paper té cada una i el perquè de ser en HBase.

3.1.1 Open source

HBase és un projecte de codi obert pertanyent a la fundació Apache. Aquesta s'encarrega de donar suport a múltiples projectes de codi obert, així com proporcionar unes llicències d'ús i distribució.
HBase es basa en una llicència Apache 2.0, aquesta dona lliure accés al projecte, distribució i ús. No té cap cost afegit.

El projecte no està desenvolupat en una sola organització, sinó que hi participen desenvolupadors de diferents organitzacions com ara: Facebook, Cloudera, StumbleUpon o Trend Micro entre altres. Això permet fer un software

de més qualitat al ser desenvolupat per col·laboradors de diferents equips amb experiència en diferents sectors.

3.1.2 Distributed

El sistema funciona de forma totalment distribuida gràcies a Apache ZooKeeper, aquest s'encarrega del balanceig de càrregues, repartiment de dades i el seu funcionament i arquitectura es detallaran més endavant.
Actualment el límit de servidors per a que el cluster funcioni correctament de forma distribuïda és de 700.

HBase accepta el funcionament en dos modes diferents: *standalone* i *distributed*:

1. **Standalone**: és el mode de funcionament per defecte, no distribueix dades sinó que treballa sobre el sistema de fitxers local. Tant el servei d'HBase com ZooKeeper són executats sota el mateix procés local de JVM.

2. **Distributed**: el mode distribuït pot ser dividit en dos categories, pseudodistribuït i completament distribuït. Tant un com l'altre requereix una instància en funcionament d'HDFS, més endavant en aquest mateix capítol també es detallarà el funcionament d'HDFS.

    - **Mode pseudodistribuït**: aquest mode és equivalent al mode distribuït però funciona en un únic servidor. Només s'utilitza per fer proves d'HBase.
    - **Mode totalment distribuït**: necessitarem més d'un servidor amb HDFS instal·lat en mode distribuït. A HBase és important activar la propietat `hbase.cluster.distributed`, per afegir nous servidors al cluster simplement cal afegir el *hostname* d'aquests al fitxer *region servers*. Aquest és el nom amb el que és denomina als nodes de dades en HBase.

HBase permet fer escalat lineal, això significa que cada nou node (o en termes d'HBase *region*) que afegim al cluster farà augmentar el rendiment, la capacitat i el nombre d'operacions per segon. Per exemple, un cluster de 50 nodes haurà de ser aproximadament 50 vegades més potent que un de simple.

3.1.3 Sorted Map Datastore

HBase vol diferenciar-se del concepte de base de dades anomenant-se datastore. Un datastore no conté funcionalitats pròpies de bases de dades com ara transaccions o esquema. Tampoc no manté integritat referencial entre dades, sinó que emmagatzema les dades en fitxers plans.

Les dades estan emmagatzemades en una estructura de tipus diccionari ordenat.
La unitat més bàsica és la columna. Una o més columnes formen una fila que és únicament identificada per una *row key* o clau primària. Un conjunt de files formen una taula, de les quals en poden haver múltiples. Cada columna pot tenir múltiples versions on cada valor diferent s'emmagatzema en una cel·la.
Les files estan ordenades lexicogràficament a nivell de byte per la seva *row key*

```
hbase(main):008:0> scan 'taula'
ROW                                   COLUMN+CELL
 rk1                                  column=familia_columnes:colA, timestamp=1357657566405, value=valor
 rk1                                  column=familia_columnes:colB, timestamp=1357657577523, value=valor
 rk2                                  column=familia_columnes:colA, timestamp=1357657594118, value=valor
2 row(s) in 0.7800 seconds
```

Figura 3.1: Exemple de llistat de dades obtingudes d'HBase.

Com hem dit abans, les files estan composades per columnes i aquestes al seu temps estan agrupades per *column families* o families de columnes. Totes les columnes agrupades en una família de columnes s'emmagatzemen conjuntament en el mateix fitxer a disc, això ens permet fer agrupacions semàntiques que sabem que seran consultades conjuntament molt sovint. Aquests fitxers on s'emmagatzemen s'anomenen *HFile*.
Les famílies de columnes s'han de definir en el moment que es crea la taula, i no conve modificar-les quan el sistema està en producció. El nombre de famílies de columnes definides no hauria de ser més de 5, a partir d'aquest el rendiment pot baixar dràsticament. Això és degut a que HBase escriu a disc totes les famílies alhora, en el cas que una família es modifiqui molt sovint i les altres no, però són molt grans, carregarà en excés de forma innecessària el sistema.

Cada cel·la pot tenir múltiples versions, això és possible perquè incorporen un *timestamp* afegit pel sistema o per l'usuari. Les diferents versions d'una cel·la s'ordenen per ordre d'inserció d'aquesta forma llegirem els valors

nous primer.

Les següents imatges agrupen tots aquests conceptes explicats en una visió lògica i física de les dades.

Row key	Dades
lauramp	info:{'edat':'33'} departament: {'ESSI':'professor'}
joancf	info:{'edat':'47', 'ciutat': 'Barcelona'} departament:{'MA2':'director'@ts=2012, 'MA2':'professor' @ts=2013, 'LSI':'professor'

Figura 3.2: Representació lògica en forma de registres. Els valors *info* i *departament* són famílies de columnes, *ts* és el *timestamp* per les múltiples versions.

Família de columnes: info

Row key	Column key	Timestamp	Valor
joancf	info:ciutat	1273516197868	Barcelona
joancf	info:edat	1043871824184	47
lauramp	info:edat	1273616297446	33

Família de columnes: departament

Row key	Column key	Timestamp	Valor
joancf	departament:LSI	1273871823022	professor
joancf	departament:MA2	1300062064923	professor
joancf	departament:MA2	1293388212294	director
lauramp	departament:ESSI	1273616297446	professor

Figura 3.3: Representació física de com s'emmagatzemen les dades en disc. Les cel·les s'ordenen per *row key*, *column key* i *timestamp* de forma descendent. Els valors de *timestamp* són en milisegons en format unix.

Com es veu a la figura anterior, les famílies de columnes s'emmagatzemen en fitxers separats. Això ens permet configurar diferents propietats per família de columna com ara:

- Compressió de dades: no comprimir, GZip, BZ2 o LZO.
- Nombre de versions emmagatzemades per cel·la.
- Prioritat d'ús de memòria cau.

3.2 Google's Big Table

HBase es basa en l'especificació de BigTable realitzada per Google l'any 2006. Algunes de les principals diferències entre HBase i BigTable són:

- HBase emmagatzema els *timestamps* en milisegons mentre que BigTable ho fa en microsegons.
- HBase usa HDFS mentre que BigTable usa GFS.
- HBase no mapeja fitxers de disc a memòria, cosa que sí està disponible a BigTable.
- BigTable incorpora el concepte *locality groups*, això permet agrupar famílies de columnes conjuntament i aplicar propietats per grups. Donat que a BigTable tota la informació s'emmagatzema en el mateix fitxer, això és útil quan diferents famílies de columnes s'accedeixen conjuntament. En el cas d'HBase, les famílies de columnes s'emmagatzemen per separat.
- HBase té una opció per descartar el *commit log*, per tal de millorar l'eficiència.

3.3 Interactuar amb HBase

HBase proporciona quatre interfícies per accedir i interactuar-hi. La opció més avançada i popular és la API Java, tot i que també hi ha disponibles: serveis web tipus REST, Apache Thrift, Apache Hive/Pig. Totes aquestes solucions acaben rebent crides en un determinat format, però la comunicació amb HBase la fan mitjançant l'API Java, això és totalment transparent per qui fa la crida.

3.3.1 API Java

Totes les classes necessàries per treballar amb HBase es troben en el paquet `org.apache.hadoop.hbase`, per exemple per demanar informació d'una taula concreta necessitem la classe `org.apache.hadoop.hbase.HTable`.

En aquest capítol no detallarem totes les operacions disponibles, sinó que es farà un petit resum de les operacions tipus CRUD.

put

L'operació `put` és l'encarregada d'emmagatzemar informació en HBase, aquesta rep com a paràmetre una instància de la classe `Put` la qual s'inicialitza amb una *row key*. A partir de llavors podem incorporar informació de múltiples famílies de columnes en la mateixa instància de `Put` perquè insereixi o incorpori les dades en la fila en cas que ja existeixi.

El següent codi és un exemple de l'ús de `put` amb les dades anteriors.

```
1 Configuration conf = HBaseConfiguration.create();
2 HTable table = new HTable(conf, "empleats");
3 Put put = new Put(Bytes.toBytes("lauramp"));
4 put.add(Bytes.toBytes("info"), Bytes.toBytes("edat"), Bytes.toBytes("33")
    );
5 put.add(Bytes.toBytes("departament"), Bytes.toBytes("ESSI"), Bytes.
    toBytes("professor"));
6 table.put(put);
```

Com es pot veure en tot l'exemple s'utilitza la classe `Bytes`, aquesta proporciona operacions per a convertir a tipus *byte* tots els tipus primitius de Java i fer la operació inversa.
En l'exemple també s'utilitza la classe `Configuration`, aquesta al crear-ne una instància llegeix el fitxer ***`hbase-site.xml`*** per obtenir informació del *cluster*.

Per a emmagatzemar múltiples versions d'una determinada columna no hem de fer cap pas diferent dels descrits previament, el propi sistema s'encarrega d'incorporar la nova informació amb el *timestamp* que li pertoqui. Per al correcte funcionament del sistema de versions és necessari tenir el rellotge de tots els sistemes del *cluster* sincronitzats.
Per defecte HBase emmagatzema tres versions d'una columna, però és un valor parametritzable.

`get`

Per tal d'obtenir informació emmagatzemada utilitzarem l'operació `get`. De forma similar a l'operació `put` haurem de proporcionar-li una instància de la classe `Get` la qual s'inicialitza a partir d'una *row key*. Una operació de tipus `get` està lligada a una fila concreta però pot retornar qualsevol nombre de columnes o cel·les contingudes.

Podem indicar quina informació obtenim com a resultat de la consulta mitjançant les operacions:

- `addFamily(byte[] family)`: s'obté la família de columnes indicada. Pot ser cridada múltiples vegades.
- `addColumn(byte[] family, byte[] col)`: igual que la operació anterior però ens permet acurar la cerca a una columna concreta.
- `setTimeRange(long minStamp, long maxStamp)`: s'obtenen cel·les en que el *timestamp* estigui comprés entre *minStamp* i *maxStamp*.
- `setTimeStamp(long timestamp)`: s'obtenen les cel·les que tenen exactament el *timestamp* indicat en el paràmetre.
- `setMaxVersions()`: es limita el nombre de versions obtingudes al màxim permés per HBase.
- `setMaxVersions(int maxVersions)`: es limita el nombre de versions al paràmetre *maxVersions*.

El següent codi exemplifica l'ús més senzill de `get`.

```
1 Configuration conf = HBaseConfiguration.create();
2 HTable table = new HTable(conf, "empleats");
3 Get get = new Get(Bytes.toBytes("lauramp"));
4 get.addColumn(Bytes.toBytes("info"), Bytes.toBytes("departament"));
5 Result result = table.get(get);
6 byte[] val = result.getValue(Bytes.toBytes("info"),Bytes.toBytes("
    departament"));
7 System.out.println("Departament de lauramp: " + Bytes.toString(val));
```

Un cop executada l'operació `get`, podem obtenir el resultat en una instància de la classe `Result`. Aquesta proporciona mètodes per accedir a totes les propietats de la fila trobada com ara: famílies de columnes, columnes, *timestamps*, etc.

`delete`

Finalment, l'operació `delete` ens permet eliminar informació emmagatzemada a HBase. Seguint la mecànica de les operacions anteriors, a partir d'un objecte de la classe `Delete`, la qual és inicialitzada a partir de la *row key* de la fila a eliminar, es podrà cridar.
Si s'intenta eliminar una versió la qual no existeix, no passa res.

De la mateixa forma que `get`, podem aplicar filtres per les diferents propietats disponibles en una fila:

- `deleteColumn(byte[] family, byte[] col)`: elimina la última versió de la columna indicada.
- `deleteColumn(byte[] family, byte[] col, long timestamp)`: elimina la versió la qual té un *timestamp* igual que el passat per paràmetre.
- `deleteColumns(byte[] family, byte[] col)`: elimina totes les versions de la columna indicada.
- `deleteColumns(byte[] family, byte[] col, long timestamp)`: elimina totes les versions de la columna indicada que tenen un *timestamp* menor o igual que el passat per paràmetre.
- `deleteFamily(byte[] family)`: elimina totes les versions de totes les columnes de la família de columnes indicada.
- `deleteFamily(byte[] family, long timestamp)`: elimina totes les versions de totes les columnes de la família de columnes indicada i que tenen un *timestamp* menor o igual que el passat per paràmetre.

El següent codi exemplifica un ús senzill de `delete`.

```
1  Configuration conf = HBaseConfiguration.create();
2  HTable table = new HTable(conf, "empleats");
3  Delete delete = new Delete(Bytes.toBytes("lauramp"));
4  delete.setTimestamp(1);
5  delete.deleteColumn(Bytes.toBytes("info"), Bytes.toBytes("edat"), 1);
6  delete.deleteColumns(Bytes.toBytes("info"), Bytes.toBytes("ciutat"));
7  delete.deleteColumns(Bytes.toBytes("departament"), Bytes.toBytes("ESSI"),
       2);
8  delete.deleteFamily(Bytes.toBytes("departament"));
9  table.delete(delete);
10 table.close();
```

3.4 Arquitectura

HBase està compost de diferents components independents, els quals es connecten entre ells i composen l'ecosistema HBase. Un representació gràfica a alt nivell seria la següent:

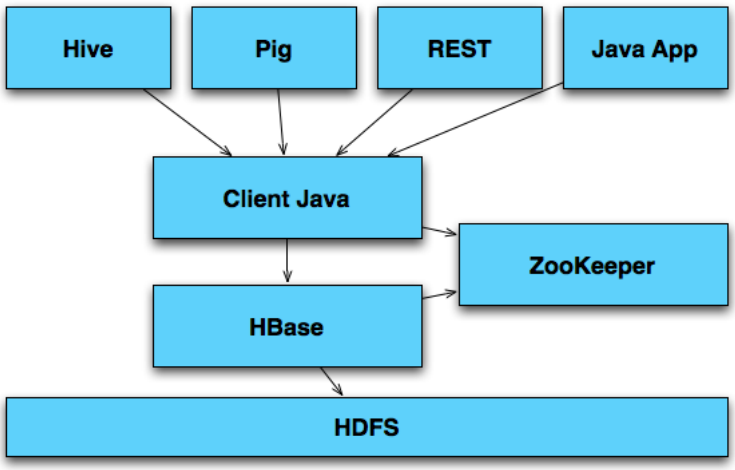


Figura 3.4: Esquema de l'arquitectura a alt nivell, la capa superior correspon als clients, la capa intermitja a la comunicació i funcionament de la base de dades i la última a l'emmagatzemament propi.

3.4.1 Hadoop Distributed File System

HDFS és un sistema de fitxers distribuït dissenyat per a adaptar-se al hardware. És el sistema de fitxers per defecte on HBase emmagatzema les dades ja que compleix tots els requisits necessaris. Alguns d'aquests requisits són:

- **Fallada de hardware**: Un *cluster* Hadoop pot estar compost de cents de nodes, la probabilitat de que qualsevol d'aquests falli és molt elevada. Així doncs, Hadoop disposa de mecanismes de detecció i recuperació automàtica de fallades.
- **Conjunts de dades grans**: HDFS està preparat per funcionar amb conjunts de dades de l'ordre de petabytes. Està optimitzat per al bon funcionament en aquests casos, cosa que pot penalitzar el funcionament en conjunts de dades petits.

- **Evitar moure dades**: un càlcul és molt més eficient si s'executa a prop de les dades amb les que opera. Quan el conjunt de dades és molt gran, això ens evita congestió en la xarxa i millora el rendiment general del *cluster*.

- **Portabilitat**: Hadoop està preparat per a ser portat a qualsevol plataforma la qual pugui executar una *JVM*.

Com s'ha anat comentant previament, HDFS té una arquitectura de tipus master/esclau. Un *cluster* HDFS està compost per un master o *namenode* i múltiples nodes esclaus o *datanodes*. Les dades estan distribuïdes en blocs en els *datanodes*, a partir d'aquests blocs es fa la replicació en cas que estigui activada.
L'administració de fitxers en cada bloc es manté des del *namenode*, el qual a partir de la replicació que necessiti el distribueix en més *datanodes* o menys.

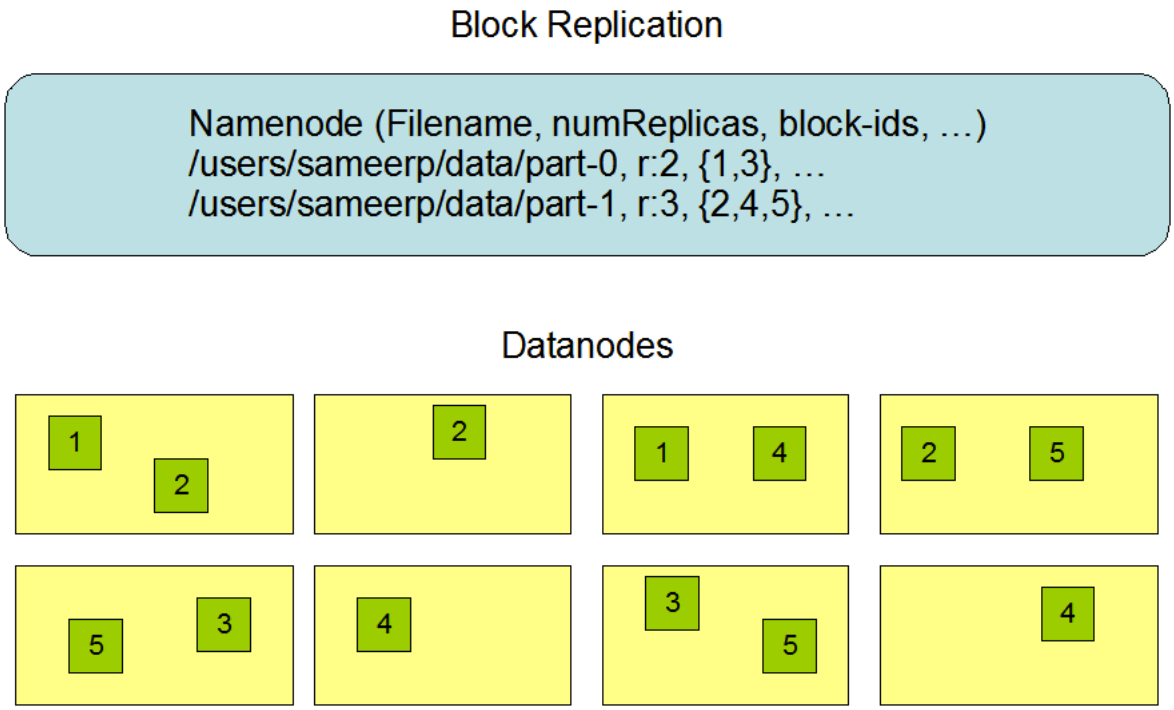


Figura 3.5: Esquema de distribució i rèplica de dos fitxers en HDFS.

MapReduce funciona sobre HDFS, aquest ens permet programar aplicacions que processen grans quantitats de dades de forma paral·lela.
Un *job* de MapReduce divideix les dades en diferents *chunks*, aquests són subconjunts independents del conjunt general. La sortida de la tasca `map`

s'ordena i es donada com a entrada per la tasca `reduce`. La següent imatge és un exemple on és útil usar MapReduce.

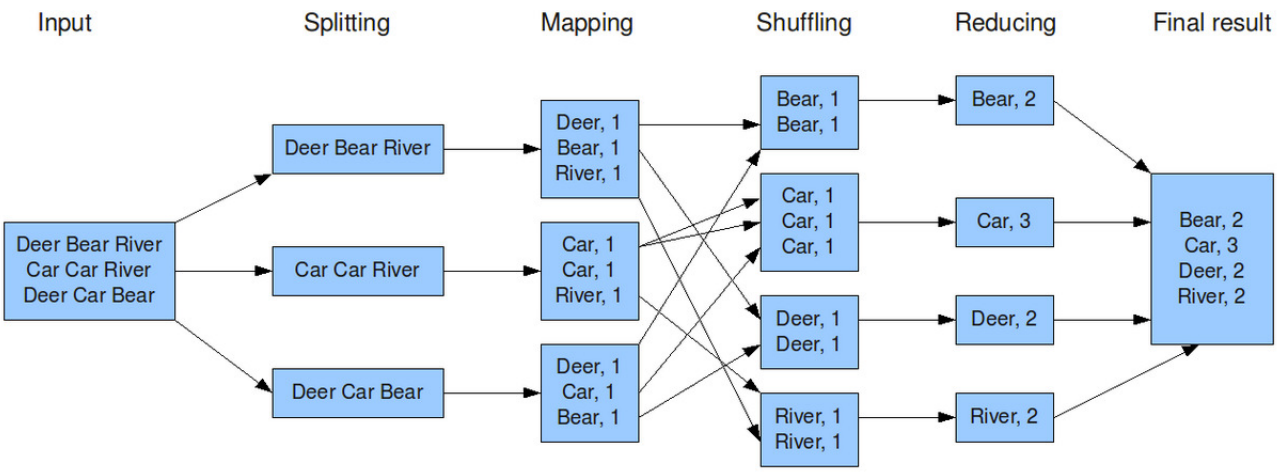


Figura 3.6: Contador de paraules usant MapReduce.

Com es pot veure en el codi del contador de paraules, sempre s'han de programar les dos tasques `map` i `reduce`. L'execució en paral·lel d'aquestes és transparent al programador.

```
1  public static class Map extends MapReduceBase implements Mapper<
       LongWritable, Text, Text, IntWritable> {
2        private final static IntWritable one = new IntWritable(1);
3        private Text word = new Text();
4
5        public void map(LongWritable key, Text value, OutputCollector<Text,
              IntWritable> output, Reporter reporter) throws IOException {
6              String line = value.toString();
7              StringTokenizer tokenizer = new StringTokenizer(line);
8              while (tokenizer.hasMoreTokens()) {
9                    word.set(tokenizer.nextToken());
10                output.collect(word, one);
11             }
12       }
13 }
14
15 public static class Reduce extends MapReduceBase implements Reducer<Text,
       IntWritable, Text, IntWritable> {
16       public void reduce(Text key, Iterator<IntWritable> values,
             OutputCollector<Text, IntWritable> output, Reporter reporter)
```

```
            throws IOException {
17            int sum = 0;
18            while (values.hasNext()) {
19                  sum += values.next().get();
20            }
21            output.collect(key, new IntWritable(sum));
22      }
23  }
```

3.4.2 ZooKeeper

ZooKeeper s'encarrega de la coordinació i comunicació entre nodes (en nomenclatura de ZooKeeper, *znodes*). Cada *region server* d'HBase es registra com a *znode* a ZooKeeper, d'aquesta forma el master usarà aquest llistat per a connectar amb els nodes disponibles.

ZooKeeper ofereix un sistema d'accés tipus sistema de fitxers, d'aquesta forma podem accedir a l'arrel i navegar pels fitxers dels que disposa cada *znode*.

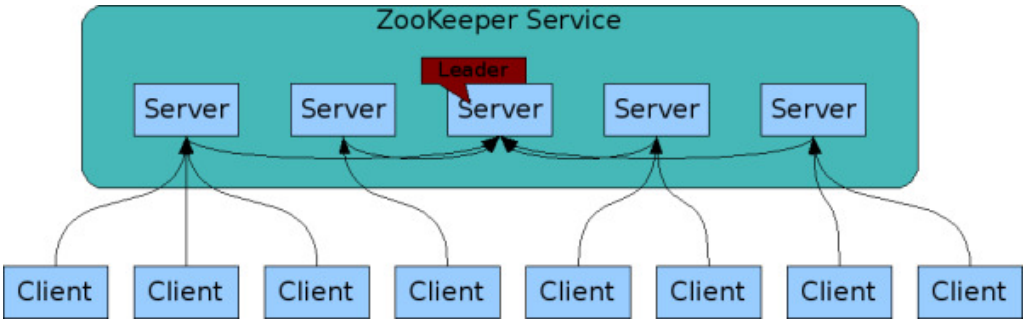


Figura 3.7: Funcionament de ZooKeeper, els servers són *znodes*.

3.4.3 Hive

Hive és una eina que permet executar consultar i analitzar grans conjunts de dades emmagatzemats en HDFS. Disposa d'un mecanisme per projectar aquestes dades en una estructura i posteriorment consultar-les mitjançant un llenguatge similar a SQL, HiveQL.

Una consulta HiveQL es transforma en una cadena de *jobs* de tipus MapReduce, els quals són totalment transparents al client.

Hive emmagatzema l'estructura d'HDFS en taules similars a SQL, aquestes metadades s'emmagatzemen en una base de dades Derby.

Hive també pot funcionar com a servidor, aquest fa córrer un servidor Thrift i permet rebre peticions de qualsevol client. Per tal de comunicar-nos amb el servidor de Hive des de Java, no hem d'aplicar cap configuració especial, només importar el *driver* JDBC. El següent codi és un exemple de connexió a Hive mitjançant Java:

```
1 Class.forName("org.apache.hadoop.hive.jdbc.HiveDriver");
2 Connection conn = DriverManager.getConnection("jdbc:hive://", "","");
3 Statement st = conn.createStatement();
4 st.execute("SELECT * FROM sales");
```

D'altra banda, Hive no només pot consultar fitxers d'HDFS sinó que també pot consultar taules HBase mitjançant `HBaseStorageHandler`. Aquestes taules de Hive no tenen perquè projectar tots els atributs disponibles en la família de columnes d'HBase, tot i que els atributs indicats sí que han d'existir. Un exemple seria, donada la següent taula en HBase:

```
create 'DEPARTAMENTS','info'
```

El mapeig el fariem a Hive de la següent forma, on aprofitem la *row key* com a clau primària de la taula.

```
CREATE EXTERNAL TABLE DEPARTAMENTS (id int, name string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,info:name"
TBLPROPERTIES("hbase.table.name" = "DEPARTAMENTS");
```

Finalment, Hive també disposa d'un client web el qual permet llançar consultes via web i navegar en l'esquema emmagatzemat en les metadades. Resumint, l'arquitectura de Hive gràficament queda de la següent forma:

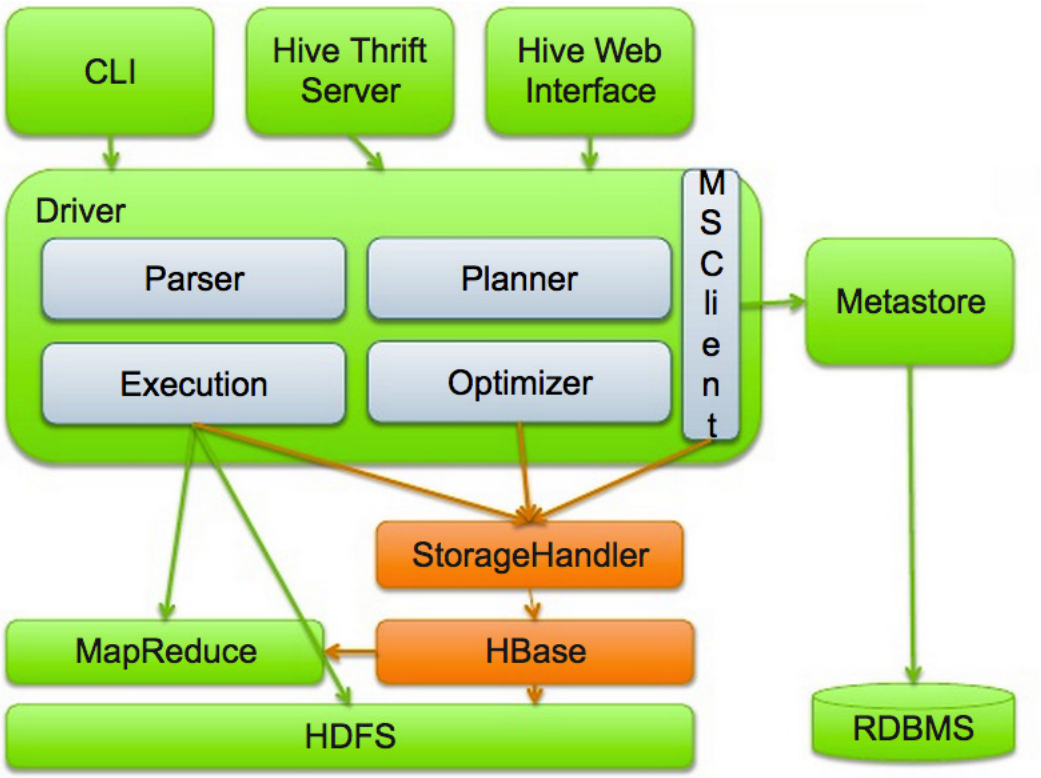


Figura 3.8: Arquitectura de Hive interactuant amb HBase.

Capítol 4

Configuració i instal·lació dels entorns

En la primera part d'aquest capítol es llistaran les especificacions dels servidors i del *cluster*, a continuació es descriurà el procés seguit per a instalar i configurar ambdós sistemes gestors de bases de dades que s'utilitzaran.

4.1 Especificacions del cluster

Per tal de poder treballar sota les mateixes condicions en les proves, usarem màquines amb el mateix hardware, totes aquestes cedides per la FIB. Com a sistema operatiu per a totes les màquines, s'usarà Ubuntu server ja que tant MySQL Cluster com HBase estan pensats per funcionar en un entorn de tipus Linux.

Pel que fa a les especificacions de hardware, totes les màquines disposen de 2GB de memòria RAM i un espai de 200 GB.

Finalment, com a configuració de xarxa totes les estaran connectades a la xarxa mitjançant cable. La següent imatge és un esquema del *cluster* a nivell de xarxa, s'han donat noms als servidors per a facilitar la configuració, concretament els nodes s'anomenen `begur`, `tossa` i `empuries`.

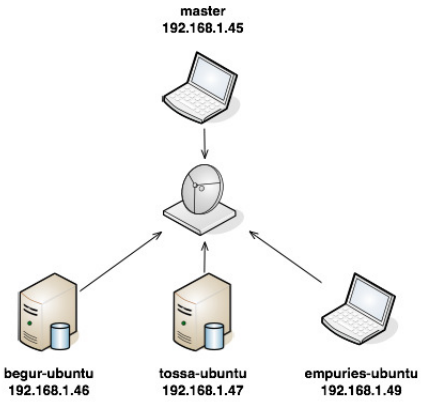


Figura 4.1: Esquema de xarxa utilitzat en el desenvolupament del projecte.

4.1.1 Configuració dels nodes

Abans d'instalar cap software de base de dades hem de preparar els servidors per al correcte funcionament del cluster.

Per tal de definir IPs estàtiques s'ha de modificar el fitxer `/etc/network/interfaces` i modificar el contingut, en el cas del master, per:

```
auto eth0
iface eth0 inet static
        address 192.168.1.45
        netmask 255.255.255.0
        network 192.168.1.0
        broadcast 192.168.1.255
        gateway 192.168.1.1
```

S'ha de modificar el *hostname* de cada node per el nom que vulguem que sigui identificat dins en el *cluster*. Aquest nom s'indica en el fitxer `/etc/hostname`.
En aquest projecte donarem el mateix nom als nodes de MySQL com els d'HBase que resideixen en una mateixa màquina física, així podrem aprofitar les configuracions entre ells.

Cada node ha de ser capaç d'identificar la resta a partir del seu *hostname*, s'ha de modificar el fitxer `/etc/hosts` i afegir el següent contingut:

```
127.0.0.1               localhost
```

```
192.168.1.45            master
192.168.1.46            begur-ubuntu
192.168.1.47            tossa-ubuntu
192.168.1.49            empuries-ubuntu
```

La comunicació entre el master i els esclaus es fa mitjançant SSH, per tal d'evitar que cada connexió sol·lici les credencials de cada node registrarem les claus en un fitxer.
Primer, haurem d'instal·lar el servidor i client SSH en tots els nodes.

```
sudo apt-get install openssh-server openssh-client
```

Un cop instal·lat el servidor SSH, amb la següent comanda generem un parell de claus RSA pública i privada.

```
ssh-keygen -t rsa
```

Des del node master afegirem a cada node esclau la clau pública del primer. Aquest procés s'ha de repetir amb totes les IPs dels nodes esclaus que tinguem:

```
cat /home/serginadalfrancesch/.ssh/id_rsa.pub >>
        /home/serginadalfrancesch/.ssh/authorized_keys
scp /home/serginadalfrancesch/.ssh/id_rsa.pub
        192.168.1.46:/home/serginadalfrancesch/.ssh/master.pub
```

D'altra banda, des de cada node esclau hem de permetre que el master s'hi connecti. Això ho farem mitjançant:

```
cat /home/serginadalfrancesch/.ssh/master.pub >>
        /home/serginadalfrancesch/.ssh/authorized_keys
```

Tant MySQL Cluster com HBase utilitzen Java per funcionar, així doncs imprescindible disposar d'ell. Amb la següent comanda el podrem instal·lar:

```
sudo apt-get install openjdk-7-jdk openjdk-7-jre
```

També és necessari afegir la ruta de Java al classpath. Modifiquem el fitxer `/etc/bash.bashrc` i al final del mateix hi afegim el següent contingut:

```
JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
export JAVA_HOME
PATH=$PATH:$JAVA_HOME/bin
export PATH
```

En aquest punt ja tenim els nodes preparats per a instal·lar ambdós sistemes que compararem en aquest projecte.

4.2 MySQL Cluster

4.2.1 Instal·lació

Per a començar amb la instal·lació de MySQL Cluster primer necessitem descarregar el paquet `libaio`, ho fem mitjançant la següent comanda:

```
sudo apt-get install libaio1 libaio-dev
```

També crearem un nou grup i un usuari d'Ubuntu específics:

```
sudo groupadd mysql
sudo useradd -g mysql mysql
```

MySQL Cluster es pot descarregar de la següent web: `http://www.mysql.com/get/Downloads/MySQL-Cluster-7.2/mysql-cluster-gpl-7.2.8-linux2.6-i686.tar.gz/from/http://cdn.mysql.com/`, aquest paquet serveix tan pel master com pels esclaus. En aquest projecte usarem la versió 7.2.8, la última versió estable disponible en el moment de realització de les proves.
Descomprimim i traslladem al directori d'instal·lació mitjançant:

```
sudo tar -C /usr/local/ -xzvf
        mysql-cluster-gpl-7.2.8-linux2.6-i686.tar.gz
sudo ln -s /usr/local/mysql-cluster-gpl-7.2.8-linux2.6-i686
        /usr/local/mysql
```

Canviem de ruta i executem l'*script* que crea les bases de dades de sistema. És important fer el canvi de ruta o l'*script* fallarà.

```
cd /usr/local/mysql
sudo ./scripts/mysql_install_db --user=mysql
```

Finalment donem els permisos necessaris per a poder gestionar els fitxers de dades.

```
cd /usr/local/mysql
sudo chown -R root .
sudo chown -R mysql data
sudo chgrp -R mysql .
```

En aquest punt ja tenim el sistema instal·lat, per a la comoditat a l'hora d'engegar el servidor i els clients canviarem els fitxers de ruta i assignarem permisos d'execució. Al master copiarem el servidor.

```
sudo cp /usr/local/mysql/bin/ndb_mgm* /usr/local/bin/
sudo chmod +x /usr/local/bin/ndb_mgm*
```

A la resta de nodes copiarem el client.

```
sudo cp /usr/local/mysql/bin/ndbd /usr/local/bin
sudo chmod +x /usr/local/bin/ndbd
```

4.2.2 Configuració

MySQL Cluster diferencia tres tipus de nodes en un *cluster*:

- Node d'administració (`ndb_mgm`): encarregat de tasques d'administració en els altres nodes com ara: configuració, inici o parada, còpies de seguretat. encarregat de tasques d'administració en els altres nodes com ara: configuració, inici o parada, còpies de seguretat.
- Node de dades (`ndbd`): node que emmagatzema les dades en el *cluster*. Podem tenir tantes rèpliques de les dades com nodes de dades tinguem.
- Node SQL (`mysqld`): node que accedeix a les dades.

En el nostre cas usarem el master com a node d'administració i SQL, la resta funcionaran com a nodes de dades.

En els nodes de dades i SQL hem de crear el fitxer `/etc/my.cnf`, aquest indica la ruta cap al node d'administració. Copiem el següent contingut al fitxer:

```
[mysqld]
ndbcluster
ndb-connectstring=master

[mysql_cluster]
ndb-connectstring=master
```

D'altra banda, al node d'administració hem de crear el fitxer de configuració per al *cluster*. S'ha de tenir en compte que MySQL Cluster reserva tota la memòria que se li indica en la configuració a tots els nodes. És important doncs, disposar de nodes amb especificacions físiques similars i que disposin de suficients recursos per a poder reservar els necessaris i seguir funcionant correctament. Aquesta informació s'emmagatzemarà al fitxer `/var/lib/mysql-cluster/config.ini`:

```
[ndbd default]
NoOfReplicas=1
DataMemory=1200M
IndexMemory=350M

[tcp default]
portnumber=2202

[ndb_mgmd]
hostname=master
datadir=/var/lib/mysql-cluster

[ndbd]
hostname=tossa
datadir=/usr/local/mysql/data

[ndbd]
hostname=begur
datadir=/usr/local/mysql/data

[ndbd]
hostname=empuries
datadir=/usr/local/mysql/data

[mysqld]
```

Les paraules definides entre claudàtors defineixen les seccions d'acord amb el rol que tingui el sistema. Algunes de les propietats més rellevants són:

- `[NoOfReplicas]`: defineix el tamany de cada grup de nodes. Un grup de nodes és un conjunt de nodes que emmagatzemem la mateixa informació.
- `[DataMemory]`: espai de memòria reservada per a emmagatzemar registres.
- `[IndexMemory]`: espai de memòria reservada per a índexs, per exemple claus primàries.

4.2.3 Arrancada

Per tal d'arrencar el sistema, un cop instal·lat i configurat, hem d'executar la següent comanda al node d'administració:

```
sudo ndb_mgmd --reload --configdir=/var/lib/mysql-cluster
        --config-file=/var/lib/mysql-cluster/config.ini
```

S'ha de tenir en compte que `ndb_mgmd` emmagatzema una *cache* de configuracions, així que si no l'arranquem amb el paràmetre `-reload` no s'aplicarà qualsevol nou canvi que haguem fet en el fitxer `config.ini`.

Per a connectar cada node de dades hem d'executar la comanda:

```
sudo ndbd --connect-string="host=master"
```

Transcorreguts pocs minuts el *cluster* estarà disponible i en funcionament. Finalment, només queda connectar el servidor `mysqld` del node SQL, en el nostre cas el propi master.
`mysqld` també es coneix com `MySQL Server` i és l'encarregat de rebre peticions de clients i gestionar l'accés a les bases de dades.
L'engegarem amb la comanda següent:

```
/usr/local/mysql/bin/mysqld --ndbcluster
        --ndb-connectstring="master:1186" --big-tables
```

El paràmetre `-ndbcluster` indica al servidor que actua en mode *cluster*. D'altra banda, `-big-tables` ens permet el funcionament amb *disk data objects*, els quals s'expliquen a la secció 5.5.1.

Un cop arrancat, per a verificar el seu estat des del node d'administració executem `ndb_mgm` i escrivim la comanda `show`. Hauriem d'obtenir un resultat com el següent:

```
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: master:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     3 node(s)
id=2    @192.168.1.48  (mysql-5.6.10 ndb-7.3.1, Nodegroup: 0, Master)
id=3    @192.168.1.47  (mysql-5.6.10 ndb-7.3.1, Nodegroup: 1)
id=4    @192.168.1.46  (mysql-5.6.10 ndb-7.3.1, Nodegroup: 2)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @192.168.1.45  (mysql-5.6.10 ndb-7.3.1)

[mysqld(API)]   5 node(s)
id=5    @192.168.1.45  (mysql-5.6.10 ndb-7.3.1)
```

Figura 4.2: Nodes correctament iniciats.

En aquest punt ja tenim el *cluster* en funcionament i preparat per rebre peticions des de qualsevol client de MySQL.

4.3 HBase

En aquesta secció es descriurà com instal·lar i configurar cada component necessari per a treballar en el projecte.

4.3.1 Instal·lació i configuració

Hadoop

La instal·lació de Hadoop és tan simple com descarregar l'últim paquet estable des del node master. Aquest el situem a una carpeta del sistema i assignem permisos:

```
cd /usr/local
sudo wget http://apache.rediris.es/hadoop/
        common/stable/hadoop-1.0.3.tar.gz
sudo tar xzf hadoop-1.0.3.tar.gz
sudo mv hadoop-1.0.3 hadoop
sudo rm hadoop-1.0.3.tar.gz
sudo chmod 777 hadoop/ -R
```

Un cop instal·lat editem els diferents fitxers de configuració. Modifiquem l'script `/usr/local/hadoop/conf/hadoop-env.sh`, descomentem la variable `JAVA_HOME` i afegim la ruta correcta. En el nostre cas:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
```

A continuació indiquem quins són els nodes del clúster que funcionaran com a nodes de dades o esclaus. Modifiquem el fitxer `/usr/local/hadoop/conf/slaves` i afegim els *hostnames* separats per salts de línia.

Hadoop ens permet indicar la carpeta en la que volem que emmagatzemi les dades, normalment anomenada *datastore*. En el nostre cas, la creem en l'arrel de l'usuari:

```
sudo mkdir /home/serginadalfrancesch/hadoopDataStore
sudo chmod 777 /home/serginadalfrancesch/hadoopDataStore -R
```

Modifiquem els diferents fitxers de configuració obligatoris per al correcte funcionament de Hadoop. Val a dir que aquests sistemes són altament parametritzables i una mala decisió en algun valor pot portar a una baixada significativa del rendiment.
El primer a modificar és `/usr/local/hadoop/conf/core-site.xml`, afegim el següent contingut:

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/serginadalfrancesch/hadoopDataStore
                /hadoop-${user.name}
  </value>
</property>

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

El paràmetre `hadoop.tmp.dir` ens indica la ruta del *datastore* comentat previament, aquest també s'utilitza com a directori temporal.
El segon paràmetre, `fs.default.name` indica el nom i ruta del sistema de fitxers per a aquest cluster. El port 54310 és el que Hadoop utilitza per defecte

i no és necessari modificar-lo.

Modifiquem el fitxer `/usr/local/hadoop/conf/mapred-site.xml`, aquest ens permet modificar les característiques del *job tracker* de MapReduce, l'encarregat de processar les peticions i enviar-les a Hadoop. Afegim el següent contingut:

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

<property>
  <name>mapred.map.tasks</name>
  <value>30</value>
</property>

<property>
  <name>mapred.reduce.tasks</name>
  <value>6</value>
</property>
```

Els paràmetres afegits són els següents:

- `mapred.job.tracker`: indica quin és el *host* i port per al *job tracker*. En cas que sigui local, les tasques MapReduce sempre seran executades en una sola operació de Map i una de Reduce.
- `mapred.map.tasks` i `mapred.reduce.tasks`: limitem el nombre de tasques Map i Reduce concurrents que poden ser executades. Tal com s'indica a la guia de configuració de Hadoop, com a norma general usarem 10 vegades el nombre d'esclaus disponibles i pel segón dos vegades el mateix nombre d'esclaus.

Finalment, modifiquem el fitxer `/usr/local/hadoop/conf/hdfs-site.xml`:

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```

El paràmetre `dfs.replication` ens permet indicar el nombre de rèpliques per defecte de cada fitxer. Aquest nombre també es pot indicar en la creació de qualsevol fitxer a HDFS, en cas de no indicar-se s'utilitzarà aquest valor per defecte.
Degut a que un dels objectius d'aquest projecte és el de fer proves amb el nombre màxim de dades possibles no utilitzarem replicació, és a dir, una sola rèplica per fitxer.

Finalment, un cop configurat correctament Hadoop només resta traspassar tots els fitxers als nodes esclaus. Primer, des de cada node esclau cal crear el directori on residirà Hadoop:

```
sudo mkdir /usr/local/hadoop
sudo chmod 777 /usr/local/hadoop -R
```

A continuació des del node master copiem a cada node esclau el directori Hadoop complet:

```
scp -r /usr/local/hadoop 192.168.1.46:/usr/local/hadoop
```

L'últim pas és formatejar el sistema de fitxers, aquest procés ens crearà el sistema de fitxers distribuït al llarg de tots els nodes esclaus:

```
/usr/local/hadoop/bin/hadoop namenode -format
```

Arrancarem DFS i el *Job Tracker* en aquest ordre des del node master mitjançant:

```
/usr/local/hadoop/bin/start-dfs.sh
/usr/local/hadoop/bin/start-mapred.sh
```

Podem comprovar el correcte funcionament, accedint des del navegador a `http://master:50030` i `http://master:50070`.

HBase

De forma similar que amb Hadoop, des del node master es descarrega l'últim paquet estable i s'assignen els permisos necessaris:

```
cd /usr/local/hbase
```

```
wget http://apache.rediris.es/hbase/stable/hbase-0.94.1.tar.gz
sudo tar xzf hbase-0.94.1.tar.gz
sudo mv hbase-0.94.1 hbase
sudo rm hbase-0.94.1.tar.gz
sudo chmod 777 hbase/ -R
```

Per a la configuració d'HBase serà necessari que Hadoop estigui aturat. El primer pas consisteix en afegir les llibreries d'HBase al *classpath* de Hadoop al fitxer `/usr/local/hadoop/conf/hadoop-env.sh`

```
export HADOOP_CLASSPATH=/usr/local/hbase/hbase-0.94.1.jar:
        /usr/local/hbase/hbase-0.94.1test.jar:
        /usr/local/hbase/conf:
        /usr/local/hbase/lib/zookeeper-3.4.3.jar
```

És necessari traslladar els canvis a tots els nodes del cluster, ho farem mitjançant `scp` per cada node.

```
scp -r /usr/local/hadoop/conf/hadoop-env.sh
        192.168.1.46:/usr/local/hadoop/conf/
```

A continuació, modifiquem el fitxer `/usr/local/hbase/conf/hbase-env.sh`. Aquest *script*, al igual que Hadoop amb `hadoop-env.sh`, s'executa al arrancar HBase. Afegirem al `classpath` les rutes de Java i d'HBase. A més a més, també és necessari descomentar l'última línia del fitxer, la qual conté la propietat `HBASE_MANAGES_ZK`. Aquesta indica que serà HBase qui farà funcionar ZooKeeper, de forma que heredarà la pròpia configuració de Hadoop i HBase. Els canvis al fitxer són els següents:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
export HBASE_CLASSPATH=/usr/local/hadoop/conf

export HBASE_MANAGES_ZK=true
```

El següent pas és modificar el fitxer de configuració d'HBase, aquest és `/usr/local/hbase/conf/hbase-site.xml`. Afegim els següent paràmetres:

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://master:54310/hbase</value>
```

```
</property>

<property>
  <name>hbase.master</name>
  <value>master:60000</value>
</property>

<property>
  <name>hbase.zookeeper.quorum</name>
  <value>master, tossa, empuries, begur</value>
</property>

<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>

<property>
  <name>hbase.zookeeper.property.maxClientCnxns</name>
  <value>300</value>
</property>
```

A continuació es descriuen cadascun dels paràmetres afegits:

- `hbase.rootdir`: directori compartit per els diferents *region servers*. Aquest forma part del sistema de fitxers de Hadoop.
- `hbase.master`: *host* i port sobre el que HBase funciona.
- `hbase.zookeeper.quorum`: llista de nodes que seran gestionats per ZooKeeper. Tots els nodes que es vulgui que disposin d'HBase han d'estar llistats en aquesta propietat en cas que la opció `HBASE_MANAGES_ZK` sigui *true*.
- `hbase.cluster.distributed`: mode de funcionament del cluster
- `hbase.zookeeper.property.maxClientCnxns`: nombre màxim de connexions de clients que acceptarà ZooKeeper.

Finalment, com a últim fitxer de configuració, modificarem `/usr/local/hbase/conf/regionservers` on llistarem els nodes esclaus d'HBase. A continuació copiarem mitjançant `scp` el directori `/usr/local/hbase`

a cada node esclau.

L'arrancada d'HBase la farem posterior a la del servei DFS i MapReduce, des del node master executarem la següent comanda:

```
/usr/local/hbase/bin/start-hbase.sh
```

Podem comprovar el correcte funcionament, accedint des del navegador a `http://master:60010`.

Hive

La instal·lació de Hive és igual que els altres dos components:

```
cd /usr/local/hive
wget http://apache.rediris.es/hive/hive-0.9.0/hive-0.9.0.tar.gz
sudo tar xzf hive-0.9.0.tar.gz
sudo mv hive-0.9.0 hive
sudo rm hive-0.9.0.tar.gz
sudo chmod 777 hive/ -R
```

És necessari crear i modificar les següents variables d'entorn, això ho farem modificant el fitxer `/etc/bash.bashrc` i afegint el següent contingut al final:

```
export HADOOP_HOME=/usr/local/hadoop
export HIVE_HOME=/usr/local/hive
export PATH=${PATH}:${HIVE_HOME}/bin:${HIVE_HOME}
export HADOOP_CLASSPATH=/usr/local/hbase/hbase-0.94.1.jar:
        /usr/local/hbase/hbase-0.94.1-test.jar:
        /usr/local/hbase/conf:
        /usr/local/hbase/lib/zookeeper-3.4.3.jar:
        /usr/local/hive/lib/*.jar:
        /usr/local/hbase
```

Gràcies a aquestes variables, Hive usarà la configuració d'HBase.
A continuació creem les carpetes temporals de Hive en el sistema de fitxers de Hadoop:

```
$HADOOP_HOME/bin/hadoop fs -mkdir       /tmp
```

```
$HADOOP_HOME/bin/hadoop fs -mkdir       /user/hive/warehouse
$HADOOP_HOME/bin/hadoop fs -chmod g+w   /tmp
$HADOOP_HOME/bin/hadoop fs -chmod g+w   /user/hive/warehouse
```

Per tal d'arrancar Hive amb el connector d'HBase hem d'executar la següent comanda:

```
$HIVE_HOME/bin/hive
  --auxpath /usr/local/hive/lib/hive-hbase handler-0.10.0.jar,
    /usr/local/hive/lib/hbase-0.92.0.jar,
    /usr/local/hive/lib/zookeeper-3.4.3.jar,
    /usr/local/hive/lib/guava-r09.jar,
    /usr/local/hbase/lib/protobuf-java-2.4.0a.jar
  -hiveconf hbase.master=master:60000
```

Capítol 5

Proves

En aquest capítol es descriu el model sobre el que es faran les proves, des del model conceptual fins a la seva generació d'informació. També es detallen les diferents proves que es duran a terme en ambdòs sistemes.

5.1 Model conceptual

Per tal de fer les proves en un entorn el més realista possible treballarem sobre un conjunt d'entitats i fets els quals podriem trobar en qualsevol petita organització. L'objectiu no és donar una complexitat alta a les proves afegint gran quantitat d'entitats i relacions, és preferible partir d'un model bàsic i fixar unes bases per tal de poder treure'n conclusions extrapolables a models més complexos.
Concretament el model estarà basat en un datamart de vendes, suposarem que les dades sobre les que treballarem són extretes d'un sistema OLTP. El model conceptual sobre el que es faran les proves, descrit a la figura 5.1.

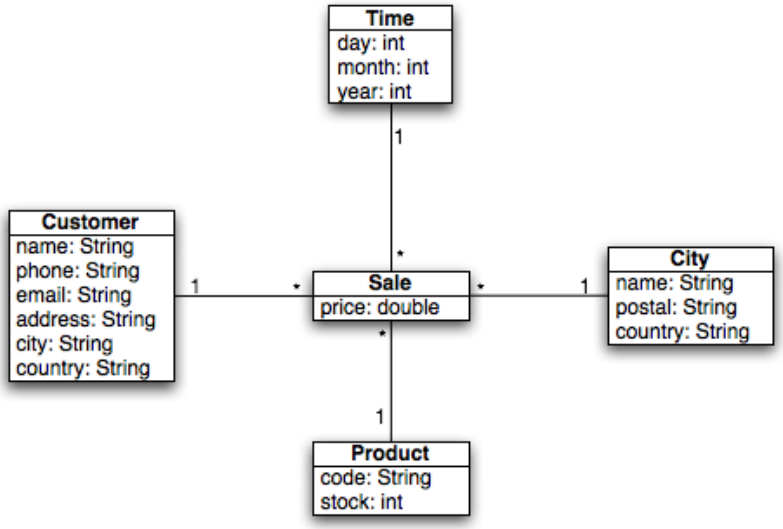


Figura 5.1: Model conceptual amb quatre dimensions i un fet.

Tota venda té assignades una combinació de client, ciutat, producte i data. L'atribut *city* de client correspon a la ciutat del client, mentre que l'entitat ciutat correspon a la ciutat on s'ha efectuat la venda.

Després d'executar un primer conjunt de proves, s'ha eliminat una dimensió del model ja que els temps d'execució i el número de proves diferents eren massa grans.
De totes formes, esperem que amb les dades recollides per a tres dimensions ens permetin extrapolar i fer estimacions de temps per a més dimensions. El model conceptual resultant i sobre el que s'executaran el conjunt de proves serà el descrit a la figura 5.2.

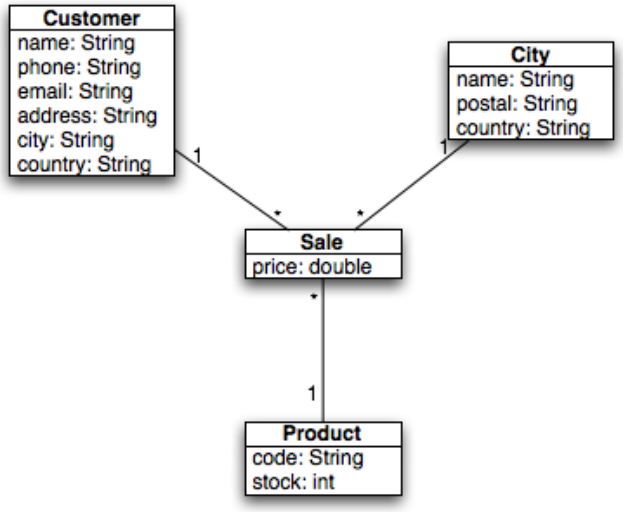


Figura 5.2: Model conceptual final, tres dimensions i un fet.

5.2 Esquema físic

Tant per totes les proves en el sistema relacional com per el primer conjunt del sistema no relacional, la informació s'emmagatzemarà a la base de dades en un esquema en estrella. A continuació detallarem les propietats d'aquest esquema i els seus pros i contres respecte d'altres configuracions d'esquema.

5.2.1 Esquema en estrella

És el més aplicat en bases de dades utilitzades com a *data warehouse*, aquest es basa en la desnormalització de les taules de dimensió.
Alguns dels beneficis que aquests aporta són:

- **Optimització de consultes**: es poden crear índexs sobre les taules de dimensions que normalment són molt més petites que les de fets. A l'hora de fer una consulta, el filtrat es pot fer a les taules de dimensions a partir d'aquests índexs.
- **Particionament**: únicament s'ha de particionar la taula de fets que per norma serà la més gran i problemàtica.
- **Fàcil comprensió de l'esquema**: és senzill veure totes les relacions directes que té un fet.

- **Reducció del nombre de joins**: s'eviten join innecessàries, ja que només s'utilitzen aquelles dimensions que necessita la consulta.

5.2.2 Altres esquemes

Tot i que l'esquema en estrella és el més utilitzat, no és l'únic emprat en *data warehousing*. L'ús d'un esquema o un altre variarà segons les dades de les que disposen, ja que l'esquema en estrella no serveix en tots els casos. Alguns altres esquemes utilitzats són:

- **Esquema en floc de neu o *snowflake***: aquest és una variant més complexa de l'esquema en estrella. Algunes parts de les dimensions es poden tornar a normalitzar per tal d'estalviar espai.
  Una aplicació comuna d'aquest és quan rarament es consulta un conjunt d'atributs de la dimensió, en aquest cas potser surt més a compte penalitzar aquestes consultes poc freqüents.

  Un exemple de normalització amb la taula de clients utilitzada pot ser el següent:

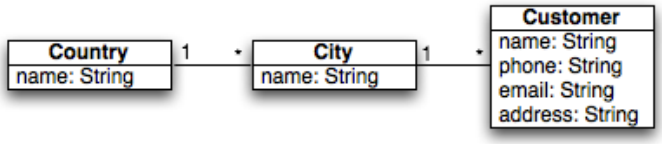


Figura 5.3: Dimensió de client normalitzada.

- **Esquema en galàxia**: utilitzarem aquest esquema quan tenim més d'una taula de fets i dimensions compartides entre aquestes.

5.3 Generació de dades

Abans d'executar tots els conjunts de proves hem de carregar informació a les taules de dimensions. Aquesta ha d'ésser la mateixa per a totes les proves per tal de mantenir la coherència de les mateixes.

Pel que fa a la càrrega d'informació a la taula de fets, s'usarà un mètode basat en probabilitats per evitar que totes les tuples de les diferents dimensions tinguin una aparició equiprobable a la taula de fets, aquest s'explicarà amb més detall a la secció 5.3.2.

5.3.1 Dimensions

Per a la càrrega de dades mestres s'usarà una aplicació *opensource*, Data Generator. Aquesta és disponible a `http://www.generatedata.com/`. Els motius de l'elecció es podrien resumir en els següents punts:

- Aplicació molt senzilla però que compleix amb els nostres requisits.
- Disposa d'una base de dades completa amb informació real com: noms, ciutats, adreces o emails.
- Desenvolupada en PHP, fàcil aplicació d'alguns canvis com ara el límit de files generades.

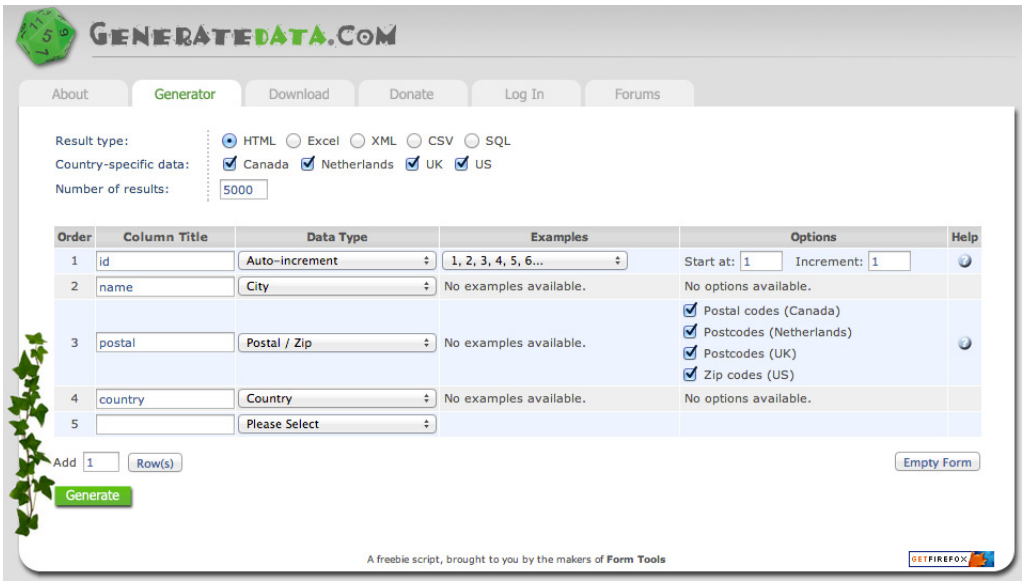


Figura 5.4: Exemple de generació de dades per ciutats.

Aquestes taules es carregaran a una base de dades MySQL resident al node *master*. Per fer les proves sobre MySQL Cluster només caldrà fer una còpia d'aquestes taules, però per a HBase caldrà un processament previ a la

càrrega.

Les sentències de creació de les taules, amb els tipus concrets que s'usaran, són les següents:

```
CREATE TABLE `CITIES` (
  `id` mediumint PRIMARY KEY,
  `name` varchar(100),
  `postal` varchar(10),
  `country` varchar(100),
  `key` mediumint
);

CREATE TABLE `CUSTOMERS` (
  `id` mediumint PRIMARY KEY,
  `name` varchar(255),
  `phone` varchar(100),
  `email` varchar(255),
  `address` varchar(255),
  `country` varchar(100),
  `city` varchar(100),
  `key` mediumint
);

CREATE TABLE `PRODUCTS` (
  `id` mediumint PRIMARY KEY,
  `code` varchar(255),
  `stock` mediumint,
  `key` mediumint
);
```

En el codi anterior es veu la columna *key* la qual no apareix al model conceptual ni se n'havia parlat fins ara. Aquesta és una còpia de la columna *id* però no està indexada, d'aquesta forma es podran executar les consultes amb els factors de selecció desitjats sense utilitzar índexs.

```
mysql> select * from CITIES limit 10;
+----+-------------------------+----------+---------------+------+
| id | name                    | postal   | country       | key  |
+----+-------------------------+----------+---------------+------+
|  2 | South Africa            | 6357OA   | Prince Albert |    2 |
|  3 | Macedonia               | 53718    | Southaven     |    3 |
|  4 | Liechtenstein           | 4165     | Gjoa Haven    |    4 |
|  5 | Liberia                 | E9A 4D6  | Milnathort    |    5 |
|  6 | Trinidad and Tobago     | GZ1G 5ZP | Houston       |    6 |
|  7 | Bosnia and Herzegovina  | 6806     | Omaha         |    7 |
|  8 | Algeria                 | 24348    | Newport       |    8 |
|  9 | Afghanistan             | Z7E 8P4  | Anderlecht    |    9 |
| 10 | Armenia                 | 1102CF   | Falmouth      |   10 |
| 11 | Congo                   | H9W 4MN  | Owensboro     |   11 |
+----+-------------------------+----------+---------------+------+
10 rows in set (0.13 sec)
```

Figura 5.5: Mostra de dades carregades a la taula *cities*.

Pel que fa a l'espai ocupat, d'acord amb la documentació de MySQL cada columna de tipus *mediumint* ocupa 3 *bytes*. Pel que fa a les columnes *varchar*, s'utilitza un *byte* més el tamany de la cadena si aquesta ocupa menys de 255 *bytes*, altrament dos *bytes* més el tamany de la cadena. Aquest fet ha estat contrastat en una base de dades de prova i els espais s'han correspós amb la documentació.
Consultant la mitjana d'espai per cada columna per les dades generades obtenim el següent (arrodonint els *bytes* a la unitat superior):

- ***CITIES***:
  - *id*: 4 *bytes*
  - *name*: 11 *bytes*
  - *postal*: 6 *bytes*
  - *country*: 9 *bytes*
  - *key*: 4 *bytes*
  - **Total**: 34 *bytes*
- ***CUSTOMERS***:
  - *id*: 4 *bytes*
  - *name*: 17 *bytes*

  - *phone*: 11 *bytes*
  - *email*: 29 *bytes*
  - *address*: 23 *bytes*
  - *country*: 10 *bytes*
  - *city*: 9 *bytes*
  - *key*: 4 *bytes*
  - **Total**: 107 *bytes*

- ***PRODUCTS***:
  - *id*: 4 *bytes*
  - *code*: 11 *bytes*
  - *stock*: 4 *bytes*
  - *key*: 4 *bytes*
  - **Total**: 23 *bytes*

Pel que fa al nombre de tuples, en primera instància s'havia plantejat treballar amb 10000, 5000 i 2500 tuples per a les taules de clients, ciutats i productes respectivament. Després d'haver fet proves els temps eren prohibitius per al hardware disponible, s'ha reduït el nombre de tuples mantenint la proporcionalitat, concretament 1000, 500 i 250.

5.3.2 Fets

La càrrega de les vendes variarà molt segons el sistema i el model aplicat, tot i així, per tal d'evitar una distribució equiprobable de les dades tota càrrega passarà per una fase comuna.
Serà necessari crear les taules presentades a la següent imatge dins MySQL.

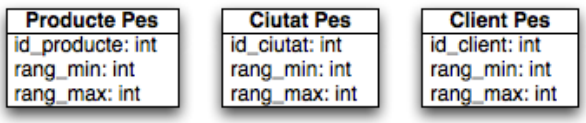


Figura 5.6: Noves taules afegides per a l'ús de rangs.

Per cada element de les taules de dimensions, existirà un equivalent a la taula de rangs. Aleatòriament es generarà un rang mínim i un rang màxim,

sempre evitant col·lisions amb els altres, de forma que la grandària d'aquest rang serà el pes que tindrà cada identificador per aparéixer a la taula de fets. Un exemple gràfic senzill seria el següent:

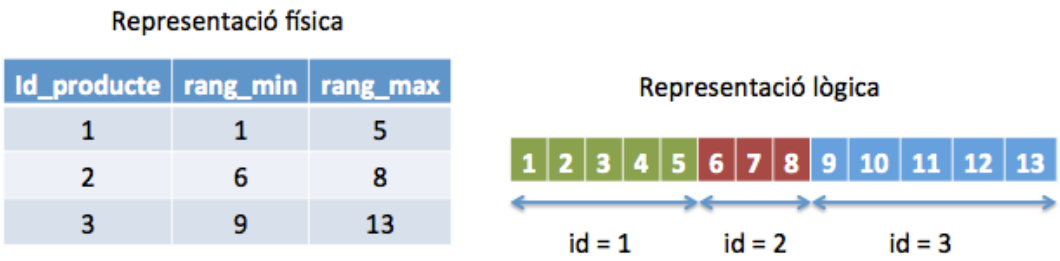


Figura 5.7: Representació física i lògica dels rangs.

A l'hora de generar els fets per a cada model, el carregador haurà de generar un número aleatori i buscar dins quin rang cau, aleshores aquest li donarà l'id de la dimensió associada.

Script de generació de rangs

La generació de rangs aleatoris per a cada taula es farà mitjançant un *script* SQL. Es mostra el codi per a rangs de productes, per la resta de dimensions és igual canviant les taules destí.

```
1  CREATE PROCEDURE ‘generar_rangs_productes‘()
2  BEGIN
3  DECLARE l_last_row_fetched int;
4
5  DECLARE num_aleatori_1 INT(50);
6  DECLARE num_aleatori_2 INT(50);
7  DECLARE producte mediumint;
8
9  DECLARE c1 CURSOR FOR SELECT id FROM PRODUCTS;
10 DECLARE CONTINUE HANDLER FOR NOT FOUND SET l_last_row_fetched=1;
11
12 SET l_last_row_fetched=0;
13 SET num_aleatori_1=0;
14
15 OPEN c1;
16 cursor_loop:LOOP
17 FETCH c1 INTO producte;
```

```
18        IF l_last_row_fetched=1 THEN LEAVE cursor_loop;
19        END IF;
20
21        SELECT FLOOR((RAND() * (100+1))) INTO num_aleatori_2;
22        SET num_aleatori_2=num_aleatori_1+num_aleatori_2;
23        INSERT INTO producte_pes (id_producte,rang_min,rang_max)
24        VALUES (producte,num_aleatori_1+1,num_aleatori_2);
25
26        SET num_aleatori_1=num_aleatori_2;
27
28  END LOOP cursor_loop;
29  END
```

Finalment, la càrrega de la informació de vendes dependrà principalment del sistema de base de dades utilitzat. Ambdues es faran des de Java, a continuació es diferencien i mostren les parts més rellevants d'ambdós casos.

Càrrega a MySQL

Degut a la gran quantitat de tuples que s'han de carregar, és inviable utilitzar interfícies tipus JDBC per a càrregues massives.
En aquest cas, la sol·lució més adient és exportar un número concret de tuples a un fitxer de text perquè a continuació el client de MySQL faci una càrrega massiva.

L'exportació de tuples es fa mitjançant la següent funció:

```
1  private static void exportRandomData(int N_ROWS) throws Exception {
2        FileWriter fw = new FileWriter(filePath);
3        for (int i = 0; i < N_ROWS; ++i) {
4              fw.append(getRandomCustomer()+","+getRandomProduct()+","+
5              getRandomCity()+","+getRandomPrice()+"\n");
6        }
7        fw.close();
8  }
```

Un cop exportades les dades es carreguen a la taula final mitjançant:

```
1  private static void insertData() throws Exception {
2        Statement st = conn.createStatement();
3        String query = "LOAD DATA INFILE '"+filePath+"' into table
4              SALES FIELDS TERMINATED BY ',' LINES TERMINATED BY '\\n';";
5        try {
6              st.execute(query);
7        } catch (SQLException exc) {
```

```
8               exc.printStackTrace();
9       }
10  }
```

Càrrega a HBase

Per a carregar les dades massivament a HBase, es farà mitjançant *threads* concurrents, d'aquesta forma evitarem les insercions seqüencials i les dades quedaran desordenades. El control del flux de *threads* és una tasca complexa i nosaltres només necessitem una cua per limitar-ne el número actiu per no desbordar el sistema. Per això, hem implementat la nostra pròpia cua, el codi de la classe és el següent:

```
1   public class Buffer {
2       private int i = 0;
3
4       public synchronized void addThread() {
5               if (getSize() >= 2500) {
6                       wait();
7               }
8               i++;
9       }
10
11      public synchronized void removeThread() {
12              i--;
13              notify();
14      }
15
16      public int getSize() {
17              return i;
18      }
19  }
```

En aquest cas usem un límit de 2500 *threads* concurrents, ja que al augmentar aquest límit baixava el rendiment o la màquina virtual de Java s'aturava.

Cada *thread* cridarà al mètode `run()`, el qual farà la inserció d'una tupla aleatòria. El següent codi insereix una tupla per al model 1.

```
1   @Override
2   public void run() {
3       table = tablePool.getTable(table_name);
4       tab = (HTable) table;
```

```
5       tab.setAutoFlush(false);
6       int city = getRandomCity();
7       int product = getRandomProduct();
8       int customer = getRandomCustomer();
9       String price = getRandomPrice();
10      Put put = new Put(Bytes.toBytes(contador));
11      put.setWriteToWAL(false);
12      put.add(Bytes.toBytes("colfamREFERENCES"),
13            Bytes.toBytes("city_id"), Bytes.toBytes(""+city));
14      put.add(Bytes.toBytes("colfamREFERENCES"),
15            Bytes.toBytes("customer_id"), Bytes.toBytes(""+customer));
16      put.add(Bytes.toBytes("colfamREFERENCES"),
17            Bytes.toBytes("product_id"), Bytes.toBytes(""+product));
18      put.add(Bytes.toBytes("colfamREFERENCES"),
19            Bytes.toBytes("price"), Bytes.toBytes(price));
20      try {
21            table.put(put);
22            tab.flushCommits();
23            tab.close();
24      } catch (Exception exc) {
25            exc.printStackTrace();
26      }
27      b.removeThread();
28  }
```

5.4 Definició de les proves

Per a provar el rendiment en ambdós sistemes i les diferents configuracions de model es faran proves mitjançant consultes SQL.
Les consultes es centren en les diferents combinacions de join entre taules i els factors de selecció aplicats.

Pel que fa a combinació de joins, sorgeixen les següents:

- 1 dimensió
  1. *CUSTOMERS*
  2. *PRODUCTS*
  3. *CITIES*
- 2 dimensions

4. *CUSTOMERS* ⋈ *PRODUCTS*
5. *CUSTOMERS* ⋈ *CITIES*
6. *CITIES* ⋈ *PRODUCTS*

- 3 dimensions

7. *CUSTOMERS* ⋈ *PRODUCTS* ⋈ *CITIES*

El control de factors de selecció es farà a partir de la columna *key* definida a cada taula de dimensions, i s'aplicaran mitjançant l'operació mòdul. Concretament el nombre de tuples resultants per a cada factor de selecció i la operació mòdul que obté les tuples són:

Factor de selecció	Dimensió	Tuples seleccionades	Filtre SQL
10^{-3}	*Customers*	1	CUSTOMERS.key%1000=1
	Cities	1	CITIES.key%500=1
	Products	1	PRODUCTS.key%250=1
10^{-2}	*Customers*	10	CUSTOMERS.key%100=1
	Cities	5	CITIES.key%100=1
	Products	3	PRODUCTS.key%150=1
10^{-1}	*Customers*	100	CUSTOMERS.key%10=1
	Cities	50	CITIES.key%10=1
	Products	25	PRODUCTS.key%10=1
10^{0}	*Customers*	1000	CUSTOMERS.key%1=0
	Cities	500	CITIES.key%1=0
	Products	250	PRODUCTS.key%1=0

Finalment, combinant els dos factors previs veiem el resultat final en format SQL per a *MySQL*, concretament una dimensió amb factor de selecció 10^{-3}:

```
1 SELECT PRODUCTS.code, AVG(SALES.price) FROM PRODUCTS JOIN SALES ON SALES.
    id_city = PRODUCTS.id WHERE PRODUCTS.key%250=1;
```

Per a obtenir dades fiables es repetirà cada prova tres vegades.

5.5 Descripció dels models

En aquesta secció es veuran les diferents configuracions de models tan per *MySQL Cluster* com per *HBase*. Es descriurà el model, les motivacions d'ús i les consultes a aplicar.

5.5.1 Model relacional: *MySQL Cluster*

Per a avaluar el temps de consulta a *MySQL* s'aprofitaran les mateixes taules de dimensions descrites en seccions anteriors.
Donat que *MySQL Cluster* emmagatzema per defecte els registres a memòria, és necessari indicar-li explícitament per a quines columnes de cada taula no ho ha de fer. Aconseguirem aquest comportament mitjançant la opció `STORAGE DISK` el qual emmagatzemarà les dades mitjançant *Disk Data Objects*.

Disk Data Objects

A part dels registres, un cop s'emmagatzemen dades mitjançant `STORAGE DISK` cal definir altres objectes. Aquests són els següents:

- ***Tablespaces***: actua com a contenidor a disc per tots els objectes de tipus *Disk Data*.
- ***Undo log***: conté la llista de transaccions executades en el sistema per tal de fer *rollback* en cas que sigui necessari recuperar un estat passat.
- ***Log file group***: pot contenir múltiples fitxers de log, posteriorment s'assigna a un *tablespace*.
- ***Data files***: emmagatzemen les tuples de la taula.

Per a les proves es crearan aquests elements, per començar es crea el *log file group* i se li afegeix un log. És important tenir en compte que, tot i que a la documentació oficial s'indica en MB, tots els valors referents a espais de disc s'han d'introduir en bytes.

```
1 CREATE LOGFILE GROUP lg ADD UNDOFILE 'undo.log'
2       INITIAL_SIZE 67108864 UNDO_BUFFER_SIZE 8388608 ENGINE NDBCLUSTER;
```

El paràmetre `INITIAL_SIZE` indica el tamany inicial del log, per defecte 128MB. Després de diferents proves, per el nostre cas 64MB (67108864 bytes) donen el mateix rendiment.
Pel que fa el paràmetre `UNDO_BUFFER_SIZE`, indica el tamany de l'*undo log* per al grup. S'usa el valor per defecte de 8MB (8388608 bytes).

Un cop creat el grup de logs, s'ha de crear el *tablespace* i assignar el log a aquest.

```
1 CREATE TABLESPACE ts ADD DATAFILE 'data.dat' USE
2       LOGFILE GROUP lg INITIAL_SIZE 134217728 ENGINE NDBCLUSTER;
```

El paràmetre `INITIAL_SIZE` indica el tamany inicial donat. En aquest cas s'utilitzarà el valor per defecte de 128 MB, ja que la majoria de tuples s'emmagatzemaran a disc.

Finalment, un cop creades totes les estructures necessàries per a mantenir *disk data objects*, ja és possible crear taules d'aquest tipus. Per al cas de la nostra taula *SALES*, la sentència SQL de creació seria:

```
CREATE TABLE ‘SALES‘ (
  ‘id_customer‘ mediumint STORAGE DISK NOT NULL,
  ‘id_product‘ mediumint STORAGE DISK NOT NULL,
  ‘id_city‘ mediumint STORAGE DISK NOT NULL,
  ‘price‘ DECIMAL(10,2) STORAGE DISK NOT NULL
) TABLESPACE ts STORAGE DISK ENGINE=NDBCLUSTER;
```

Pel que fa a les consultes, s'utilitzarà la paraula clau `STRAIGHT_JOIN` com a pista o *hint* per a l'optimitzador de consultes. Tot i que l'optimitzador acostuma a usar el pla d'accés més òptim, és possible que consulti les taules en un ordre molt perjudicial. Donat que la configuració de taules usada resulta en una taula molt gran i la resta petites, té lògica consultar primer les petites per tal de reduir el nombre de tuples seleccionades per tal de minimitzar el cost de la operació join amb la taula *SALES*.
Aquest comportament descrit és exactament el que provoca l'ús de la paraula clau `STRAIGHT_JOIN`, consulta les taules de dreta a esquerra en el mateix ordre que s'ha escrit a la sentència SQL.

La sintaxi usada en la sentència SQL variarà segons el nombre de taules implicades en la join, pel cas que només es consulta una dimensió s'usarà el mateix mètode que l'operació `JOIN` de *MySQL*. Per exemple, la següent sentència per una dimensió i factor de selecció 10^{-3}:

```
SELECT CUSTOMERS.name, AVG(SALES.price) FROM CUSTOMERS STRAIGHT_JOIN
    SALES ON SALES.id_customer = CUSTOMERS.id WHERE CUSTOMERS.key%1000=1
    GROUP BY CUSTOMERS.name;
```

D'altra banda, quan el nombre de joins és major que 1 si utilitzéssim el mateix mètode d'abans la taula *SALES* seria la segona en ser consultada i només es podria aplicar la selecció per una de les dimensions. És per això que la join es farà a partir de les parts `FROM` i `WHERE` de la consulta, per exemple per a dues dimensions i factor de selecció 10^{-3}:

```
SELECT STRAIGHT_JOIN CITIES.name, PRODUCTS.code, AVG(SALES.price) FROM
    CITIES,PRODUCTS,SALES WHERE SALES.id_city = CITIES.id AND SALES.
    id_product = PRODUCTS.id AND PRODUCTS.key%250=1 AND CITIES.key%500=1
    GROUP BY CITIES.name, PRODUCTS.code;
```

Finalment, tot i donar pistes a l'optimitzador de consultes, per tal de millorar el rendiment de les consultes es modificaran dos paràmetres globals del sistema, aquests són:

- `ndb_join_pushdown`: permet la paral·lelització de les consultes en tots els nodes de dades, evitant que tota es dugui a terme on resideix el servidor *mysqld*. Per defecte a MySQL Cluster està desactivada i s'activarà.
- `ndb_use_transactions`: MySQL permet decidir si s'usen transaccions en qualsevol moment, per el procés de proves al estar només consultant dades podem prescindir d'aquesta gestió. Per defecte està activada i es desactivarà.

5.5.2 Model 1

El primer model és una rèplica del relacional, consisteix en tres taules diferents per a les dimensions i una per als fets. Cada taula té una sola *column family*, per a les dimensions `colfamINFO` i per a les relacions de la taula de fets `colfamREFERENCES`.

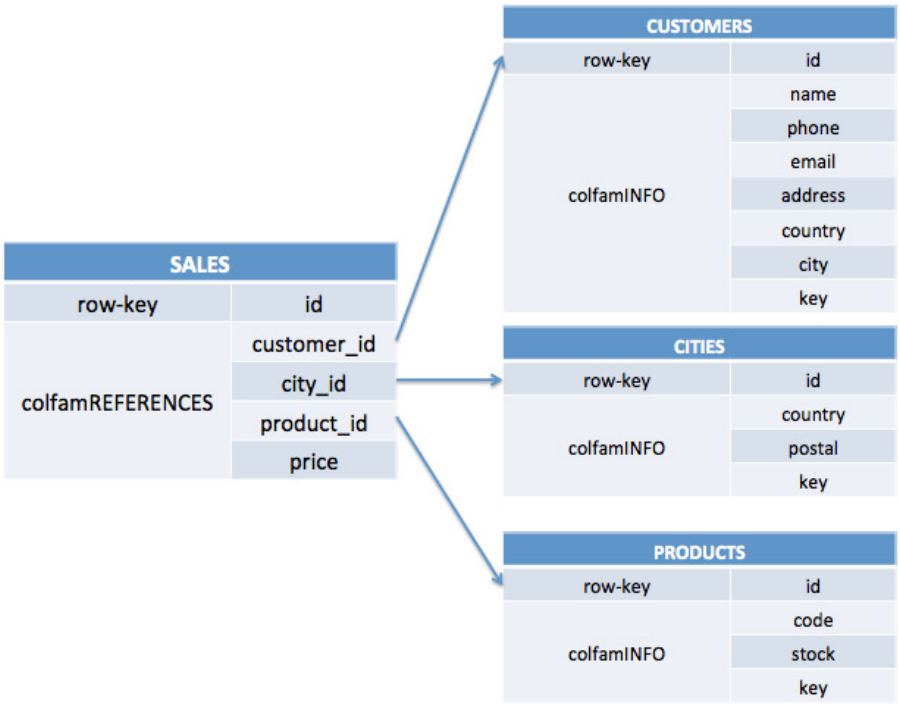


Figura 5.8: Representació gràfica del model 1.

Per aquest model com per la resta, s'usaran com a *row-key* números naturals consecutius. Aquests naturals s'emmagatzemaran amb el tipus String per tal de simular un entorn on la taula de fets pot ser molt gran i superar el màxim nombre que és capaç d'emmagatzemar un enter.

La conversió aplicada des del model relacional és la més senzilla i directa. S'espera un rendiment baix degut a les operacions de join tot i no fer un ús intensiu d'espai.

Per a crear les taules a *HBase* s'executaran les següents comandes:

```
create 'CUSTOMERS','colfamINFO'
create 'CITIES','colfamINFO'
create 'PRODUCTS','colfamINFO'
create 'SALES','colfamREFERENCES'
```

A continuació s'han de crear les taules a Hive, aquest cop indicant els camps concrets:

```
CREATE EXTERNAL TABLE CUSTOMERS(id int, name string, phone string, email
    string, address string, country string, city string, key int) STORED
    BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
    SERDEPROPERTIES ("hbase.columns.mapping" = ":key,colfamINFO:name,
    colfamINFO:phone,colfamINFO:email,colfamINFO:address,colfamINFO:
    country,colfamINFO:city,colfamINFO:key") TBLPROPERTIES("hbase.table.
    name" = "CUSTOMERS");

CREATE EXTERNAL TABLE PRODUCTS(id int, code string, stock int, key int)
    STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
    SERDEPROPERTIES ("hbase.columns.mapping" = ":key,colfamINFO:code,
    colfamINFO:stock,colfamINFO:key") TBLPROPERTIES("hbase.table.name" = "
    PRODUCTS");

CREATE EXTERNAL TABLE CITIES(id int, country string, postal string, name
    string, key int) STORED BY 'org.apache.hadoop.hive.hbase.
    HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping" = "
    :key,colfamINFO:country,colfamINFO:postal,colfamINFO:name,colfamINFO:
    key") TBLPROPERTIES("hbase.table.name" = "CITIES");

CREATE EXTERNAL TABLE SALES(id string, city_id int, customer_id int,
    product_id int, price double) STORED BY 'org.apache.hadoop.hive.hbase.
    HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping" = "
    :key,colfamREFERENCES:city_id,colfamREFERENCES:customer_id,
    colfamREFERENCES:product_id,colfamREFERENCES:price") TBLPROPERTIES("
    hbase.table.name" = "SALES");
```

A HBase el tipus int ocupa 4 *bytes*, i suposarem que els String també, ja que els valors utilitzats no seran tant grans com `MAX_INT`. D'altra banda, el tipus double ocupa 8 *bytes*.
En base a això, cada tupla de la taula de fets del model 1 ocupa 24 *bytes*.

Hive no utilitza llenguatge SQL estàndar, de forma que les mateixes consultes utilitzades a MySQL no serveixen. Es disposen de diferents variants de l'operació join, concretament:

- **Join**: es converteix cada join en una tasca MapReduce, en cas que s'utilitzi el mateix camp per a cada taula la consulta es pot resoldre en una sola tasca MapReduce.
- **Map Side Join**: si totes les taules que s'uneixen excepte una són petites , la join es pot resoldre en una sola tasca Map. No és necessari aplicar la funció Reduce.
  Hi ha una restricció per a aquest cas, no és possible fer Map Side Joins per FULL OUTER JOIN o RIGHT OUTER JOIN.
  S'indicarà mitjançant /*+ MAPJOIN(a) */, on *a* és la taula gran.
- **Bucketed Map Join**: un *bucket* és una porció de taula d'una partició de la taula. En cas que el nombre de *buckets* sigui múltiple entre les diferents taules, aquestes es poden resoldre aplicant una sola operació Map.
  Es pot aplicar aquest comportament mitjançant el paràmetre `hive.optimize.bucketmapjoin`.

En el nostre cas s'usarà l'operació join simple i s'activarà el paràmetre `hive.auto.convert.join`, aquest aplicarà la variant més convenient segons l'optimitzador de consultes.

Un exemple de consulta resultant per al model 1 és la següent:

```
1 SELECT customers.name, products.code, cities.name, AVG(SALES.price) FROM
    customers JOIN sales ON (customers.id = sales.customer_id) JOIN
    products ON (products.id = sales.product_id) JOIN cities ON (cities.id
     = sales.city_id) WHERE customers.key%1000=1 AND products.key%250=1
    AND cities.key%500=1 GROUP BY customers.name, products.code, cities.
    name;
```

5.5.3 Model 2

El segon model pretén aprofitar les avantatges de gran emmatzemmament de dades que ofereix HBase desnormalitzant les dades. Passem de tres taules de dimensions i una de fets, a una sola taula de fets que englobarà tota la informació.
Hi haurà una elevada repetició d'informació, tot i així esperem que al evitar l'ús d'operacions join es millori el temps de consulta.

Pel que fa a les *column family* de la taula s'utilitzarà una per a cada dimensió i una per al fet, és a dir quatre. Les recomanacions dels desenvolupadors d'HBase són de no superar les tres famílies de columnes, volem veure realment com afecta aquest fet.

SALES	
row-key	id
colfamPRODUCT	id
	code
	stock
	key
colfamCITY	id
	country
	postal
	key
colfamCUSTOMER	id
	name
	phone
	email
	address
	country
	city
	key
colfamSALE	price

Figura 5.9: Representació gràfica del model 2.

Crearem la taula a HBase mitjançant:

```
1 create 'SALES','colfamPRODUCT','colfamCITY','colfamCUSTOMER','colfamSALE'
```

Al tractar les dimensions com a famílies de columnes diferents, podem tractar aquestes de forma unitaria com si es tractés d'una estructura tipus taula hash. La sentència de creació a Hive és la següent:

```
1 CREATE EXTERNAL TABLE SALES(id string, product map<string,string>, city
    map<string,string>, customer map<string,string>, price double) STORED
    BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
    SERDEPROPERTIES ("hbase.columns.mapping" = ":key,colfamPRODUCT:,
    colfamCITY:,colfamCUSTOMER:,colfamSALE:price") TBLPROPERTIES("hbase.
    table.name" = "SALES");
```

Pel que fa a l'espai de cada tupla, s'utilitzarà el mateix espai descrit a la secció 5.3.1 per les dimensions. Així doncs, cada tupla ocuparà 176 *bytes*, concretament el detall és:

- ***CITIES***: 34 *bytes*
- ***CUSTOMERS***: 107 *bytes*
- ***PRODUCTS***: 23 *bytes*
- ***id***: 4 *bytes*
- ***price***: 8 *bytes*

Finalment, una possible consulta per a aquest model és la següent:

```
1 SELECT sales.customer["name"], sales.product["code"], sales.city["name"],
     AVG(SALES.price) FROM sales WHERE sales.customer["key"]%1000=1 AND
    sales.product["key"]%250=1 AND sales.city["key"]%500=1 GROUP BY sales.
    customer["name"], sales.product["code"], sales.city["name"];
```

5.5.4 Model 3

Després d'haver provat el funcionament de les famílies de columnes en el model 2, aquest model vol comprovar el funcionament quan només hi ha una família de columnes i tota la informació volcada a dins. També es vol comprovar com afecta això a la distribució de dades.

SALES	
row-key	id
colfamSALE	prodId
	prodCode
	prodStock
	prodKey
	cityId
	cityCountry
	cityPostal
	cityKey
	custId
	custName
	custPhone
	custEmail
	custAddress
	custCountry
	custCity
	custKey
	price

Figura 5.10: Representació gràfica del model 3.

Per a aquest cas la creació de taula a HBase és molt simple:

```
1 create 'SALES','colfamSALE'
```

I la creació a Hive és similar al model 1:

```
1 CREATE EXTERNAL TABLE SALES(id string, custId int, custName string,
    custPhone string, custEmail string, custAddress string, custCountry
    string, custCity string, custKey int, prodId int, prodCode string,
    prodStock int, prodKey int, cityId int, cityCountry string, cityPostal
     string, cityName string, cityKey int, price double) STORED BY 'org.
    apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("
    hbase.columns.mapping" = ":key,colfamSALE:custId,colfamSALE:custName,
    colfamSALE:custPhone,colfamSALE:custEmail,colfamSALE:custAddress,
    colfamSALE:custCountry,colfamSALE:custCity,colfamSALE:custKey,
    colfamSALE:prodId,colfamSALE:prodCode,colfamSALE:prodStock,colfamSALE:
    prodKey,colfamSALE:cityId,colfamSALE:cityCountry,colfamSALE:cityPostal
    ,colfamSALE:cityName,colfamSALE:cityKey,colfamSALE:price")
    TBLPROPERTIES("hbase.table.name" = "SALES");
```

L'espai ocupat per cada tupla d'aquest model és el mateix cas que el model 2, 176 *bytes*.

Finalment, un exemple de consulta seria el següent:

```
SELECT custName, prodCode, cityName, AVG(price) FROM sales WHERE custKey
    %1000=1 AND prodKey%250=1 AND cityKey%500=1 GROUP BY custName,
    prodCode, cityName;
```

5.5.5 Model 4

Aquest i els dos següents models són iguals que els tres anteriors però aplicant-hi compressió. Amb això, es vol veure com afecta el volum de dades i el procés de descompressió a les consultes.
HBase disposa de tres algorismes de compressió a nivell de família de columnes, a continuació els llistem i detallem les característiques:

- **Snappy**: optimitzat per a proporcionar altes velocitats de compressió i decompressió i un factor de compressió raonable. No és compatible amb altres llibreries de compressió i s'ha d'instal·lar manualment a cada *region server*.
- **LZO**: optimitza la velocitat de descompressió i proporciona un factor de compressió més elevat que Snappy. Igual que l'anterior s'ha d'instal·lar manualment a cada *region server*.
- **GZ**: més lent que els anteriors comprimint i descomprimint però ofereix un factor de compressió més elevat que els altres dos. A més a més, està incorporat en les llibreries de Hadoop.
  Un desavantatge important és que aquest algorisme utilitza gran quantitat de CPU, cosa que pot provocar càrrega no desitjada del sistema.

En aquest cas utilitzarem GZ ja que no requereix cap instal·lació extra.

Pel que fa a aquest model, al estar normalitzat no hi ha dades repetides i possiblement podrà aplicar poca compressió de dades, s'esperen uns resultats similars als del model 1.

Per a la creació de taules hi ha diferències respecte a les propietats:

```
create 'CUSTOMERS',{NAME=>'colfamINFO:', COMPRESSION=>'GZ'}
create 'CITIES',{NAME=>'colfamINFO:', COMPRESSION=>'GZ'}
create 'PRODUCTS',{NAME=>'colfamINFO:', COMPRESSION=>'GZ'}
create 'SALES',{NAME=>'colfamINFO:', COMPRESSION=>'GZ'}
```

Tant la creació de taules a Hive com les consultes són iguals que el model equivalent així que no es reescriuran.

5.5.6 Model 5

En aquest cas s'espera una millora molt significativa del temps de consulta ja que la compressió s'aplica a nivell de família de columna. Aquest model hauria de ser el que menys espai ocupa.

La creació de la taula es fa indicant a cada família de columnes per separat la compressió aplicada:

```
1 create 'SALES',{NAME=>'colfamPRODUCT', COMPRESSION=>'GZ'},{NAME=>'
    colfamCITY', COMPRESSION=>'GZ'},{NAME=>'colfamCUSTOMER', COMPRESSION=>
    'GZ'},{NAME=>'colfamSALE', COMPRESSION=>'GZ'}
```

5.5.7 Model 6

L'últim model, tot i aplicar compressió no s'espera que hi apliqui un factor de compressió gaire elevat degut a la distribució de dades aplicada. Així doncs, tot i esperar una ocupació de disc baixa no hauria de ser tant com el model 5. El mateix raonament s'hauria d'aplicar per al temps de consulta.

```
1 create 'SALES',{NAME=>'colfamSALE', COMPRESSION=>'GZ'}
```

5.6 Execució de les proves

La gran quantitat de combinacions de consultes per cada variant de model, ens obliga a automatitzar el procés d'execució de les proves. La primera idea era mantenir un mateix programa escrit en Java, que amb petites modificacions valgués tan per MySQL com per HBase, però degut a la poca maduresa i difícil sincronització de Hadoop, Hive i HBase va provocar un canvi de mètode.

5.6.1 Proves a MySQL Cluster

L'execució de proves a MySQL es fan a un programa Java, a partir de diferents fitxers on s'emmagatzemen les consultes es llancen 3 repeticions de cada una. Des del mateix programa un cop s'acaba tot el conjunt de consultes,

s'insereixen noves tuples per a tornar a repetir el procés.
L'execució de cada consulta implica executar el següent codi:

```
1 COMMIT;
2 ** Consulta SQL **
3 ROLLBACK;
4 RESET QUERY CACHE;
```

Després de diferents proves es va detectar que després d'executar múltiples consultes, el sistema perdia rendiment perquè no alliberava memòria. Al fer `COMMIT` i `ROLLBACK` aquest comportament desapareix.
D'altra banda, si no s'executa `RESET QUERY CACHE`, la segona i la tercera repetició de la consulta és immediata ja que MySQL emmagatzema els resultats en cache.

Al final del document, s'inclou un apèndix amb el codi d'execució de proves complet i detallat.

5.6.2 Proves a HBase

Degut a la dificultat d'executar les consultes desde codi Java a Hive, s'ha optat per fer-ho des del client de consola, més conegut com a `Apache CLI`, a partir d'un *shellscript*.
Aquest llegeix les consultes per a cada model concret i llança el client adjuntant la consulta com a paràmetre. Concretament, això s'aconsegueix mitjançant la comanda:

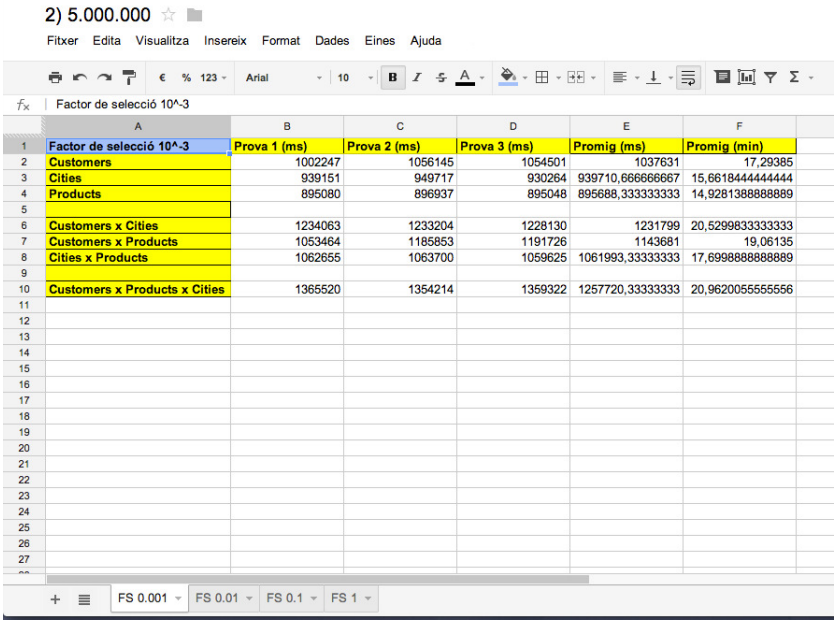
```
1 hive -f /home/serginadalfrancesch/consultes/sql.sql -S
```

El paràmetre `-f` indica el fitxer d'on llegir la consulta, en aquest previament s'hi ha afegit la consulta concreta. D'altra banda, el paràmetre `-S` activa el mode silenciós, d'aquesta forma el fitxer que conté la sortida amb els resultats no conté la sortida de les tasques MapReduce.

De la mateixa forma que amb MySQL, al final del document s'inclou un apèndix amb el codi complet detallat.

5.7 Anàlisi dels resultats

A mesura que es van executant proves per a diferents models i magnituds, els resultats s'han anat emmagatzemant en fitxers diferents tipus Excel tal com l'exemple de la següent figura.



2) 5.000.000

Fitxer Edita Visualitza Insereix Format Dades Eines Ajuda

Factor de selecció 10^-3

	A	B	C	D	E	F
1	Factor de selecció 10^-3	Prova 1 (ms)	Prova 2 (ms)	Prova 3 (ms)	Promig (ms)	Promig (min)
2	Customers	1002247	1056145	1054501	1037631	17,29385
3	Cities	939151	949717	930264	939710,666666667	15,6618444444444
4	Products	895080	896937	895048	895688,333333333	14,9281388888889
5						
6	Customers x Cities	1234063	1233204	1228130	1231799	20,5299833333333
7	Customers x Products	1053464	1185853	1191726	1143681	19,06135
8	Cities x Products	1062655	1063700	1059625	1061993,33333333	17,6998888888889
9						
10	Customers x Products x Cities	1365520	1354214	1359322	1257720,33333333	20,9620055555556
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						

FS 0.001 FS 0.01 FS 0.1 FS 1

Figura 5.11: Resultats per a 5.000.000 de tuples per al model 2 d'HBase.

Finalment, un cop recollides totes les dades cal analitzar-les per a poder-ne treure conclusions. Degut a la gran quantitat de resultats recollits cal cercar un mètode que ens permeti agrupar i comparar les dades de forma automàtica. Pel que fa al número de resultats recollits (K), estan definits per les combinacions dels diferents factors variables que tenim, concretament $K = Mo \times Ma \times S \times C \times R$. On les diferents variables són:

- Mo: nombre de models diferents.
- Ma: nombre de magnituds diferents aplicades.
- S: nombre de diferents factors de selecció aplicats.
- C: nombre de diferents consultes executades.
- R: nombre de repeticions per consulta.

En el nostre cas tenim, $Mo = 7$, $Ma = 4$, $S = 4$, $C = 7$ i $R = 3$, cosa que resulta en 2352 resultats emmagatzemats.

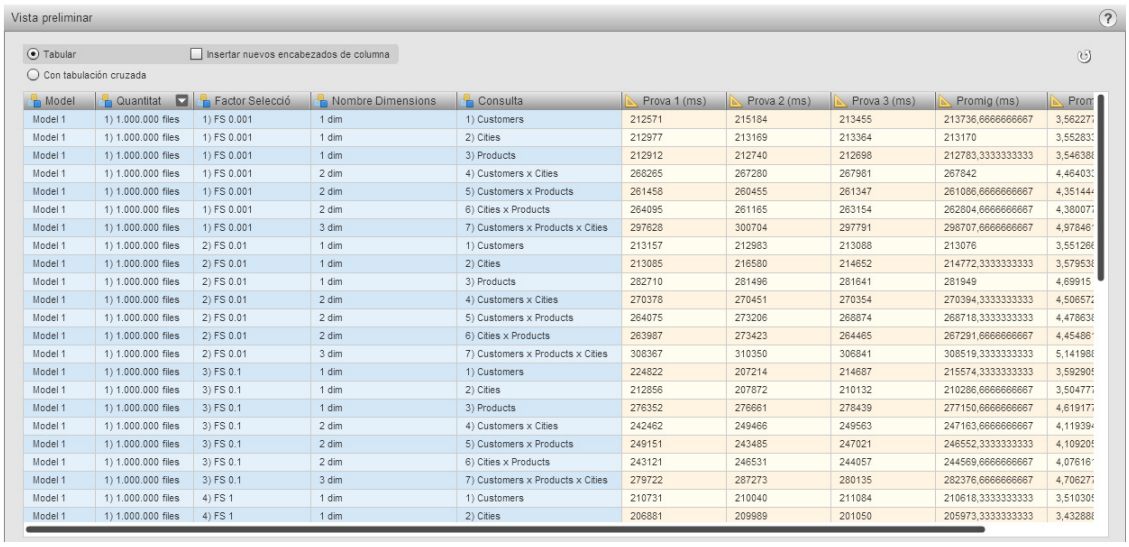
Per tal d'analitzar tota aquesta informació s'utilitzarà l'eina MicroStrategy Reporting Suite, la versió gratuïta de MicroStrategy. Aquesta és una

eina de *business intelligence*, i que gràcies a la seva funcionalitat *Excel Data Import* ens permetrà veure informes i gràfics sobre les dades recollides i unides utilitzant com a origen de dades un fitxer Excel.

	A	B	C	D	E	F	G	H	I	J
1	**Model**	**Quantitat**	**Factor selecció**	**Nombre dimensions**	**Consulta**	Prova 1 (ms)	Prova 2 (ms)	Prova 3 (ms)	Promig (ms)	Promig (min)
2	Model 1	1) 1.000.000 files	1) FS 0.001	1 dim	1) Customers	212571	215184	213455	213736,6667	3,562277778
3	Model 1	1) 1.000.000 files	1) FS 0.001	1 dim	2) Cities	212977	213169	213364	213170	3,552833333
4	Model 1	1) 1.000.000 files	1) FS 0.001	1 dim	3) Products	212912	212740	212698	212783,3333	3,546388889
5	Model 1	1) 1.000.000 files	1) FS 0.001	2 dim	4) Customers x Cities	268265	267280	267981	267842	4,464033333
6	Model 1	1) 1.000.000 files	1) FS 0.001	2 dim	5) Customers x Products	261458	260455	261347	261086,6667	4,351444444
7	Model 1	1) 1.000.000 files	1) FS 0.001	2 dim	6) Cities x Products	264095	261165	263154	262804,6667	4,380077778
8	Model 1	1) 1.000.000 files	1) FS 0.001	3 dim	7) Customers x Products x Cities	297628	300704	297791	298707,6667	4,978461111
9	Model 1	1) 1.000.000 files	2) FS 0.01	1 dim	1) Customers	213157	212983	213088	213076	3,551266667
10	Model 1	1) 1.000.000 files	2) FS 0.01	1 dim	2) Cities	213085	216580	214652	214772,3333	3,579538889
11	Model 1	1) 1.000.000 files	2) FS 0.01	1 dim	3) Products	282710	281496	281641	281949	4,69915
12	Model 1	1) 1.000.000 files	2) FS 0.01	2 dim	4) Customers x Cities	270378	270451	270354	270394,3333	4,506572222
13	Model 1	1) 1.000.000 files	2) FS 0.01	2 dim	5) Customers x Products	264075	273206	268874	268718,3333	4,478638889
14	Model 1	1) 1.000.000 files	2) FS 0.01	2 dim	6) Cities x Products	263987	273423	264465	267291,6667	4,454861111
15	Model 1	1) 1.000.000 files	2) FS 0.01	3 dim	7) Customers x Products x Cities	308367	310350	306841	308519,3333	5,141988889
16	Model 1	1) 1.000.000 files	3) FS 0.1	1 dim	1) Customers	224822	207214	214687	215574,3333	3,592905556
17	Model 1	1) 1.000.000 files	3) FS 0.1	1 dim	2) Cities	212856	207872	210132	210286,6667	3,504777778
18	Model 1	1) 1.000.000 files	3) FS 0.1	1 dim	3) Products	276352	276661	278439	277150,6667	4,619177778
19	Model 1	1) 1.000.000 files	3) FS 0.1	2 dim	4) Customers x Cities	242462	249466	249563	247163,6667	4,119394444
20	Model 1	1) 1.000.000 files	3) FS 0.1	2 dim	5) Customers x Products	249151	243485	247021	246552,3333	4,109205556
21	Model 1	1) 1.000.000 files	3) FS 0.1	2 dim	6) Cities x Products	243121	246531	244057	244569,6667	4,076161111
22	Model 1	1) 1.000.000 files	3) FS 0.1	3 dim	7) Customers x Products x Cities	279722	287273	280135	282376,6667	4,706277778
23	Model 1	1) 1.000.000 files	4) FS 1	1 dim	1) Customers	210731	210040	211084	210618,3333	3,510305556
24	Model 1	1) 1.000.000 files	4) FS 1	1 dim	2) Cities	206881	209989	201050	205973,3333	3,432888889
25	Model 1	1) 1.000.000 files	4) FS 1	1 dim	3) Products	228712	275931	234497	246380	4,106333333
26	Model 1	1) 1.000.000 files	4) FS 1	2 dim	4) Customers x Cities	243191	248119	245012	244774	4,079566667
27	Model 1	1) 1.000.000 files	4) FS 1	2 dim	5) Customers x Products	249279	249176	249088	249181	4,153016667
28	Model 1	1) 1.000.000 files	4) FS 1	2 dim	6) Cities x Products	249854	243232	248834	247306,6667	4,121777778
29	Model 1	1) 1.000.000 files	4) FS 1	3 dim	7) Customers x Products x Cities	284364	289032	287234	286876,6667	4,781277778
30	Model 1	2) 5.000.000 files	1) FS 0.001	1 dim	1) Customers	829157	825863	827035	827351,6667	13,78919444
31	Model 1	2) 5.000.000 files	1) FS 0.001	1 dim	2) Cities	777006	777300	776850	777052	12,95086667
32	Model 1	2) 5.000.000 files	1) FS 0.001	1 dim	3) Products	819868	573858	818560	737428,6667	12,29047778
33	Model 1	2) 5.000.000 files	1) FS 0.001	2 dim	4) Customers x Cities	611290	616389	615044	614241	10,23735
34	Model 1	2) 5.000.000 files	1) FS 0.001	2 dim	5) Customers x Products	630070	882363	876455	796296	13,2716
35	Model 1	2) 5.000.000 files	1) FS 0.001	2 dim	6) Cities x Products	823024	861283	854561	846289,3333	14,10482222
36	Model 1	2) 5.000.000 files	1) FS 0.001	3 dim	7) Customers x Products x Cities	918926	911854	917729	916169,6667	15,26949444
37	Model 1	2) 5.000.000 files	2) FS 0.01	1 dim	1) Customers	823026	816027	819841	819631,3333	13,66052222
38	Model 1	2) 5.000.000 files	2) FS 0.01	1 dim	2) Cities	774670	741699	751056	755808,3333	12,59680556
39	Model 1	2) 5.000.000 files	2) FS 0.01	1 dim	3) Products	825942	813054	818711	819235,6667	13,65392778
40	Model 1	2) 5.000.000 files	2) FS 0.01	2 dim	4) Customers x Cities	855491	621772	846970	774744,3333	12,91240556
41	Model 1	2) 5.000.000 files	2) FS 0.01	2 dim	5) Customers x Products	871288	973221	869011	871173,3333	14,51955556
42	Model 1	2) 5.000.000 files	2) FS 0.01	2 dim	6) Cities x Products	831158	832399	841025	834860,6667	13,91434444
43	Model 1	2) 5.000.000 files	2) FS 0.01	3 dim	7) Customers x Products x Cities	891607	886266	892500	890124,3333	14,83540556
44	Model 1	2) 5.000.000 files	3) FS 0.1	1 dim	1) Customers	837179	814147	822005	824443,6667	13,74072778
45	Model 1	2) 5.000.000 files	3) FS 0.1	1 dim	2) Cities	779550	783911	778001	780487,3333	13,00812222
46	Model 1	2) 5.000.000 files	3) FS 0.1	1 dim	3) Products	814478	830004	819185	821222,3333	13,68703889

Figura 5.12: Fragment de l'Excel amb els resultats de les proves units.

L'eina és descarregable des de la web de MicroStrategy, un cop instal·lada i des del client web podem crear un cub a partir d'un fitxer Excel. Aquest emmagatzema les dades que hi ha a l'Excel en aquell moment, és important tenir en compte que no s'actualitza quan el fitxer es modifica.



Model	Quantitat	Factor Selecció	Nombre Dimensions	Consulta	Prova 1 (ms)	Prova 2 (ms)	Prova 3 (ms)	Promig (ms)	Prom
Model 1	1) 1.000.000 files	1) FS 0.001	1 dim	1) Customers	212571	215184	213455	213736,6666666667	3,562271
Model 1	1) 1.000.000 files	1) FS 0.001	1 dim	2) Cities	212977	213169	213364	213170	3,552833
Model 1	1) 1.000.000 files	1) FS 0.001	1 dim	3) Products	212912	212740	212698	212783,3333333333	3,546388
Model 1	1) 1.000.000 files	1) FS 0.001	2 dim	4) Customers x Cities	268265	267280	267981	267842	4,464033
Model 1	1) 1.000.000 files	1) FS 0.001	2 dim	5) Customers x Products	261458	260455	261347	261086,6666666667	4,351444
Model 1	1) 1.000.000 files	1) FS 0.001	2 dim	6) Cities x Products	264095	261165	263154	262804,6666666667	4,380077
Model 1	1) 1.000.000 files	1) FS 0.001	3 dim	7) Customers x Products x Cities	297628	300704	297791	298707,6666666667	4,978461
Model 1	1) 1.000.000 files	2) FS 0.01	1 dim	1) Customers	213157	212983	213088	213076	3,551266
Model 1	1) 1.000.000 files	2) FS 0.01	1 dim	2) Cities	213085	216580	214652	214772,3333333333	3,579538
Model 1	1) 1.000.000 files	2) FS 0.01	1 dim	3) Products	282710	281496	281641	281949	4,69915
Model 1	1) 1.000.000 files	2) FS 0.01	2 dim	4) Customers x Cities	270378	270451	270354	270394,3333333333	4,506572
Model 1	1) 1.000.000 files	2) FS 0.01	2 dim	5) Customers x Products	264075	273206	268874	268718,3333333333	4,478636
Model 1	1) 1.000.000 files	2) FS 0.01	2 dim	6) Cities x Products	263987	273423	264465	267291,6666666667	4,454861
Model 1	1) 1.000.000 files	2) FS 0.01	3 dim	7) Customers x Products x Cities	308367	310350	306841	308519,3333333333	5,141988
Model 1	1) 1.000.000 files	3) FS 0.1	1 dim	1) Customers	224822	207214	214687	215574,3333333333	3,592905
Model 1	1) 1.000.000 files	3) FS 0.1	1 dim	2) Cities	212856	207872	210132	210286,6666666667	3,504777
Model 1	1) 1.000.000 files	3) FS 0.1	1 dim	3) Products	276352	276661	278439	277150,6666666667	4,619177
Model 1	1) 1.000.000 files	3) FS 0.1	2 dim	4) Customers x Cities	242462	249486	249563	247163,6666666667	4,119394
Model 1	1) 1.000.000 files	3) FS 0.1	2 dim	5) Customers x Products	249151	243485	247021	246552,3333333333	4,109205
Model 1	1) 1.000.000 files	3) FS 0.1	2 dim	6) Cities x Products	243121	246531	244057	244569,6666666667	4,076161
Model 1	1) 1.000.000 files	3) FS 0.1	3 dim	7) Customers x Products x Cities	279722	287273	280135	282376,6666666667	4,706277
Model 1	1) 1.000.000 files	4) FS 1	1 dim	1) Customers	210731	210040	211084	210618,3333333333	3,510305
Model 1	1) 1.000.000 files	4) FS 1	1 dim	2) Cities	206881	209989	201050	205973,3333333333	3,432888

Figura 5.13: Previsualització de l'Excel carregat a MicroStrategy.

Finalment, un cop generat el cub de dades ja es pot crear qualsevol tipus d'informe o gràfic d'entre els disponibles. En aquest cas, les diferents variables previament comentades s'han creat com a dimensions, mentre que els valors dels resultats com a mesures o indicadors.

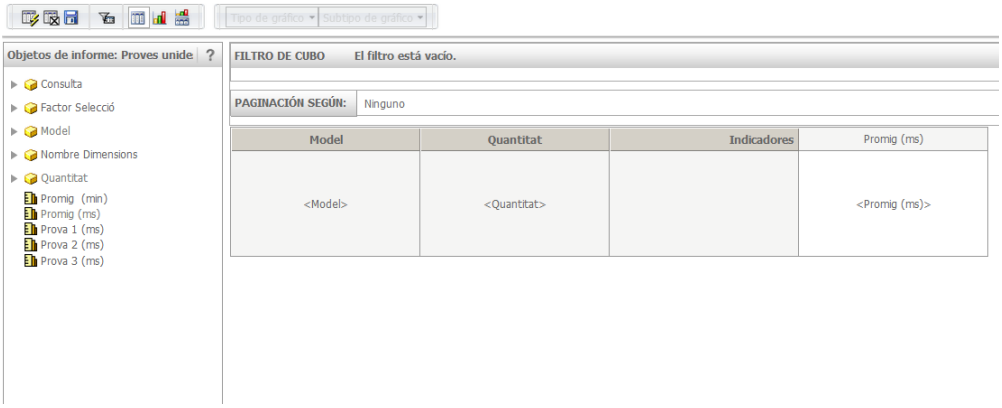


Figura 5.14: Generació d'informes a partir de dimensions i indicadors.

Capítol 6

Resultats

En aquest capítol es mostraran detalladament els resultats recollits per a cada model. Es descriuran les diferències entre cadascun i s'intentaran buscar raons per justificar els diferents resultats obtinguts respecte el seu predecessor. Finalment es farà una comparació global i se'n treuran conclusions.

6.1 Model relacional

El primer model a avaluar correspon al model relacional emmagatzemat a MySQL Cluster. Les taules només estan indexades per clau primària, donat que les consultes cerquen un camp que no està indexat no se'n podran aprofitar.
Això no significa que a MySQL Cluster no es puguin indexar camps que no són clau primària, però s'ha fet així per igualar les condicions amb HBase.

6.1.1 Hipòtesis

S'espera un molt bon temps de consulta per a magnituds baixes, però que aquest empitjori molt per a magnituds altes. MySQL Cluster emmagatzema les tuples a memòria, però com que s'utilitzaran *disk data objects* com es comentava al capítol 5, no es podrà aprofitar aquest fet i s'espera que el rendiment disminueixi.
D'altra banda, també s'espera que el factor de selecció aplicat i el nombre de joins de la consulta afectin al temps d'execució.

Pel que fa a l'espai ocupat, aquest hauria de ser baix ja que la taula de fets només emmagatzema referències a les de dimensions.

6.1.2 Resultats obtinguts

Un dels principals problemes detectats a MySQL és que en els pitjors casos, és a dir consultes amb factor de selecció elevat i molta magnitud, el servidor causava una excepció de memòria i s'aturava provocant un reinici d'aquest. Possiblement aquesta excepció és deguda als resultats intermitjos de consulta emmagatzemats en la mateixa memòria. Donat que el servidor *mysqld* resideix al node master, és possible que s'estigui fent *data shipping* a aquest i la memòria quedi completa.
Aquest fet s'ha quantificat mitjançant un temps d'execució molt gran, concretament 99999999 ms. En els gràfics, donat que la unitat emprada són minuts, el valor resulta en 1666 minuts. Així doncs, no s'ha de pendre aquest valor com un temps d'execució sinó com una marca d'aturada, cap de les proves executades en aquest projecte té una durada més gran de 800 minuts. En la figura 6.2 hi ha un exemple d'aquest cas descrit.

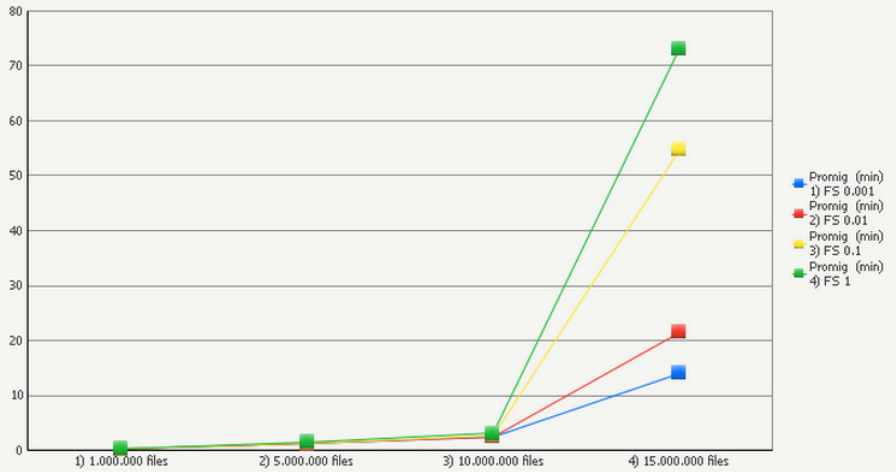


Figura 6.1: 1 dimensió i tots els factors de selecció.

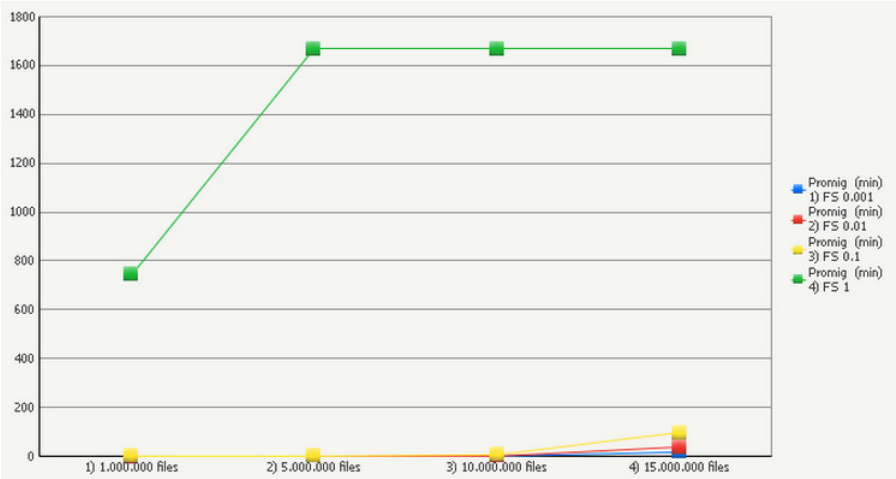


Figura 6.2: 3 dimensions i tots els factors de selecció.

D'altra banda, el nombre de joins aplicades afecta com es pot veure en les figures 6.3 i 6.4, es formen tres grups amb temps clarament diferenciats.

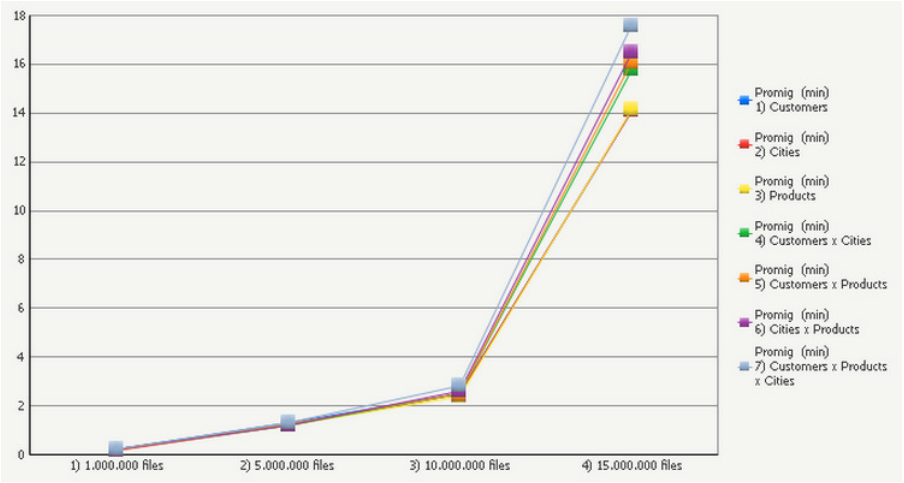


Figura 6.3: Factor de selecció 10^{-3} i totes les consultes.

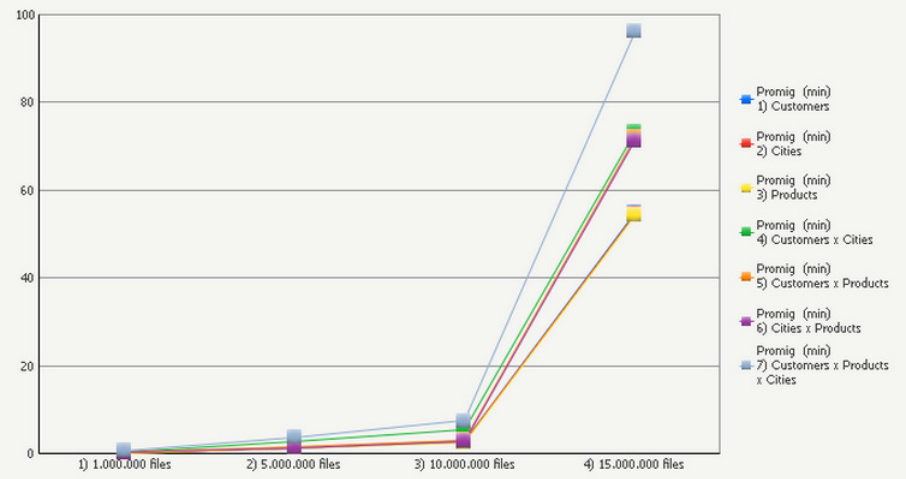


Figura 6.4: Factor de selecció 10^{-1} i totes les consultes.

Pel que fa a l'espai, aquest és baix com s'esperava i segueix un creixement lineal. No ha sigut possible extreure un informe de la distribució de dades que ha fet MySQL, ja que la consola d'administració no disposa d'aquesta funcionalitat.

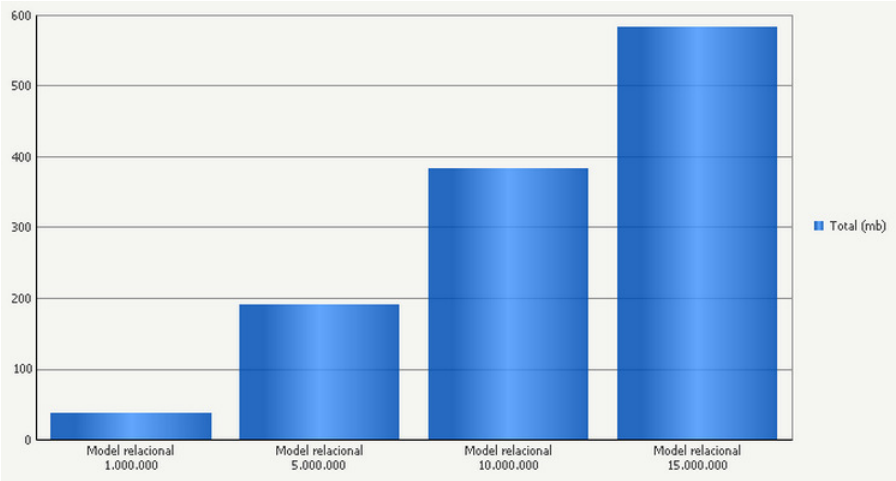


Figura 6.5: Espai ocupat per cada magnitud de tuples.

6.2 Model 1

Recordem que el model 1 és una traducció directa del model relacional, cada dimensió correspon a una taula diferent dins HBase amb una família de columnes. La taula de fets conté una sola família de columnes amb les referències a les dimensions a partir de la seva *row key*.

6.2.1 Hipòtesis

S'espera un temps de consulta elevat ja que s'ha de fer join entre taules. Pel que fa a la magnitud s'espera que segueixi el mateix comportament que el model relacional i l'augment de temps de consulta no es correspongui a una funció lineal.
Pel que fa al nombre de joins i el factor de selecció, també s'espera que siguin factors que augmentin el temps de consulta.
En resum, referint-nos a temps de consulta, a priori hauria de ser el pitjor model.

S'espera que l'espai ocupat sigui baix degut a la normalització del model.

6.2.2 Resultats obtinguts

Unint tots els factors de selecció, veiem que no té cap efecte augmentar el factor de selecció aplicat. En una mateixa quantitat, tots els factors de selecció tenen el mateix temps d'execució. Podem veure-ho en les figures 6.6, 6.7 i 6.8.

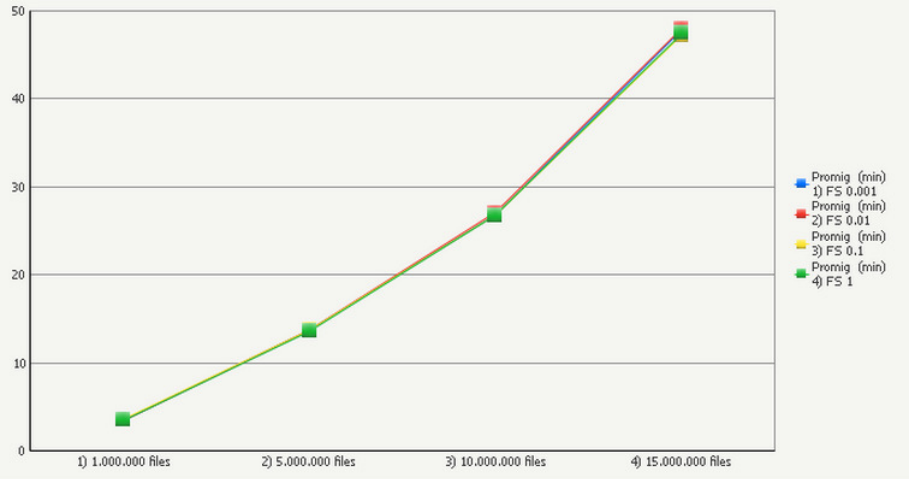


Figura 6.6: 1 dimensió i tots els factors de selecció.

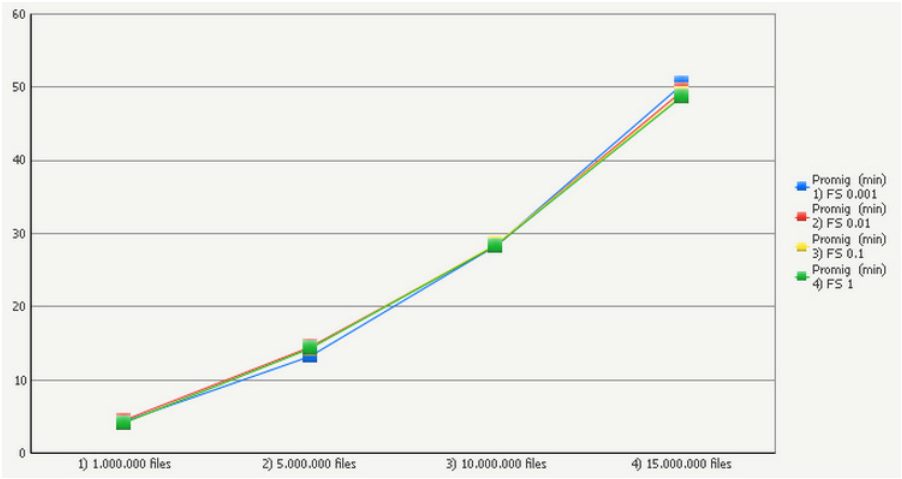


Figura 6.7: 2 dimensions i tots els factors de selecció.

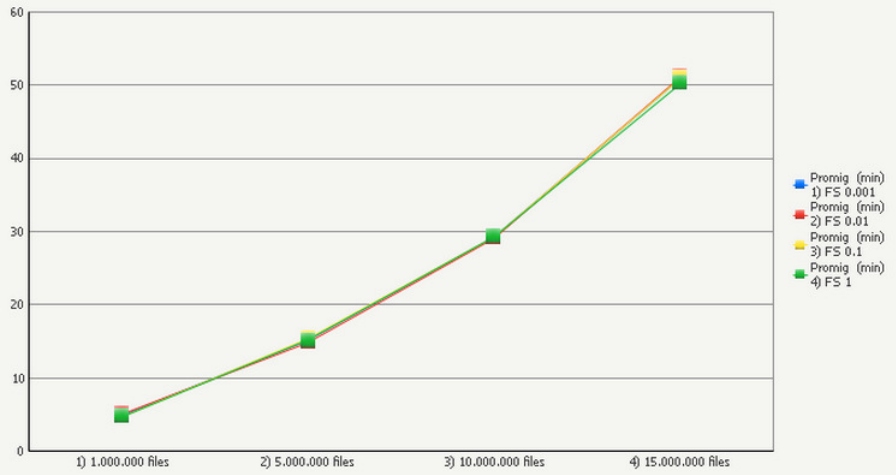


Figura 6.8: 3 dimensions i tots els factors de selecció.

D'altra banda, i al contrari del que s'esperava, sembla que el nombre de joins per consulta tampoc té cap efecte. Aquest fet es pot veure en les figures 6.9 i 6.10, amb el mínim i el màxim factor de selecció.

L'efecte del nombre de joins, com l'anterior amb el factor de selecció pot tenir explicació en el pla d'accés que utilitza Hive.
D'acord amb la documentació oficial de Hive, sempre s'aplica la operació de JOIN abans de la clàusula WHERE. Donat que les taules amb les que es fa JOIN són molt petites comparada amb la taula de fets, el nombre de joins aplicades pot tenir molt poc pes.

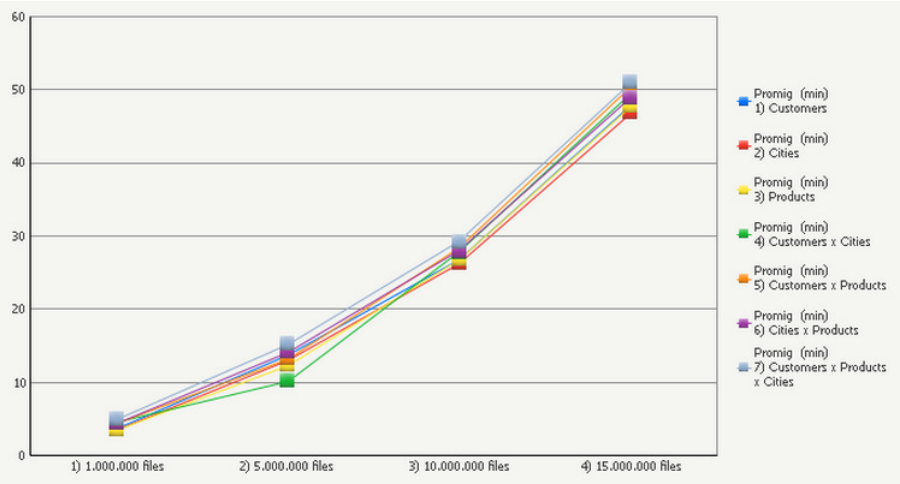


Figura 6.9: Factor de selecció 10^{-3} i totes les consultes.

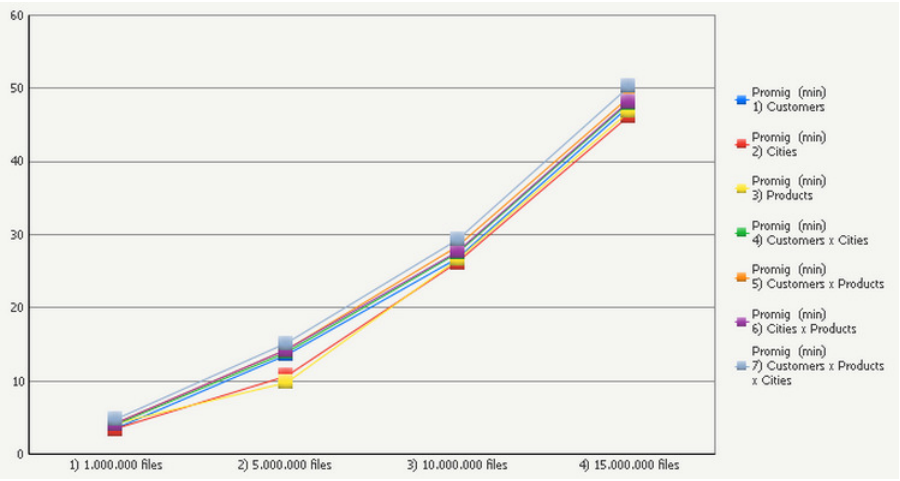


Figura 6.10: Factor de selecció 10^{0} i totes les consultes.

En els gràfics 6.9 i 6.10 es pot apreciar com la hipòtesi de que el creixement no és lineal segons la quantitat també és certa. Si comprovem un cas concret, el de la figura 6.11, veiem com el l'augment és cada cop major. Concretament, la diferència entre 1.000.000 i 5.000.000 de tuples són uns 6 minuts, mentre que als 10.000.000 de tuples ja en són uns 15 i, finalment per a 15.000.000 de tuples aproximadament 25 minuts.

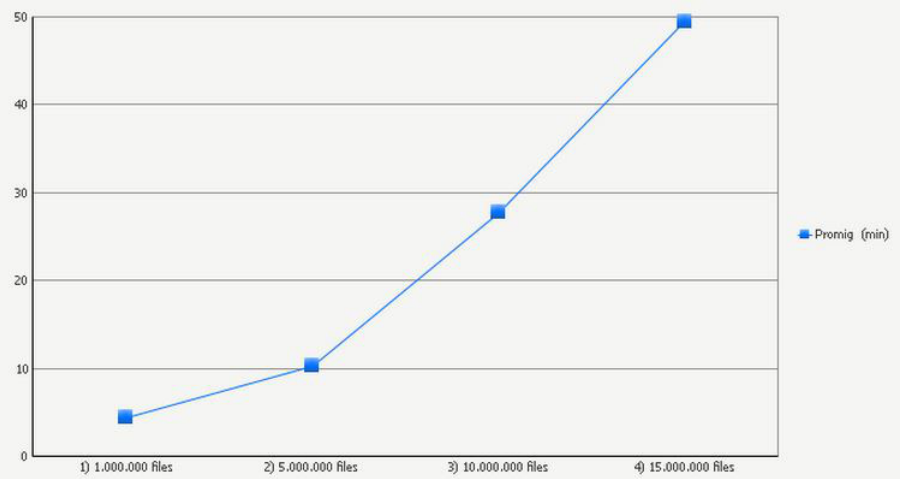


Figura 6.11: Factor de selecció 10^{-3} i consulta sobre 2 dimensions.

Pel que fa a l'espai ocupat, augmenta força respecte el model relacional. Com es pot veure en la figura 6.12, aproximadament l'augment de 5.000.000 de tuples equival a 1GB d'espai a disc.

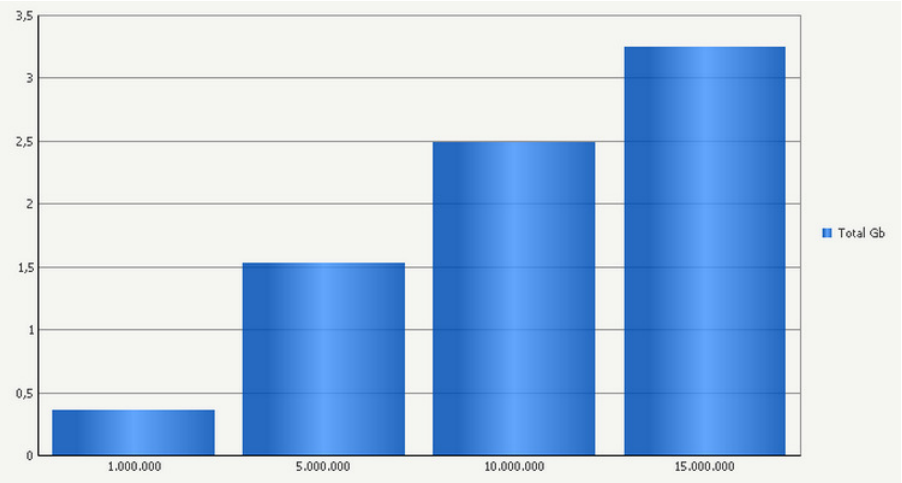


Figura 6.12: Espai ocupat per cada magnitud de tuples.

Finalment, una de les raons de pes que explicarien el mal rendiment de les consultes, és la mala distribució de dades entre els diferents nodes. Com es pot veure en la figura 6.13, la gran majoria de dades queden emmagatzemades en un sol node, cosa que provoca un coll d'ampolla a l'hora d'accedir-hi.

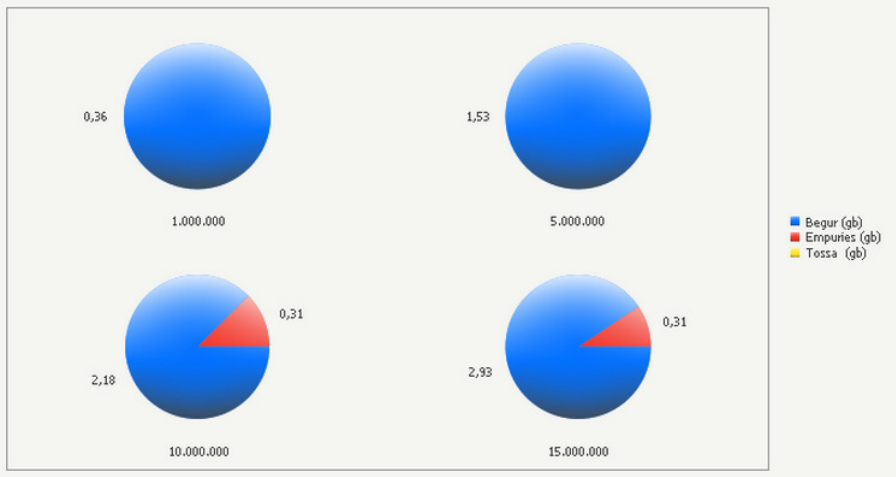


Figura 6.13: Distribució de dades entre els nodes.

6.3 Model 2

El segon model és el resultat d'haver aplicat una denormalització al primer model, en aquest cas tota la informació està emmagatzemada en una sola taula amb 4 famílies de columnes diferents, una per cada dimensió.

6.3.1 Hipòtesis

Gràcies a la denormalització ens estalviem fer joins, així doncs esperem obtenir un millor temps de consulta respecte el primer model. D'altra banda, referent al nombre de joins (o de dimensions consultades), s'espera que sigui un factor crític ja que teòricament si no es consulta una dimensió es pot prescindir de llegir completament una família de columnes.
D'altra banda, s'espera que en funció de la magnitud i el factor de selecció augmenti el temps de consulta.

Pel que fa a l'espai ocupat, s'espera que sigui el model que més espai ocupa degut a la denormalització i a la necessitat de mantenir múltiples famílies de columnes.

6.3.2 Resultats obtinguts

Per factors de selecció baixos els resultats són bons, i sembla que no afecta la magnitud de tuples o la consulta aplicada. D'altra banda, per a factors de

selecció alts el rendiment empitjora dràsticament, aparentment el creixement és exponencial.

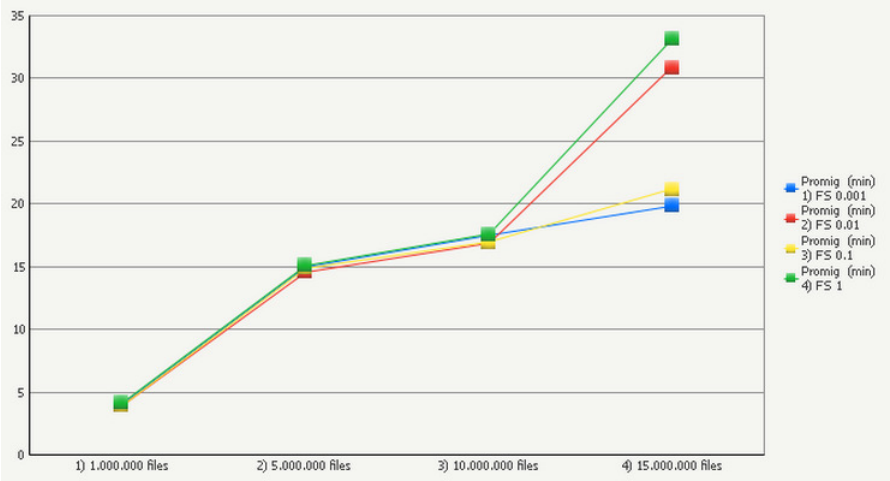


Figura 6.14: 1 dimensió i tots els factors de selecció.

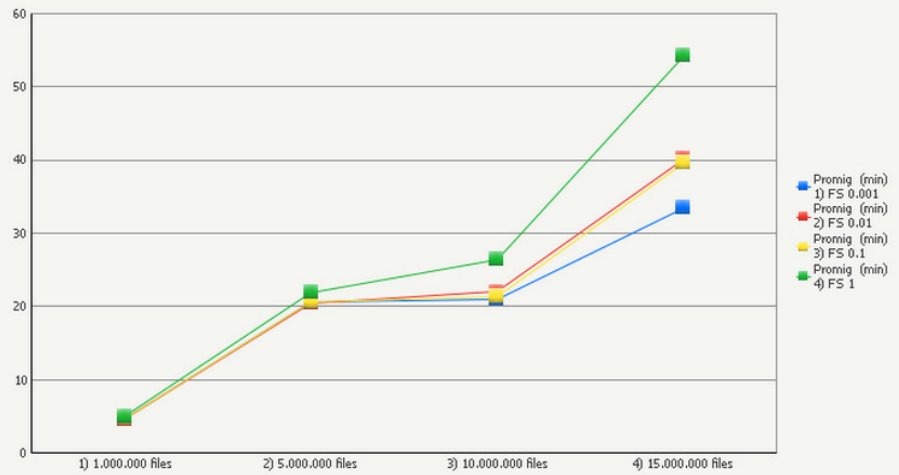


Figura 6.15: 2 dimensions i tots els factors de selecció.

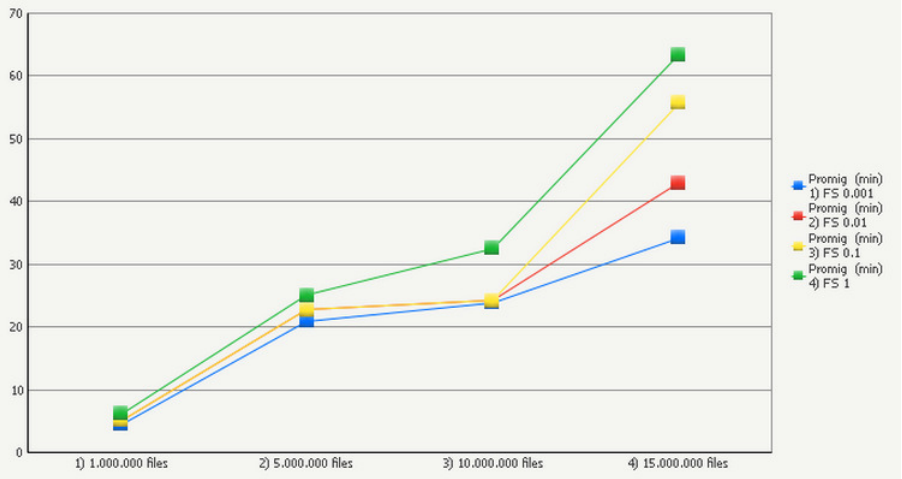


Figura 6.16: 3 dimensions i tots els factors de selecció.

El tipus de consulta aplicada no impacta tant el rendiment com el factor de selecció com es pot veure en les figures 6.17 i 6.18. El factor de selecció segueix sent decisiu.
Així doncs, la hipòtesi de que en funció de la consulta el rendiment augmenta o disminueix molt és falsa. En aquest cas no es pot donar la mateixa explicació del model 1, les consultes pel model 2 no contenen joins. Donada que la integració de Hive amb HBase encara és poc madura i en constant evolució, no hi ha informació sobre el pla d'accés que utilitza Hive per a múltiples famílies de columnes. Així doncs, no podem oferir una explicació sobre el perquè ocorre aquest fet.

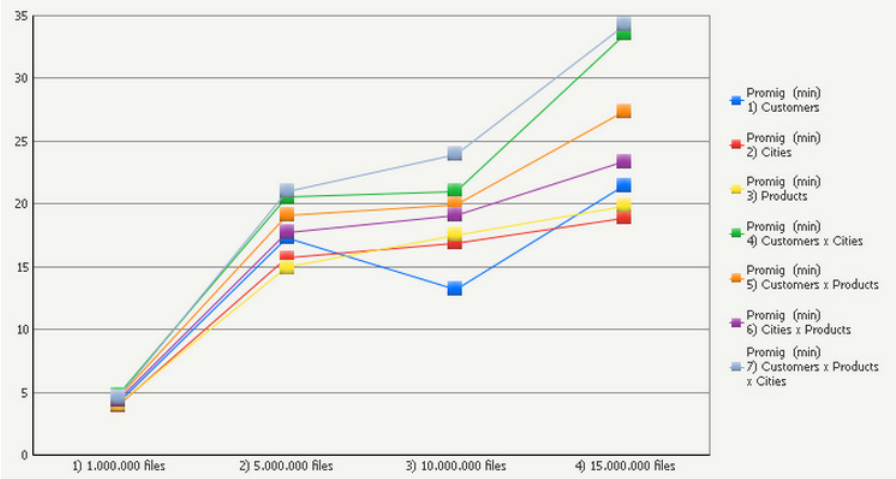


Figura 6.17: Factor de selecció 10^{-3} i totes les consultes.

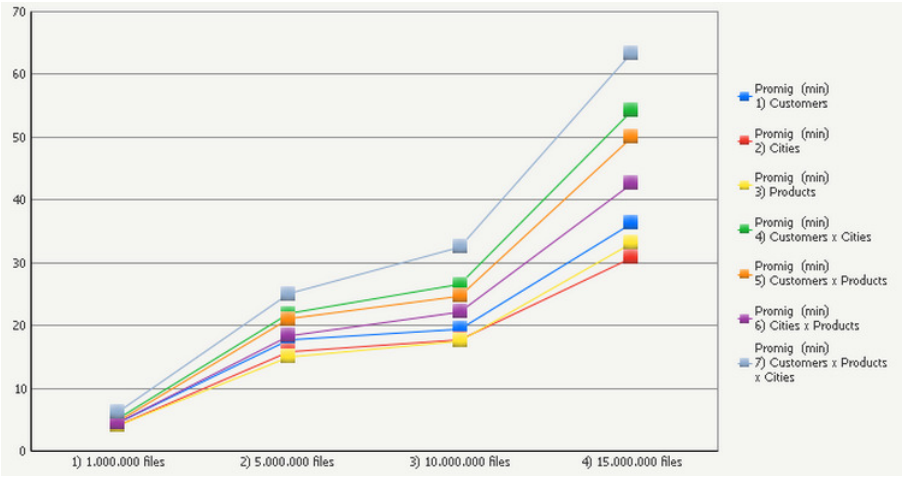


Figura 6.18: Factor de selecció 10^{0} i totes les consultes.

Pel que fa a l'espai de disc, es confirma l'hipòtesi que aquest augmenta dràsticament respecte el model 1.

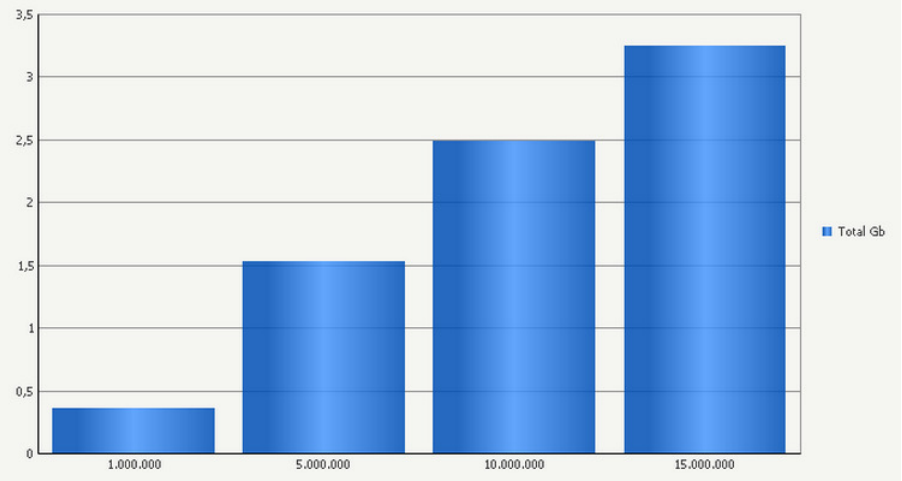


Figura 6.19: Espai ocupat per cada magnitud de tuples.

Finalment, la distribució de dades segueix sent un dels principals problemes que ens pot provocar colls d'ampolla. Sembla que com més augmenta l'espai ocupat la distribució es va repartint més equitativament, però s'haurien de fer proves amb molt més espai que no pas uns pocs GB per a comprovar-ho.

Un fet curiós, i preocupant, és la redistribució de dades que fa HBase, com es pot veure en la figura 6.20, al passar d'1.000.000 de tuples a 5.000.000 moltes dades han canviat de node. Això pot provocar sobrecàrregues del sistema i la xarxa innecessàries.

Figura 6.20: Distribució de dades entre els nodes.

6.4 Model 3

Aquest model és una evolució del model 2. Continua amb la denormalització de dades, però en aquest cas no es mantenen múltiples famílies de columnes sinó que totes les dades es guarden en una sola família de columnes.

6.4.1 Hipòtesis

Per una banda, s'espera baixar el temps de consulta respecte el model 1 ja que tampoc hi ha operacions join. D'altra banda però, ara fer qualsevol consulta requereix recórrer la taula sencera ja que les columnes estan barrejades a disc, així doncs s'espera un empitjorament de temps de consulta respecte el model 2 en funció del nombre de joins per consulta.

Pel que fa a utilització de disc, s'espera que l'espai ocupat disminueixi respecte al model 2 degut a que només s'ha de mantenir una família de columnes.

6.4.2 Resultats obtinguts

Tal com s'esperava, tots els temps han empitjorat respecte al model 2. D'altra banda, ara el creixement no és exponencial sinó que és lineal com es pot veure en les figures 6.21, 6.22 i 6.23. Això ens porta a pensar que amb més tuples hi haurà un punt de tall on a partir d'aquí aquest model doni millors resultats que el model 2.

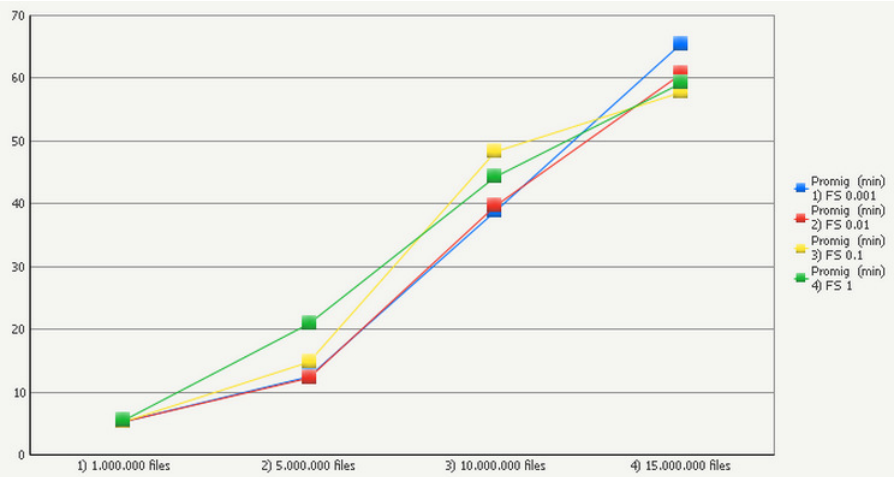


Figura 6.21: 1 dimensió i tots els factors de selecció.

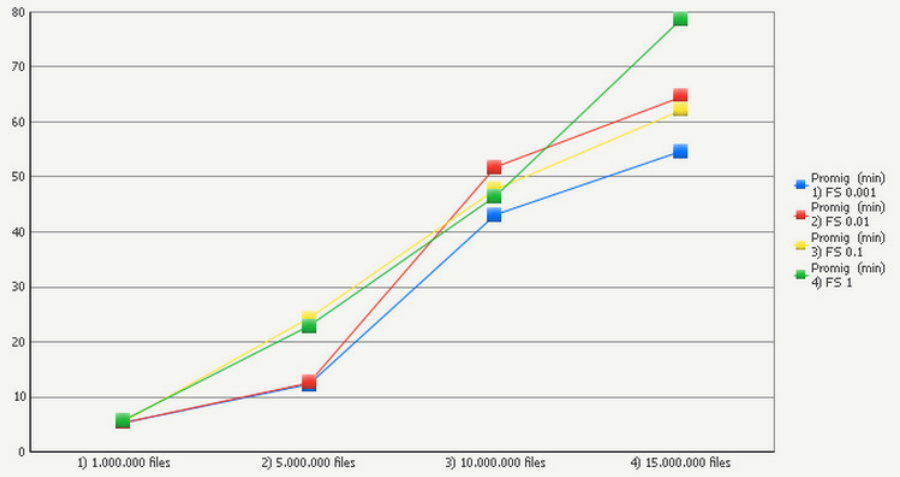


Figura 6.22: 2 dimensions i tots els factors de selecció.

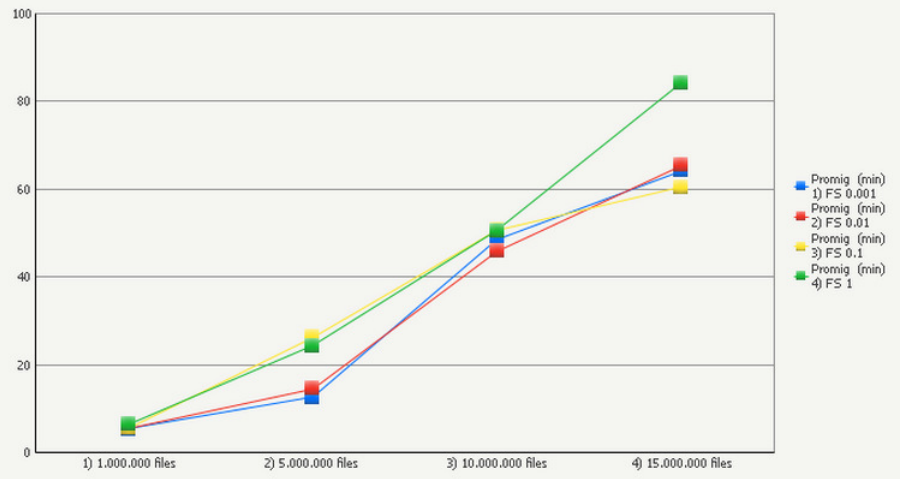


Figura 6.23: 3 dimensions i tots els factors de selecció.

El temps de consulta per a magnituds baixes es manté constant al llarg de tots els tipus de consulta, aquest varia tal com augmenta la magnitud.

En alguns casos, com es pot veure en els diagrames de barres de les figures 6.24 i 6.25, és més ràpida una consulta sobre 2 dimensions que sobre 1. Donat que és una desviació baixa no se li donarà més importància, possiblement es deu a l'organització de les dades i el pla d'accés emprat per accedir a aquestes.

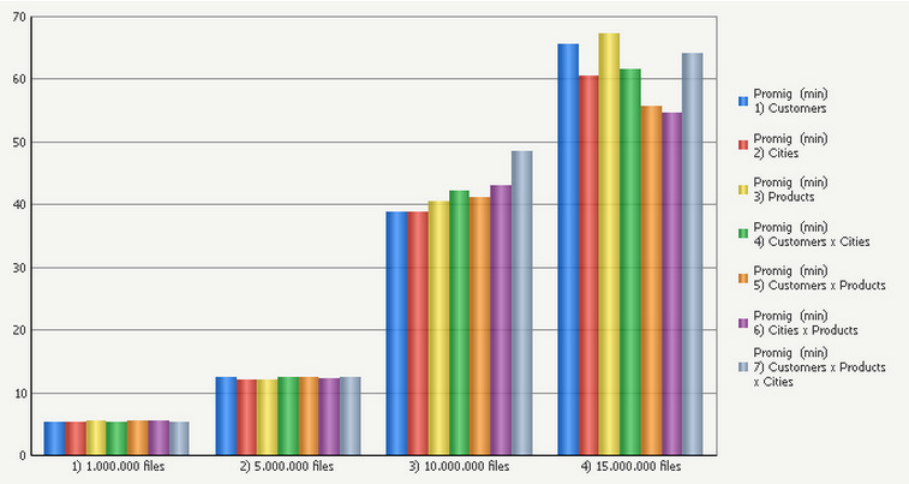


Figura 6.24: Factor de selecció 10^{-3} i totes les consultes.

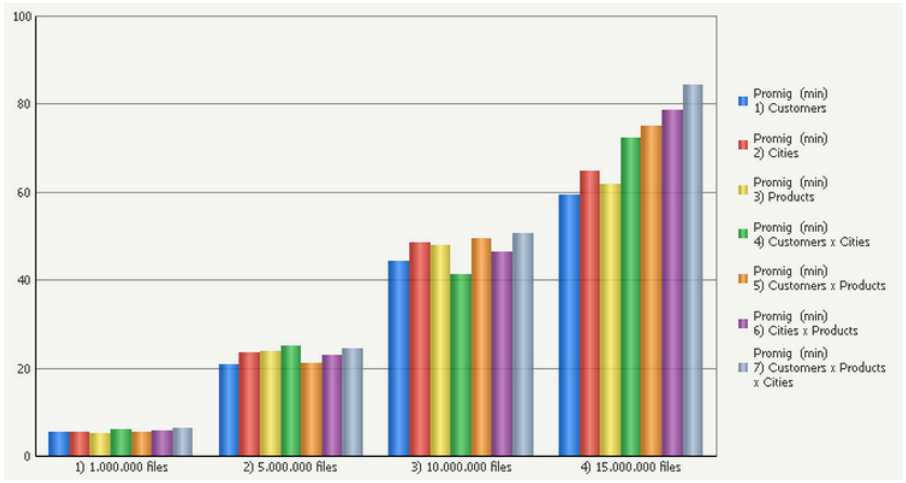


Figura 6.25: Factor de selecció 10^{0} i totes les consultes.

D'altra banda, pel que fa a l'espai ocupat es confirma la hipòtesi de que s'utilitza menys espai de disc que el model 2. Un fet a tenir en compte és que cada nou fragment de tuples sembla que ocupa menys espai, com es pot veure clarament en la figura 6.26, això possiblement està lligat amb la distribució de dades i la gestió de famílies de columnes per part d'HBase.

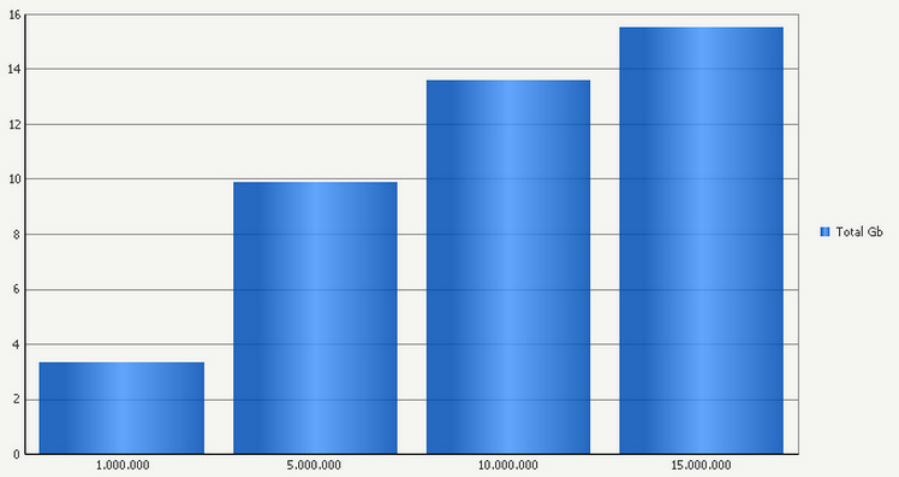


Figura 6.26: Espai ocupat per cada magnitud de tuples.

Finalment, pel que fa a la distribució de dades seguim tenint el mateix problema de mala distribució que amb els altres models. De totes formes, sembla que es confirma la hipòtesi plantejada en el model 2 que amb més dades la distribució millora.

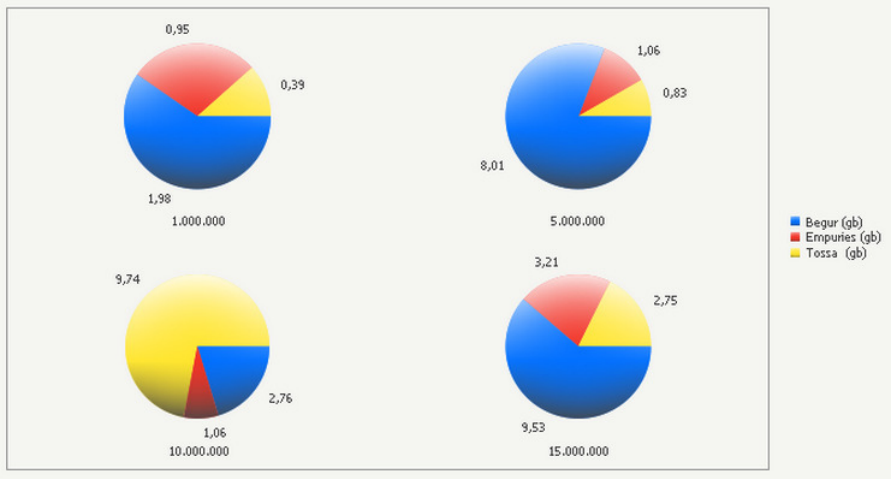


Figura 6.27: Distribució de dades entre els nodes.

6.5 Model 4

A partir d'ara, aquest model i els següents equivalen als anteriors però amb l'opció de compressió activada. En aquest cas, el model 4 és equivalent al model 1.

6.5.1 Hipòtesis

Al comprimir, esperem que l'espai ocupat respecte al model 1 baixi. D'altra banda, també s'espera que a l'haver de llegir menys quantitat de dades, el temps de consulta també baixi respecte al model 1.

6.5.2 Resultats obtinguts

Per a quantitats baixes s'obtenen millors temps de consulta que el model 1, però a l'augmentar-la empitjora dràsticament i supera el model 1. Es pot veure clarament en les figures 6.28, 6.29 i 6.30, on s'uneixen els factors de selecció.

Val la pena destacar un fet interessant referent als factors de selecció. Com es pot veure, per a quantitats baixes el factor de selecció és totalment independent del temps de consulta, però de sobte per a 10.000.000 de tuples es veu clarament una diferència. D'altra banda, per a 15.000.000 de tuples tornen a unir-se els temps de consulta pel que sembla que el factor de selecció no torna a tenir efecte sobre el temps de consulta.
Aquest fet es podria explicar amb la mala distribució de dades, però com veurem més endavant aquesta no canvia de 10.000.000 a 15.000.000 de tuples.

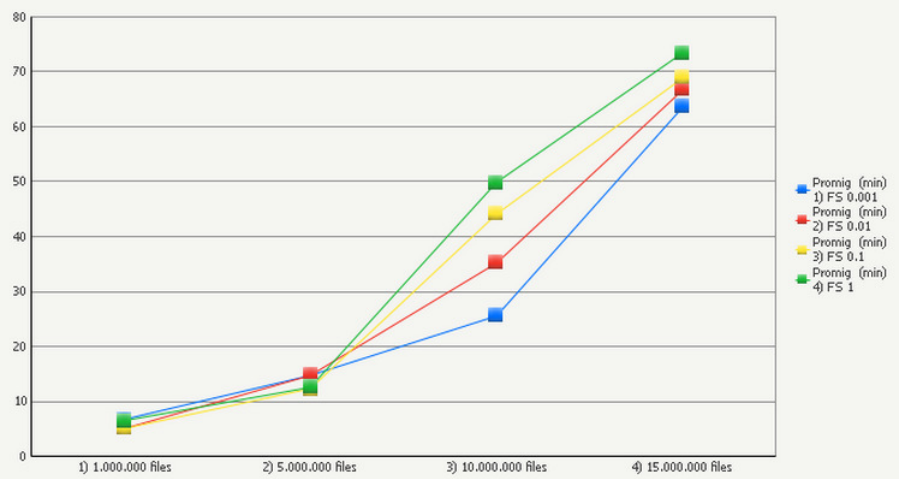


Figura 6.28: 1 dimensió i tots els factors de selecció.

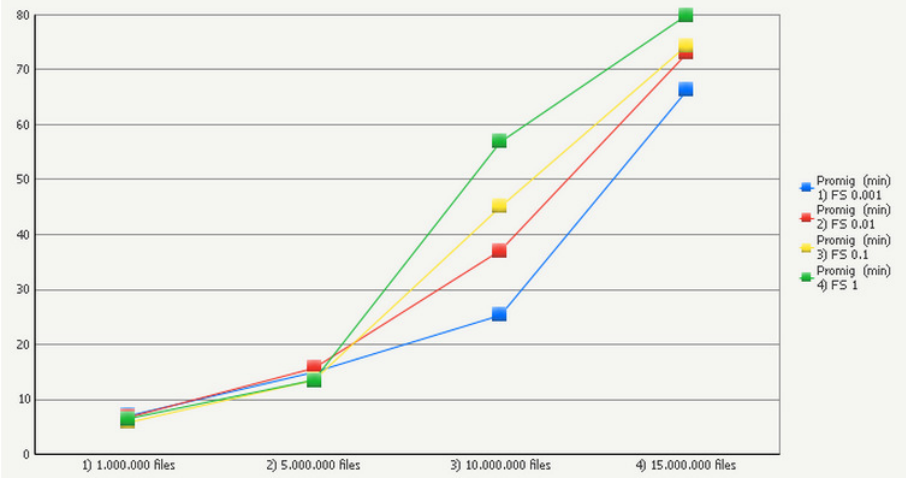


Figura 6.29: 2 dimensions i tots els factors de selecció.

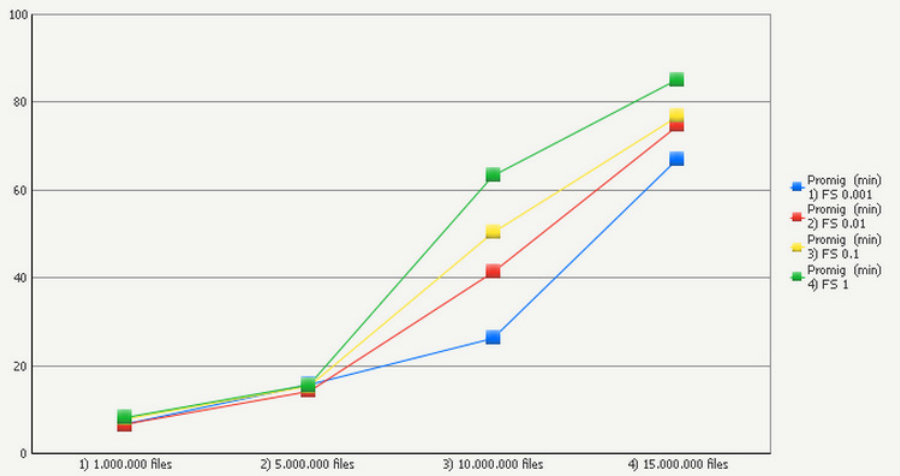


Figura 6.30: 3 dimensions i tots els factors de selecció.

D'altra banda, com es pot veure en les figures 6.31 i 6.32, pel que fa a les consultes al igual que el model 1 sembla que el nombre de joins no afecta al temps d'execució, només la magnitud de tuples i el factor de selecció.

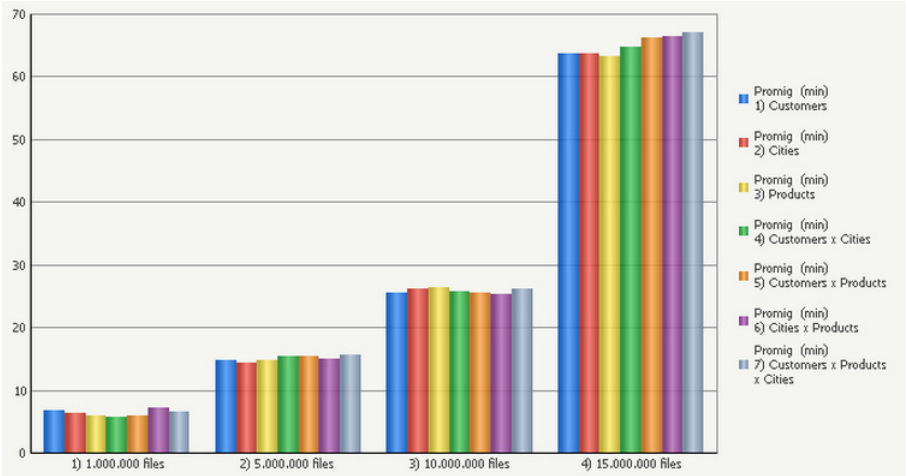


Figura 6.31: Factor de selecció 10^{-3} i totes les consultes.

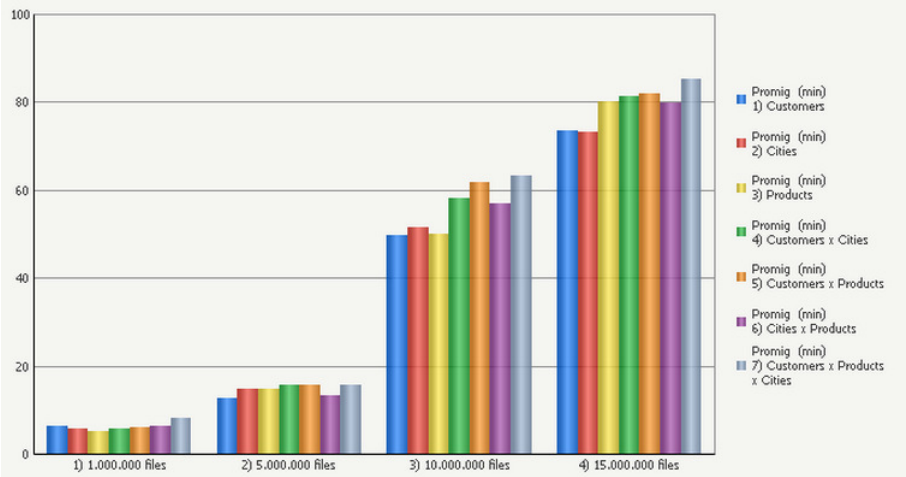


Figura 6.32: Factor de selecció 10^0 i totes les consultes.

Pel que fa a l'espai ocupat, es confirma l'hipòtesi plantejada de que aquest baixa. Concretament, la disminució d'espai resulta en un 15% de l'espai ocupat per el model 1.
Aquest és el model on el guany d'espai és més elevat.

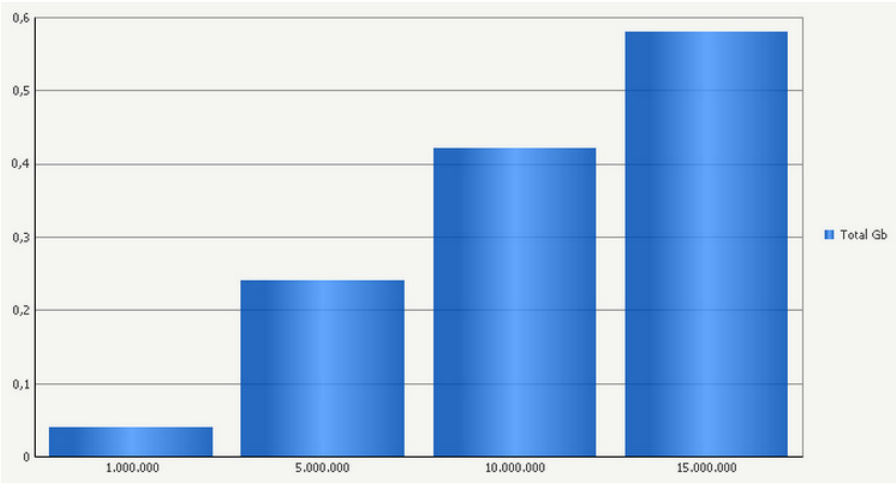


Figura 6.33: Espai ocupat per cada magnitud de tuples.

Finalment, pel que fa a la distribució de dades al tenir moltes menys dades el problema de la mala distribució queda clarament reflectit. Tota la informació queda emmagatzemada al mateix node provocant un coll d'ampolla i eliminant tota possibilitat d'execucions paral·leles.

Figura 6.34: Distribució de dades entre els nodes.

6.6 Model 5

El cinquè model equival al model 2 però amb compressió.

6.6.1 Hipòtesis

En aquest cas, a diferència del model 2 que és el que ocupa més espai, l'espai ocupat ha de baixar. Això, sumant-hi que no s'hi duen a terme operacions de join, hauria de provocar una disminució del temps de consulta.

D'altra banda, al tenir una sola taula amb famílies de columnes separades, la compressió hauria de funcionar molt bé donat que treballa a nivell de família de columna.
El guany d'espai hauria de ser considerable respecte al model 2.

6.6.2 Resultats obtinguts

A primera vista, a partir de les figures 6.35, 6.36 i 6.37, es detecta que es perd el comportament de creixement exponencial del model 2 per a un creixement clarament lineal segons la magnitud de tuples. Aquest fet el pot explicar que en el model 2 la major part de temps s'emprava llegint dades de disc, ara el volum de dades llegides de disc és molt més baix i la major part del temps es perd en CPU per descomprimir.

En les mateixes figures 6.36 i 6.37, també es pot veure com el factor de selecció no és crític per augmentar el temps de consulta com si ho era al model 2, possiblement per la mateixa raó d'intercanvi de temps de disc per CPU.

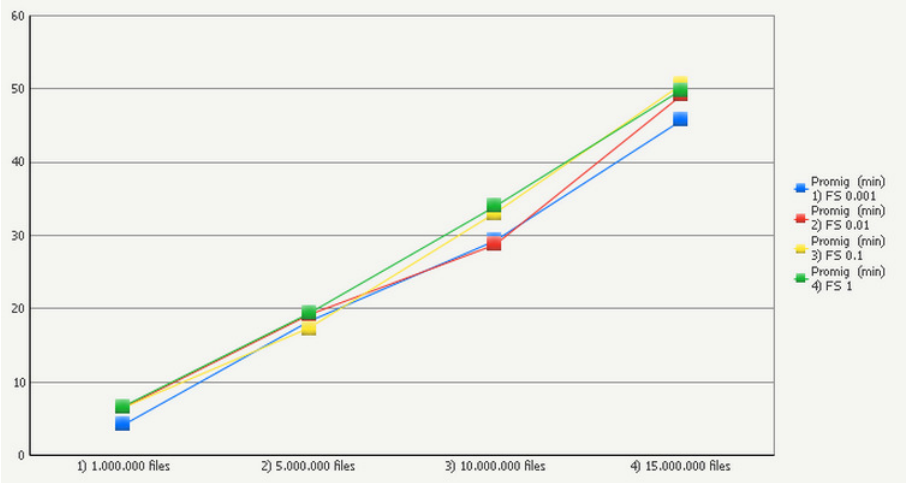


Figura 6.35: 1 dimensió i tots els factors de selecció.

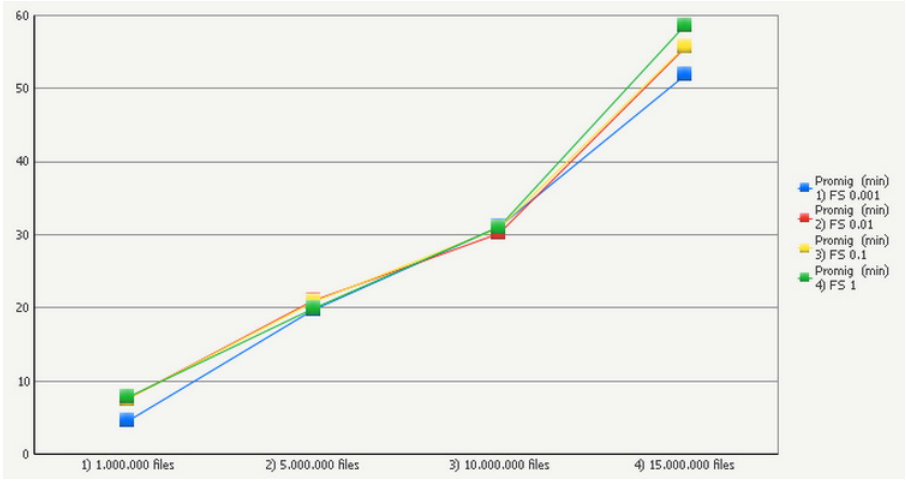


Figura 6.36: 2 dimensions i tots els factors de selecció.

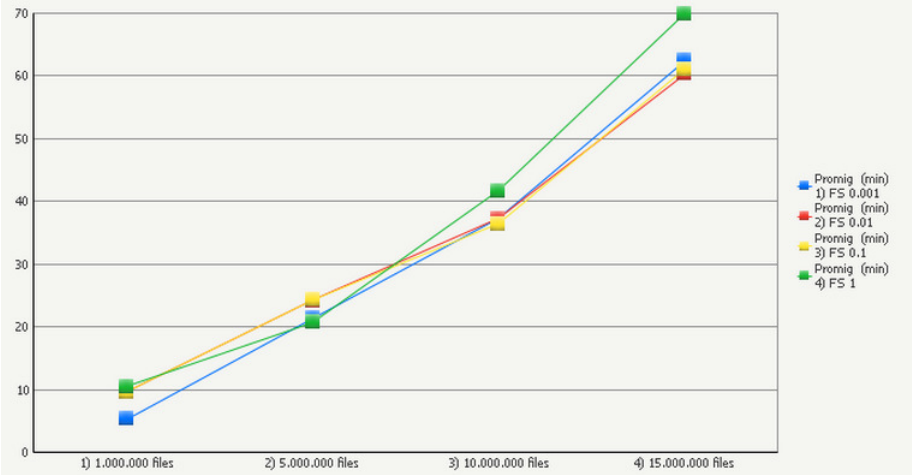


Figura 6.37: 3 dimensions i tots els factors de selecció.

D'altra banda, la variable consulta tampoc té gaire importància, ja que per a mateixes quantitats i factors de selecció els temps són gairebé iguals. Ocorre el mateix cas que en el model 2.

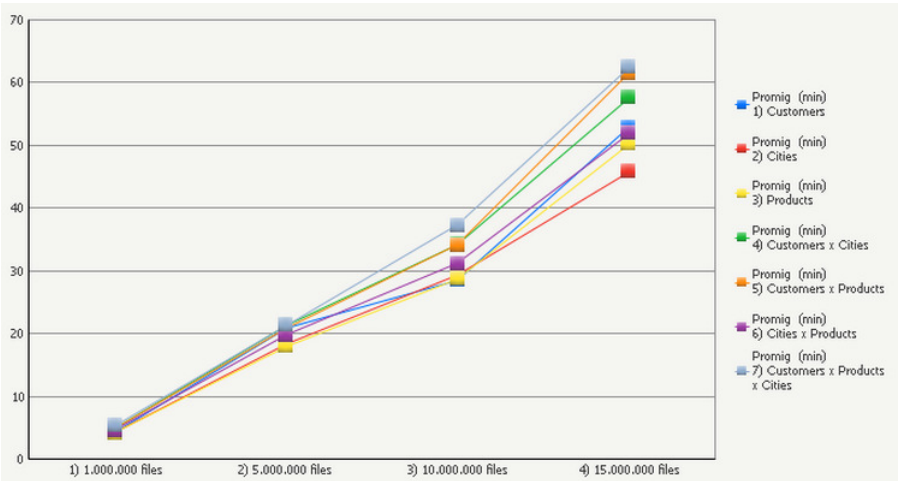


Figura 6.38: Factor de selecció 10^{-3} i totes les consultes.

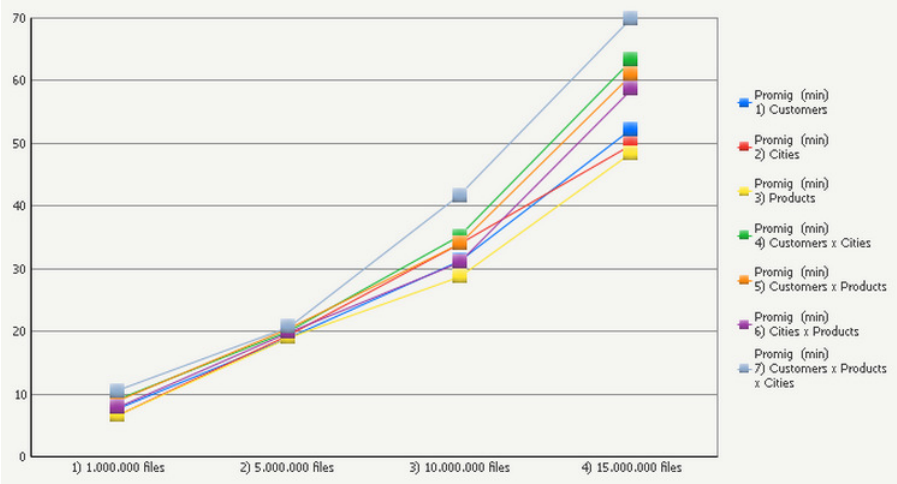


Figura 6.39: Factor de selecció 10^0 i totes les consultes.

Pel que fa a l'espai ocupat, és evident l'aplicació de compressió i el guany d'espai és molt gran, concretament un 20% respecte l'espai ocupat per el model 2.

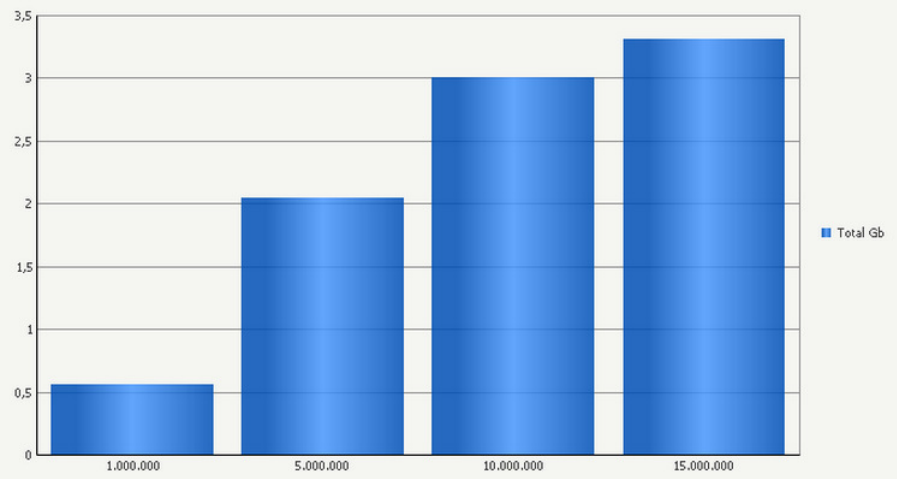


Figura 6.40: Espai ocupat per cada magnitud de tuples.

Finalment, pel que fa a la distribució de dades continua existint el problema de la mala distribució entre nodes, penalitzant clarament el paral·lelisme de les operacions.

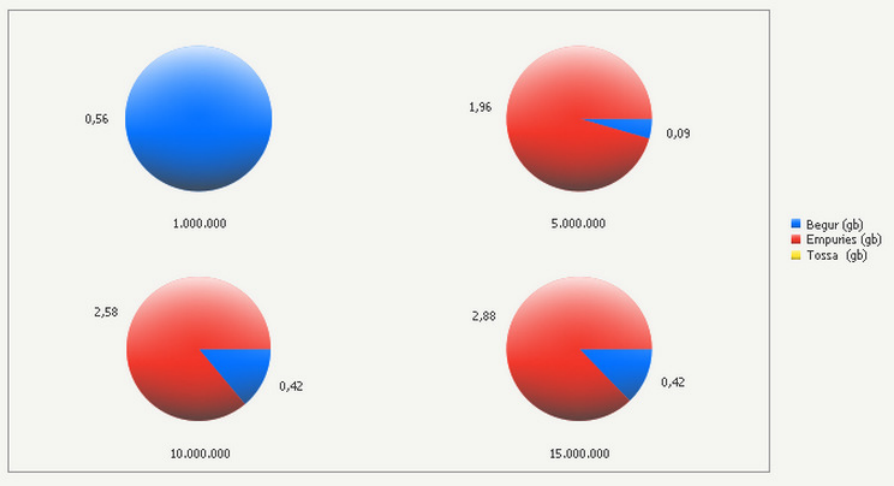


Figura 6.41: Distribució de dades entre els nodes.

6.7 Model 6

L'últim model equival al model 3 amb compressió.

6.7.1 Hipòtesis

En aquest cas, s'esperen millorar els resultats del model 3 al aplicar la compressió. D'altra banda, no s'espera millorar els resultats del model 5 ja que la compressió possiblement no funcionarà tant bé com abans.

6.7.2 Resultats obtinguts

Es pot observar, en les figures 6.42, 6.43 i 6.44, que tots els resultats es mantenen constants al llarg dels diferents factors de selecció, és a dir, no afecten al temps de consulta. D'altra banda, es detecta un creixement no lineal a partir de la magnitud de tuples.

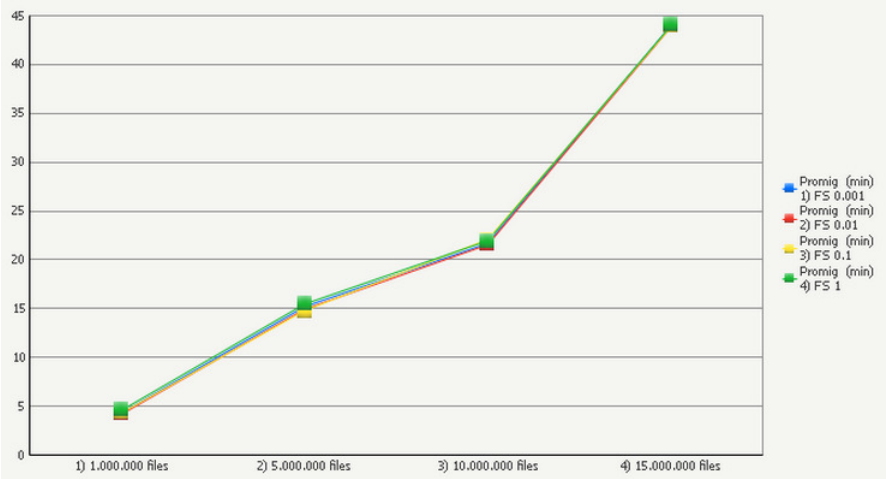


Figura 6.42: 1 dimensió i tots els factors de selecció.

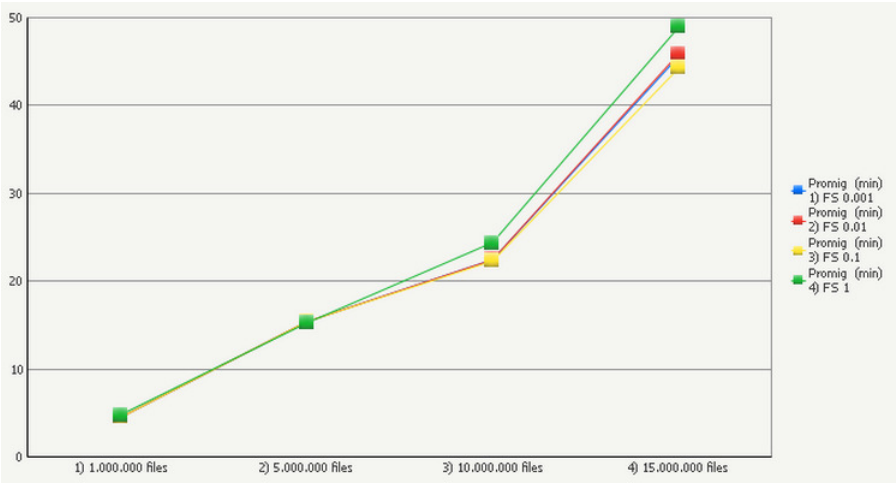


Figura 6.43: 2 dimensions i tots els factors de selecció.

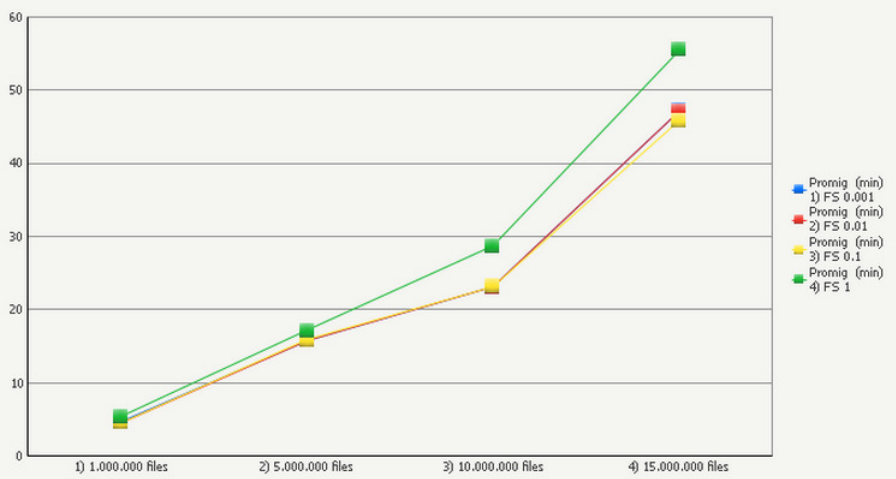


Figura 6.44: 3 dimensions i tots els factors de selecció.

Pel que fa a les consultes, sembla que tampoc afecta el tipus de consulta aplicat al temps d'execució. En aquest cas, com al model 3, s'hi ha de fer un *full table scan*.

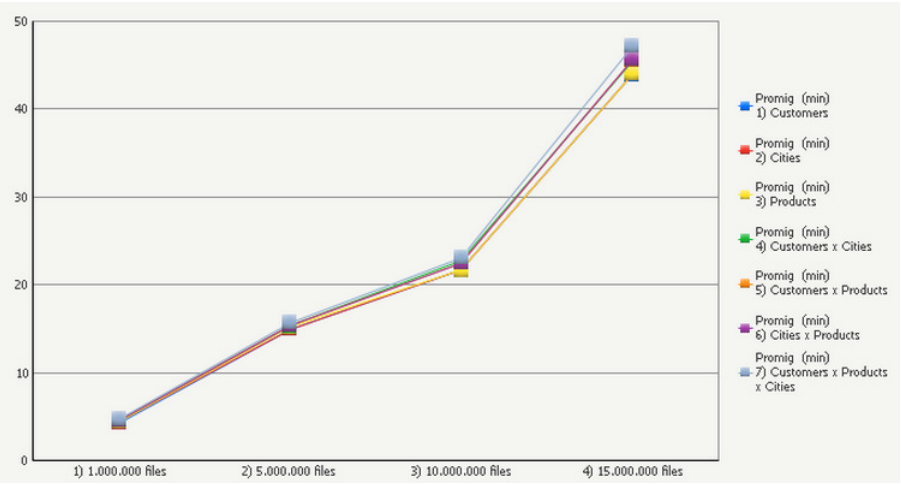


Figura 6.45: Factor de selecció 10^{-3} i totes les consultes.

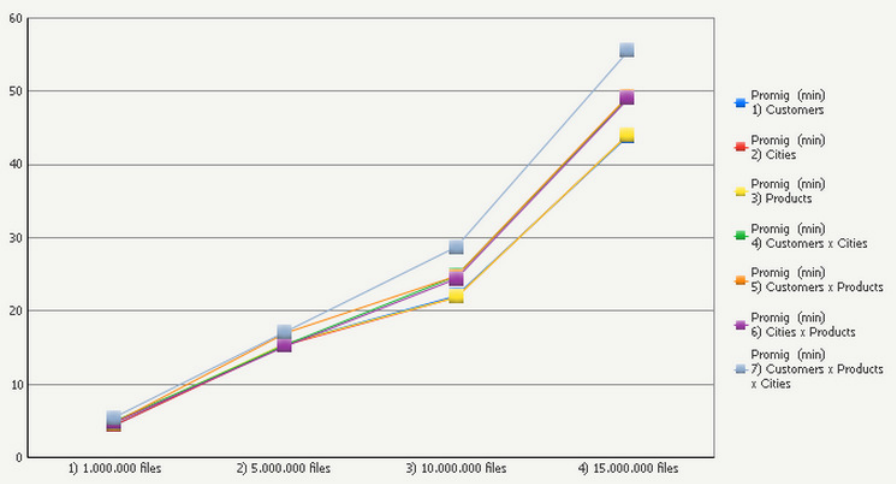


Figura 6.46: Factor de selecció 10^0 i totes les consultes.

Referent a l'espai ocupat, aquest ha millorat respecte el model 3 però no tant com amb el model 5. Clarament, té a veure en com HBase comprimeix les famílies de columnes.
Concretament, l'espai resulta en aproximadament un 17% de l'espai ocupat per el model 3.

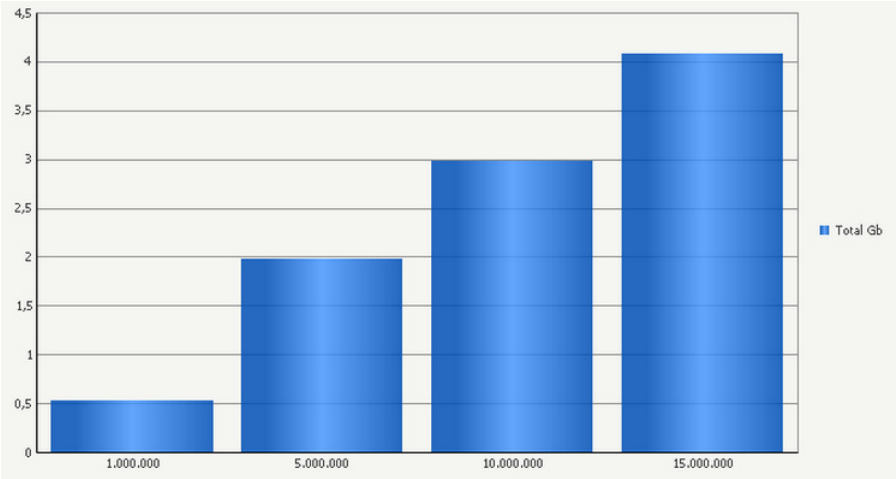


Figura 6.47: Espai ocupat per cada magnitud de tuples.

Finalment, la distribució de dades continua sent un dels principals problemes.

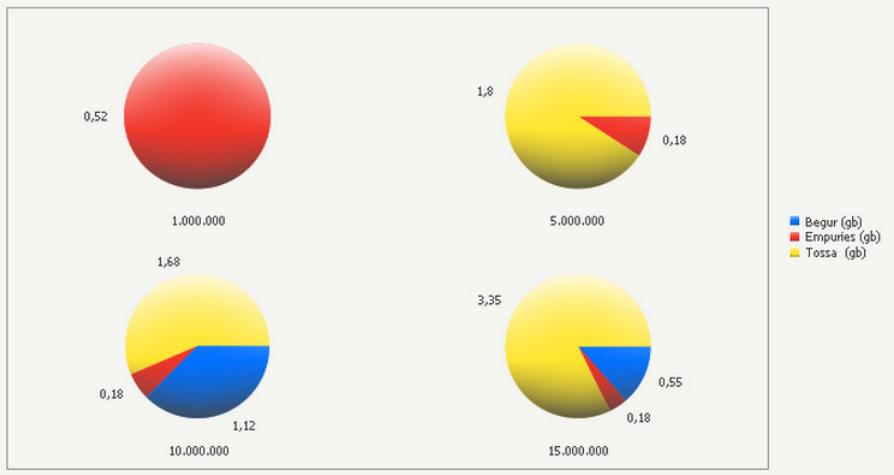


Figura 6.48: Distribució de dades entre els nodes.

6.8 Comparació global

Un cop analitzats els models per separat, val la pena fer-ne una comparació unint-los tots per tal de veure en quines condicions quin model dóna millors resultats.

Mirant els casos extrems, evitant mostrar aquells en que el model relacional falla en el qual no es pot fer comparació, veiem que per factors de selecció baixos independentment de la consulta el model relacional de MySQL dóna els millors resultats.
Donat que les magnituds de tuples provades són relativament baixes, per a considerar-se *Big Data*, MySQL té molt bons mecanismes per a executar consultes en aquestes condicions. El fet d'emmagatzemar les claus primàries de cada tupla a memòria és un punt a favor clar, ja que les joins se'n beneficien.

D'altra banda, a l'hora d'executar consultes sobre HBase es fa mitjançant tasques MapReduce. Una tasca MapReduce es compon de 15 subtasques les quals impliquen: lectures de disc, moviments de dades entre nodes, divisions i fusions entre altres. Així doncs, per a grans volums de dades tots aquests passos no són un problema, però per a volums petits clarament la consulta queda penalitzada.

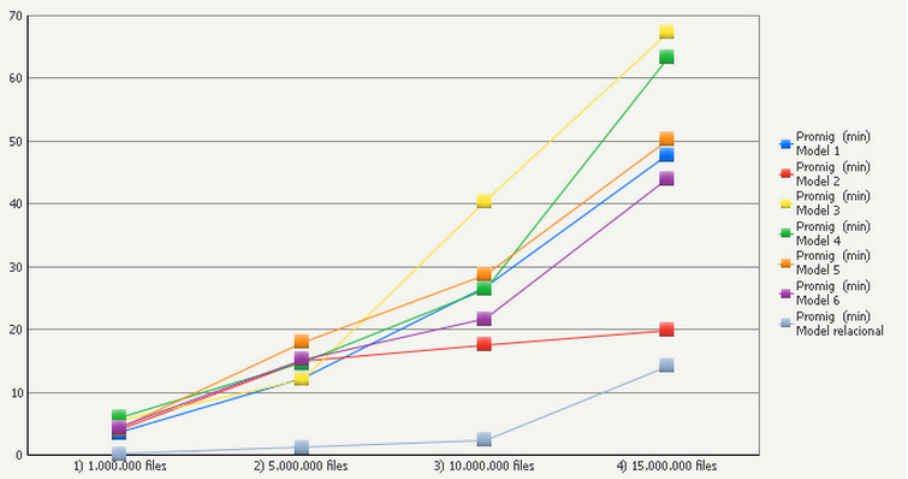


Figura 6.49: Factor de selecció 10^{-3} i consulta d'una dimensió.

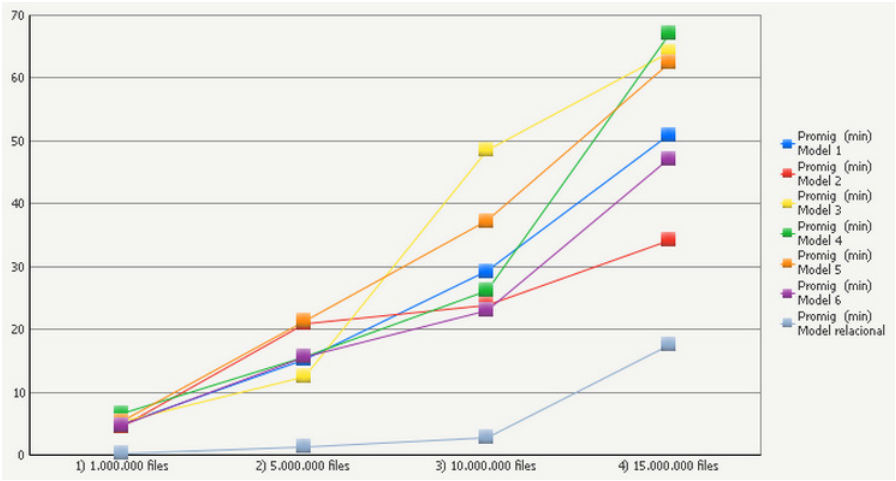


Figura 6.50: Factor de selecció 10^{-3} i consulta de tres dimensions.

Un cop s'augmenta el factor de selecció, el model relacional perd tot el bon rendiment que tenia i passa a ser un dels pitjors. En aquest cas, la solució NOSQL amb el model 2 dóna els millors resultats, excepte en el cas de la figura 6.53 on és el model 6.
La pèrdua de rendiment en el model relacional en factors de selecció alts pot tenir a veure amb la lectura de dades de disc. En el cas d'operacions que requereixen una lectura intensiva de disc, les tasques MapReduce donen millors resultats.

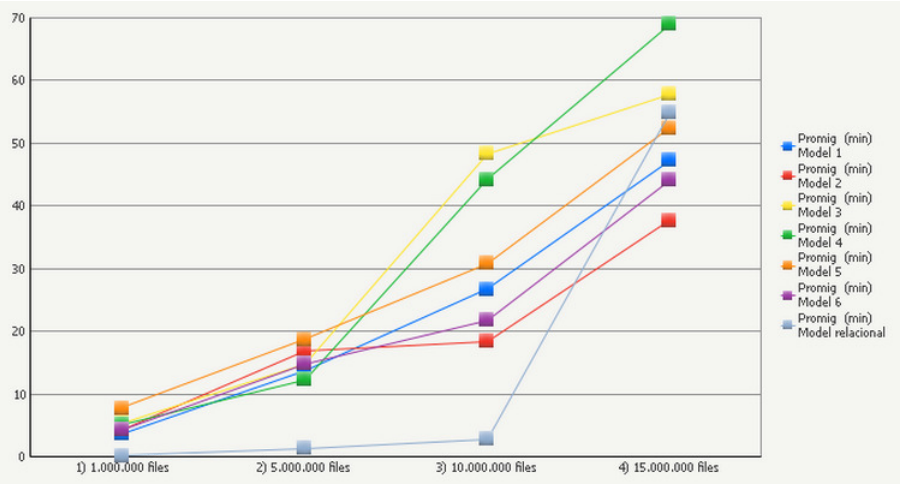


Figura 6.51: Factor de selecció 10^{-1} i consulta d'una dimensió.

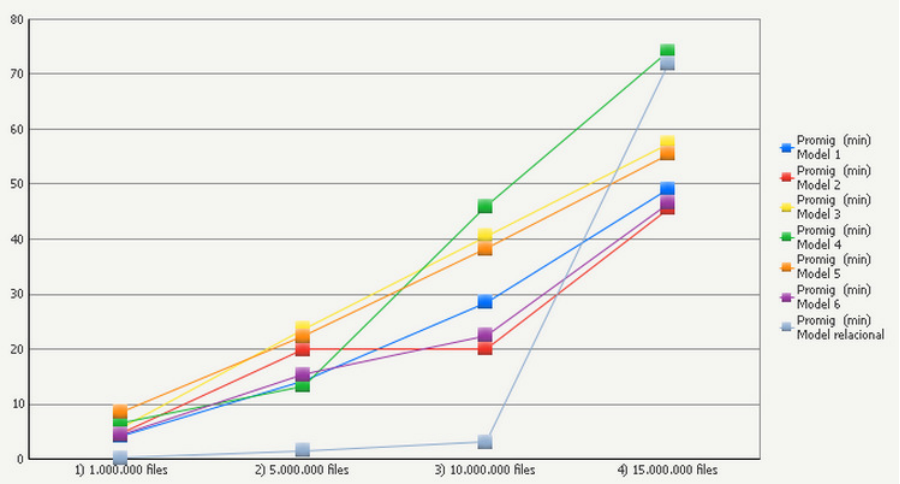


Figura 6.52: Factor de selecció 10^{-1} i consulta de dues dimensions.

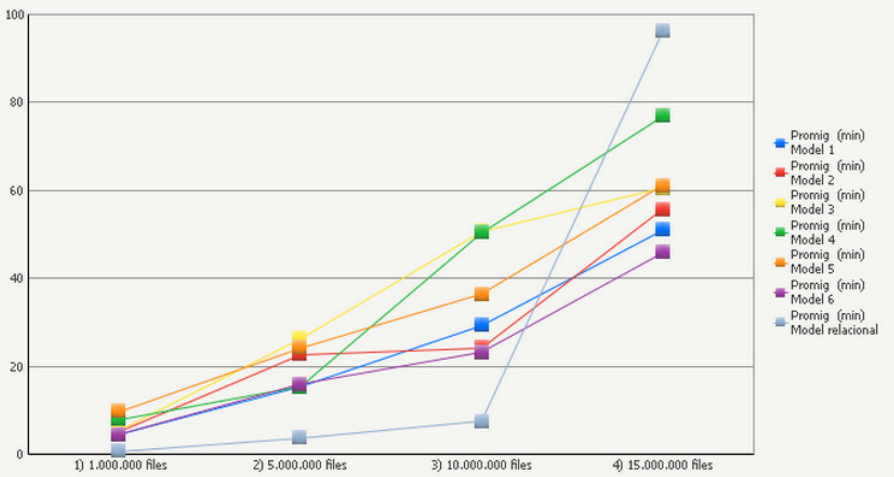


Figura 6.53: Factor de selecció 10^{-1} i consulta de tres dimensions.

Pel que fa a l'espai ocupat, el model 2 és clarament el pitjor de tots mentre que el model 1 i 4 són els millors.

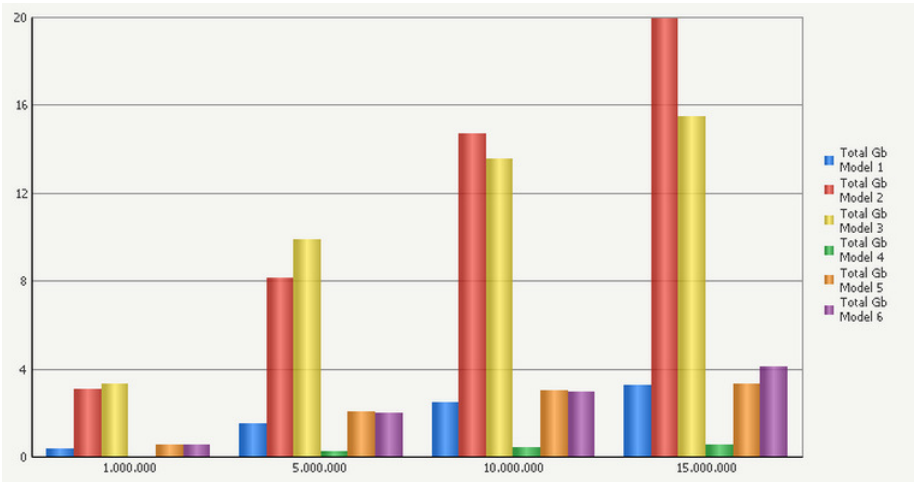


Figura 6.54: Espai ocupat per cada magnitud de tuples.

Capítol 7

Tècniques i conclusions

En aquest últim capítol es descriuran les millors conversions de model detectades a partir dels resultats obtinguts. D'altra banda, també es detallaran les conclusions i lliçons apreses durant el desenvolupament d'aquest projecte.

7.1 Tècniques de conversió

7.1.1 Denormalització

Com s'ha comprovat, denormalitzar dona millors resultats que fer operacions join tot i tenir activats els paràmetres que optimitzen aquestes operacions.

7.1.2 Comprimir no millora els temps

En molts casos, la compressió de dades en comptes de millorar els temps de consulta els penalitzava. Donat que aquests sistemes estan preparats per emmagatzemar grans quantitats de dades, no hauria de preocupar gaire el tamany ocupat dels models.
Val la pena dir que en aquest projecte s'ha utilitzat l'algorisme de compressió més lent, concretament GZ. És possible que els resultats millorin utilitzant Snappy o LZO.

D'altra banda, és lògica aquesta pèrdua de rendiment ja que per a fer les consultes primer s'han de descomprimir les dades i després fer-hi les cerques.

7.1.3 Com més espai, millor distribució

El model 2, el més gran en espai, també és el que millor distribució de dades presentava, i el que millors resultats dona. Possiblement això es degut a que es pot aprofitar millor el paral·lelisme entre nodes.
Així doncs, això ens porta a pensar que potser val la pena mantenir models grans en espai per tal d'aprofitar al màxim la capacitat d'HBase de paral·lelitzar les consultes.

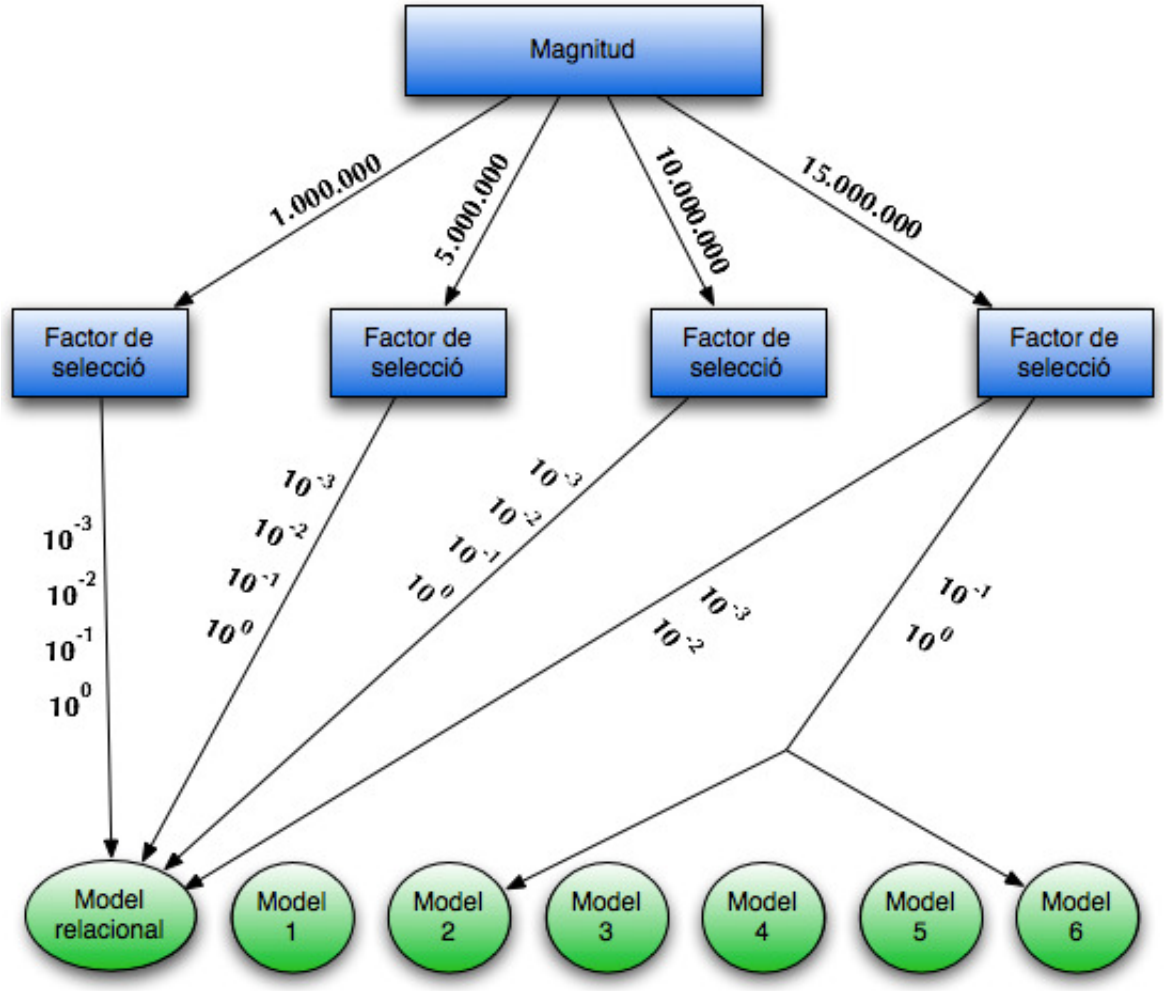
7.1.4 Hive, un possible coll d'ampolla

Ni HBase ni Hadoop estan dissenyats per executar-hi consultes tipus SQL, és per això que un conversor d'SQL a MapReduce molt possiblement farà feina extra o no de la forma més òptima.

Sempre que sigui possible, el millor és programar a mà les propies tasques MapReduce, de forma que conegut el model es poden fer molt més òptimes.

7.2 Arbre de decisió

A partir de les dades recollides de les proves i el seu anàlisi, s'ha confeccionat un arbre de decisió on es resumeixen les conclusions del projecte.
Aquest arbre no té en compte la variable consulta, o nombre de joins, per una banda perquè l'arbre tindria massa arestes i seria difícil la seva comprensió. D'altra banda, tal com s'indica al principi d'aquest document, l'objectiu és construir un *data warehouse* per a que pugui ser explotat des d'una eina de *business intelligence*. Així doncs, no convé tenir un domini tancat de consultes, val la pena buscar un esquema que permeti l'execució de qualsevol tipus de consulta.



7.3 Lliçons apreses

Un cop finalitzat el projecte i fent una mirada enrera, la primera conclusió que se'n treu és la gran dificultat que implica la instalació i correcta configuració d'aquests sistemes.
Tan MySQL com HBase són tecnologies poc madures, cosa que indica la gran quantitat d'informació que es disposa però alhora dispersa en quan a contingut. Possiblement s'haguessin pogut obtenir resultats molt millors en ambdós sistemes, però la gran quantitat de paràmetres i opcions per a configurar dificulten molt aquesta tasca.

En la meva opinió, és per aquest fet de configuració i manteniment que,

de moment, només les grans organitzacions estan utilitzant aquest nou tipus de sistemes. Són pocs els que es poden permetre tenir un equip de gent experta treballant unicament en la base de dades.

D'altra banda, una altra conclusió és que ni MySQL Cluster ni HBase són els sistemes adequats per a duur a terme consultes multidimensionals en *big data*. Tal com s'ha comentat en el capítol 1, existeixen altres tipus de sistemes que possiblement s'adapten a aquestes necessitats com les bases de dades d'emmagatzematge per columnes o les de tipus documentals.

Capítol 8

Bibliografia

8.1 Articles

[1] Erik Meijer, Gavin Bierman. *A co-Relational Model of Data for Large Shared Data Banks*. Microsoft Research.

[2] E.F. Codd. *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, Volum 13, Número 6, Juny 1970.

[3] Seth Gilbert, Naney Lynch. *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services*.

[4] Daniel J. Abadi, Samuel R. Madden, Nabil Hachem. *Column-Stores vs. Row-Stores: How Different Are They Really?*.

[5] Google Inc, *Bigtable: A Distributed Storage System for Structured Data*. Google.

[6] Matthias Mann, *History and Comparison of Relational Database Management Systems*. TechnoCircle HVB Information Services, juny 2006.

[7] Michael Stonebraker, *SQL Databases v. NOSQL Databases*. Communications of the ACM, Volum 53, Número 4, Abril 2010.

[8] Michael Stonebraker, *Stonebraker on NOSQL and Enterprises*. Communications of the ACM, Volum 53, Número 8, Agost 2011.

[9] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, Pat Helland. *The End of and Architectural Era (It's Time for a Complete Rewrite)*

[10] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, *The Google File System*. Google.

8.2 Llibres

[11] Jordi Torres. *De la informàtica en núvol al big data, visió introductòria per a joves emprenedors*. Edicions UOC, setembre 2012.

[12] Tom White, *Hadoop, The Definitive Guide*. O'Reilly, Yahoo Press, gener 2012.

[13] Lars George, *HBase, The Definitive Guide*. O'Reilly, setembre 2011.

[14] Oracle Inc, *Optimizing Performance of the MySQL Cluster Database*. MySQL White Paper, juliol 2012.

8.3 Pàgines Web

[15] Daniel Abadi, *Distinguishing Two Major Types of Column-Stores*. Disponible a: `http://dbmsmusings.blogspot.com.es/2010/03/distinguishing-two-major-types-of_29.html`

[16] Julian Browne, *Brewer's CAP Theorem*. Disponible a: `http://www.julianbrowne.com/article/viewer/brewers-cap-theorem`

[17] From Dual Inc, *MySQL Cluster Memory Sizing*. Disponible a: `http://www.fromdual.ch/mysql-cluster-memory-sizing`

[18] Oracle Inc, *MySQL Cluster Manual*. Disponible a: `https://dev.mysql.com/doc/refman/5.1/en/mysql-cluster.html`

[19] Craig Utley, *Designing the Star Schema Database*. Disponible a: `http://www.ciobriefings.com/Publications/WhitePapers/DesigningtheStarSchemaDatabase/tabid/101/Default.aspx`

[20] Apache Hive, *Apache Hive Wiki*. Disponible a: `https://cwiki.apache.org/Hive/home.html`.

[21] Facebook Engineering, *Join Optimization in Apache Hive*. Disponible a: `https://www.facebook.com/notes/facebook-engineering/join-optimization-in-apache-hive/470667928919`

Glossari

API Conjunt de funcions i procediments oferts per una llibreria i que proporciona una capa d'abstracció. 23

BLOB *Binary Large Object*: poden ser des de tipus de dades bàsics a imatge, so o vídeo. 12

CAP També conegut com a teorema de Brewer proposat per el professor Eric Brewer l'any 2000. 15

classpath És una variable d'entorn la qual indica a la JVM on buscar els paquets i classes a usar. 35

cluster Un cluster és un conjunt de servidors (nodes) que actuen com un únic sistema. 15

CRUD Acrònim de l'anglès *create*,*read*, *update* i *delete*. Això equival al cicle de vida d'un objecte: crear, llegir, actualitzar i esborrar. 24

datamart Versió reduïda d'un *data warehouse* on les dades emmagatzemades són d'un subconjunt concret d'aquest. 48

datastore Repositori de dades. 21

Derby Base de dades relacional que permet ser distribuïda amb qualsevol aplicació Java sense necessitat de configuració. S'acostuma a usar per la gestió de metadades. 31

E-R Model Entitat-Relació, proposat per Peter Chen l'any 1976 en l'article "*The Entity Relationship Model: Toward a Unified View of Data*'. 9

GFS *Google File System*: sistema de fitxers distribuït desenvolupat per Google. 23

HDFS *Hadoop Distributed File System*: el sistema de fitxers distribuït d'Apache. 20

JVM *Java Virtual Machine*, o màquina virtual de Java és un entorn tancat on s'executen els programes Java, això permet que sigui independent de l'arquitectura. 20

memòria cau Memòria instal·lada en el processador la qual és de poc tamany i alta velocitat de lectura-escriptura. 23

NOSQL *Not Only SQL*. 12

OLTP Sigles de *Online Transaction Processing*, és un tipus de sistema d'informació que administra aplicacions transaccionals, és a dir, d'entrada, recuperació i processament de transaccions. 48

REST Patró arquitectònic de programació de webservices, es caracteritza per: manca d'estat, operacions definides a priori, sintaxi universal. 23

RSA Algorisme criptografic per a xifrar i signar digitalment a partir d'una clau publica i una privada. 35

SQL *Structured Query Language*, desenvolupat a principis dels anys 70 per Donald D. Chamberlin i Raymond F. Boyce a IBM, en contínua evolució i millora per part de les organitzacions ISO i IEC. 10

SSH *Secure Shell*, protocol de comunicació en xarxa a través d'un canal xifrat i segur. 35

Ubuntu server Sistema operatiu de codi obert i distribució de GNU/Linux. 33

Apèndix A

FAQ: Instalació i configuració entorns

En aquest capítol es farà un llistat dels problemes que han sorgit durant la instal·lació de tot el software necessari per a desenvolupar el projecte. També s'hi adjuntarà la solució aplicada.
És important dir que aquestes solucions s'han aplicat en les condicions descrites en aquest projecte, és possible que en d'altres especificacions o procediments d'instal·lació no siguin vàlides.

A.1 Cluster

A.1.1 Ampliar espai en màquines virtuals

P: Tot i ampliar el disc disponible a la màquina virtual, l'espai disponible no creix a Ubuntu.

R: Aquest problema és degut a que el nou espai es dona a una partició sense assignar. Per tal d'assignar-lo a la partició primària, mitjançant l'eina `gparted` seguim el següent procés:

1. Arrenquem la màquina virtual amb la ISO de `gparted`.
2. Ampliar la partició `extended` al màxim a la dreta.
3. Ampliar la partició `linux-swap` al màxim a la dreta.
4. Reduir per l'esquerra la partició `linux-swap` al tamany original.
5. Reduir per l'esquerra la partició `extended` al mínim, és a dir al tamany de `linux-swap`.

6. Ampliar la partició `ext4` per la dreta el màxim que poguem.

A.2 MySQL Cluster

A.2.1 Arrancada `mysqld`

P: A l'arrencar `mysqld` surt el següent error:

```
[ERROR] Can't start server : Bind on unix socket: No such
        file or directory
[ERROR] Do you already have another mysqld server
        running on socket: /var/run/mysqld/mysqld.sock ?
```

R: Crear directori i assignar permisos, és necessari fer-ho cada vegada que es reinicia el node.

```
sudo mkdir /var/run/mysqld
sudo chmod 777 /var/run/mysqld/
```

A.2.2 The table is full

P: La càrrega de dades falla prematurament amb l'error:

```
ERROR 1114: The table 'SALES' is full
```

R: Aquest error es dona perquè la memòria de dades o la d'índexos està plena. Algunes possibles optimitzacions són:

- Arrancar `mysqld` amb l'opció `-big-tables`.
- Crear la taula amb més tuples amb el paràmetre `MAX_ROWS`.
- Modificar la configuració al fitxer `config.ini` per tal que donar-li més memòria a índexs o dades segons calgui.

A.2.3 Creació de tablespaces

P: Tot i arrancar els nodes de dades amb l'opció `-initial`, al crear de nou els *tablespaces* dona error.

R: Al iniciar els nodes amb el paràmetre `-initial` no s'esborren els *tablespaces*, però tampoc no es permet reutilitzar-los amb les noves dades. En el cas dels tablespaces creats en aquest projecte, els fitxers implicats són:

`undo_1.log`, `undo_2.log`, `data_1.dat`, `data_2.dat`. S'han de borrar de cada node de dades abans de crear-los de nou, per tal de buscar-los usarem la següent comanda:

```
sudo find / -iname undo_1.log &&
        sudo find / -iname undo_2.log &&
        sudo find / -iname data_1.dat &&
        sudo find / -iname data_2.dat
```

A.2.4 Desactivar estadístiques d'índexs a mysqld

P: A l'arrencar el servidor `mysqld`, surt cada segon el següent *warning* i el rendiment del *cluster* baixa.

```
[Warning] create index stats tables failed: error 4720 line 403
```

R: Arrancar `mysqld` amb l'opció:

```
Arrancar mysqld amb --ndb-index-stat-enable=false
```

A.2.5 Esquema bloquejat

P: Un cop arrancat `mysqld` al fer qualsevol acció des del client dona el següent error:

```
[Error] could not acquire global schema lock (4009) cluster failure.
```

R: Possiblement el *host* de la connexió a `mysqld` està mal indicat, en cas que no serà necessari reinstalar.

A.2.6 Solapament entre consultes

P: Al llençar múltiples consultes amb poc interval de temps entre elles, la consulta falla amb l'error següent:

```
Lock wait timeout exceeded; try restarting transaction
```

R: Quan s'intenta executar una nova consulta, en algun dels nodes de dades encara queda traça de l'anterior, això provoca un solapament de transaccions i error. Algunes possibles optimitzacions són:

- Abans de llançar la consulta executar `COMMIT` i després `ROLLBACK`.

- A la configuració de `config.ini`, afegir el paràmetre `TransactionDeadlockDetectionTimeout` amb un valor alt (entre 10 i 15 segons).
- Desactivar transaccions mitjançant: `set ndb_use_transactions=off`.

A.3 HBase

A.3.1 Blocs perduts

P: A la web d'administració de Hadoop surt el següent missatge de *warning*:

```
WARNING : There are about 5 missing blocks.
        Please check the log or run fsck.
```

R: Amb el DFS parat, executar les següents comandes:

```
hadoop fsck /
hadoop fsck -delete /
hadoop fsck -move /
```

A.3.2 Namespace IDs incompatibles

P: A l'engegar el DFS surt el següent error:

```
ERROR org.apache.hadoop.dfs.DataNode:
        java.io.IOException: Incompatible namespaceIDs
```

R: La solució més simple i dràstica consisteix en formatejar el DFS:

1. Aturar el *cluster*.
2. Eliminar el directori de dades del node problemàtic. Aquest directori està especificat per la propietat `dfs.data.dir` en el fitxer `conf/hdfs-size.xml`.
3. Formatejar el *cluster*.
4. Reiniciar el cluster.

A.4 Hive

A.4.1 Connexió des de Java

P: Al connectar des del client Java, l'aplicació queda bloquejada al fer:

```
1 DriverManager.getConnection("jdbc:hive://master:10000/default", "","");
```

R: Tancar la consola de Hive. Aquesta bloqueja la base de dades Derby, la qual no permet més d'una connexió concurrent.

A.4.2 Metadades bloquejades

P: Al llençar qualsevol consulta a Hive obtenim el següent error:

```
ERROR XSDB6: Another instance of Derby may
        have already booted the database
        /var/lib/hive/metastore/metastore_db.
```

R: Borrem la base de dades de metadades i creem les taules de nou.

```
sudo rm /var/lib/hive/metastore/metastore_db/ -r
```

Apèndix B

Codis de les proves

Tan en les proves a MySQL com les d'HBase, les consultes són llegides d'un fitxer com el següent:

```
1  %1 DIMENSIO
2  % CUSTOMERS
3  SELECT CUSTOMERS.name, AVG(SALES.price) FROM CUSTOMERS STRAIGHT_JOIN
       SALES ON SALES.id_customer = CUSTOMERS.id WHERE CUSTOMERS.key%100=1;
4  % CITIES
5  SELECT CITIES.name, AVG(SALES.price) FROM CITIES STRAIGHT_JOIN SALES ON
       SALES.id_city = CITIES.id WHERE CITIES.key%100=1;
6  % PRODUCTS
7  SELECT PRODUCTS.code, AVG(SALES.price) FROM PRODUCTS STRAIGHT_JOIN SALES
       ON SALES.id_city = PRODUCTS.id WHERE PRODUCTS.key%150=1;
8  %
9  %2 DIMENSIONS
10 % CUSTOMERS x CITIES
11 SELECT STRAIGHT_JOIN CUSTOMERS.name, CITIES.name, AVG(SALES.price) FROM
       CUSTOMERS,CITIES,SALES WHERE SALES.id_customer = CUSTOMERS.id AND
       SALES.id_city = CITIES.id AND CUSTOMERS.key%100=1 AND CITIES.key
       %100=1;
12 % CUSTOMERS x PRODUCTS
13 SELECT STRAIGHT_JOIN CUSTOMERS.name, PRODUCTS.code, AVG(SALES.price) FROM
        CUSTOMERS,PRODUCTS,SALES WHERE SALES.id_customer = CUSTOMERS.id AND
       SALES.id_product = PRODUCTS.id AND CUSTOMERS.key%100=1 AND PRODUCTS.
       key%150=1;
14 % CITIES x PRODUCTS
15 SELECT STRAIGHT_JOIN CITIES.name, PRODUCTS.code, AVG(SALES.price) FROM
       CITIES,PRODUCTS,SALES WHERE SALES.id_city = CITIES.id AND SALES.
       id_product = PRODUCTS.id AND PRODUCTS.key%150=1 AND CITIES.key%100=1;
16 %
```

```
%3 DIMENSIONS
% CUSTOMERS x PRODUCTS x CITIES
SELECT STRAIGHT_JOIN CUSTOMERS.name, PRODUCTS.code, CITIES.name, AVG(
    SALES.price) FROM CUSTOMERS,PRODUCTS,CITIES,SALES WHERE SALES.
    id_customer = CUSTOMERS.id AND SALES.id_product = PRODUCTS.id AND
    SALES.id_city = CITIES.id AND CUSTOMERS.key%100=1 AND PRODUCTS.key
    %150=1 AND CITIES.key%100=1;
```

Els símbols % indiquen comentaris i aquests han de ser escrits per pantalla, d'aquesta forma serà més fàcil la recollida de resultats de les proves.

B.1 Proves a MySQL

La classe `TestsParser` s'encarrega de llegir les consultes i separar els comentaris del codi SQL.

```
public static ArrayList<Pair> parse(String filePath) {
      ArrayList<Pair> consultes = new ArrayList<Pair>();
      try {
            FileInputStream fstream = new FileInputStream(filePath);
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new InputStreamReader(
                in));
            String strLine;
            while ((strLine = br.readLine()) != null) {
                  Pair p = new Pair();
                  if (strLine.charAt(0) == '%') {
                        p.msg = strLine.substring(1);
                        p.isSQL=false;
                  } else {
                        p.msg = strLine;
                        p.isSQL = true;
                  }
                  consultes.add(p);
            }
            in.close();
      } catch (Exception e){
            e.printStackTrace();
      }
      return consultes;
}
```

A l'hora d'executar les proves, i degut a que era inviable fer-ho d'una sola tirada, s'ha definit una matriu de tres dimensions que indica quines

consultes executar per cada magnitud i factor de selecció. D'altra banda, també es manté una llista amb les magnituds a sumar per tal de fer una càrrega incremental i no carregar de nou.
Ambdós estructures estan definides a la classe `utils`. Aquesta classe també inclou un mètode per executar les consultes i retornar el temps emprat.

```
1  public static int magnituds[] = {1000000,4000000,5000000,5000000};
2
3  public static boolean aplica[][][] = {
4        { /* 1.000.000 */
5              {true,true,true,true,true,true,true}, /* fs3 */
6              {true,true,true,true,true,true,true}, /* fs2 */
7              {true,true,true,true,true,true,true}, /* fs1 */
8              {true,true,true,true,true,true,true}, /* fs0 */
9        },
10       { /* 5.000.000 */
11             {true,true,true,true,true,true,true}, /* fs3 */
12             {true,true,true,true,true,true,true}, /* fs2 */
13             {true,true,true,true,true,true,true}, /* fs1 */
14             {true,true,true,true,true,true,true}, /* fs0 */
15       },
16       { /* 10.000.000 */
17             {true,true,true,true,true,true,true}, /* fs3 */
18             {true,true,true,true,true,true,true}, /* fs2 */
19             {true,true,true,true,true,true,true}, /* fs1 */
20             {true,true,true,true,true,true,true}, /* fs0 */
21       },
22       { /* 15.000.000 */
23             {true,true,true,true,true,true,true}, /* fs3 */
24             {true,true,true,true,true,true,true}, /* fs2 */
25             {true,true,true,true,true,true,true}, /* fs1 */
26             {true,true,true,true,true,true,true}, /* fs0 */
27       }
28 };
29
30 public static long getSQLTime(String SQL) throws SQLException {
31       Statement st = conn.createStatement();
32       long t1=0,t2=0;
33       try {
34             st.execute("set ndb_join_pushdown=on");
35             st.execute("set ndb_use_transactions=off");
36             st.execute("COMMIT");
37             t1 = System.currentTimeMillis();
38             st.execute(SQL);
```

```
39              t2 = System.currentTimeMillis();
40              st.execute("ROLLBACK");
41              st.execute("RESET QUERY CACHE");
42        } catch (SQLException exc) {
43              exc.printStackTrace();
44        }
45        return t2-t1;
46  }
```

La classe `TestsRunner`, és l'encarregada d'utilitzar les estructures i mètodes de la classe `utils`. Concretament executa totes les proves per a cada magnitud i factor de selecció.

```
1   public static void start(ArrayList<Pair> consultes) throws Exception {
2         boolean primer=true;
3         int contador_cons=0;
4         for (int i = 0; i < consultes.size(); ++i) {
5               if (!consultes.get(i).isSQL) utils.getInstance().escriu(2,
                    consultes.get(i).msg);
6               else {
7                     if (utils.aplica[utils.magnitud_actual][utils.FSactual][
                          contador_cons]) {
8                           if (primer) {
9                                 utils.carrega();
10                                primer=false;
11                          }
12                          long p1 = utils.getInstance().getSQLTime(consultes
                                .get(i).msg);
13
14                          utils.getInstance().escriu(5, "1.- "+p1+" ms ("+
                                String.format("%d min, %d sec",
15                                    TimeUnit.MILLISECONDS.toMinutes(p1),
16                                    TimeUnit.MILLISECONDS.toSeconds(p1) -
17                                    TimeUnit.MINUTES.toSeconds(TimeUnit.
                                          MILLISECONDS.toMinutes(p1))
18                                    )+")");
19
20                          long p2 = utils.getInstance().getSQLTime(consultes
                                .get(i).msg);
21                          utils.getInstance().escriu(5, "2.- "+p2+" ms ("+
                                String.format("%d min, %d sec",
22                                    TimeUnit.MILLISECONDS.toMinutes(p2),
23                                    TimeUnit.MILLISECONDS.toSeconds(p2) -
```

```
24                                          TimeUnit.MINUTES.toSeconds(TimeUnit
                                                .MILLISECONDS.toMinutes(p2))
25                                      )+")");
26                          long p3 = utils.getInstance().getSQLTime(consultes
                                .get(i).msg);
27                          utils.getInstance().escriu(5, "3.- "+p3+" ms ("+
                                String.format("%d min, %d sec",
28                                      TimeUnit.MILLISECONDS.toMinutes(p3),
29                                      TimeUnit.MILLISECONDS.toSeconds(p3) -
30                                            TimeUnit.MINUTES.toSeconds(TimeUnit
                                                .MILLISECONDS.toMinutes(p3))
31                                      )+")");
32                          utils.getInstance().escriu(5, "Promig.- "+((p1+p2+
                                p3)/3));
33                  }
34                  ++contador_cons;
35              }
36      }
37  }
```

B.2 Proves a HBase

Les proves a HBase s'han realitzat mitjançant un *shellscript*, aquest llegeix el fitxer de consultes corresponent a cada factor de selecció i llança el client de Hive amb la consulta com a paràmetre.

Cada model requereix un codi diferent, aquest ha de cridar al codi Java carregador (`main.jar`) i al fitxer de consultes per el model concret. A continuació s'adjunta el codi del model 1.

```
1  #!/bin/bash
2
3  getCurrentTimeInMili() {
4   date +'%H 3600 * %m 60 * + %S + 1000 * %N 1000000 / + p' | dc
5  }
6
7  echo "Inici proves HBase"
8
9  consultes=("/home/serginadalfrancesch/Dropbox/PFC/Proves/HBase/ProvesFS3.
       sql" "/home/serginadalfrancesch/Dropbox/PFC/Proves/HBase/ProvesFS2.sql
       " "/home/serginadalfrancesch/Dropbox/PFC/Proves/HBase/ProvesFS1.sql" "
       /home/serginadalfrancesch/Dropbox/PFC/Proves/HBase/ProvesFS0.sql")
```

```
10 magnituds=(15000000 5000000 5000000 5000000)
11 rk=(0 15000001 20000001 25000001)
12 rkindex=0
13 for i in "${magnituds[@]}"
14 do
15       echo "Carregant "$i" files comenant per rk = "${rk[rkindex]}
16       java -jar ./main.jar $i ${rk[rkindex]}
17       for j in "${consultes[@]}"
18       do
19             while read consulta
20             do
21                   if [ ${consulta:0:1} == "%" ];then
22                         echo "  "$consulta
23                   else
24                         echo $consulta > /home/serginadalfrancesch/
                               consultes/sql.sql
25                         t1=‘date +%s%N | cut -b1-13‘
26                         hive --auxpath /usr/local/hive/lib/hive-hbase-
                               handler-0.10.0.jar,/usr/local/hive/lib/hbase
                               -0.92.0.jar,/usr/local/hive/lib/zookeeper
                               -3.4.3.jar,/usr/local/hive/lib/guava-r09.jar,/
                               usr/local/hbase/lib/protobuf-java-2.4.0a.jar -
                               hiveconf hbase.master=master:60000 -f /home/
                               serginadalfrancesch/consultes/sql.sql -S >
                               sort.txt
27                         t2=‘date +%s%N | cut -b1-13‘
28                         diff=$((t2-t1))
29                         echo "    Prova 1: "$diff" ms"
30                         t1=‘date +%s%N | cut -b1-13‘
31                         hive --auxpath /usr/local/hive/lib/hive-hbase-
                               handler-0.10.0.jar,/usr/local/hive/lib/hbase
                               -0.92.0.jar,/usr/local/hive/lib/zookeeper
                               -3.4.3.jar,/usr/local/hive/lib/guava-r09.jar,/
                               usr/local/hbase/lib/protobuf-java-2.4.0a.jar -
                               hiveconf hbase.master=master:60000 -f /home/
                               serginadalfrancesch/consultes/sql.sql -S >
                               sort.txt
32                         t2=‘date +%s%N | cut -b1-13‘
33                         diff=$((t2-t1))
34                         echo "    Prova 2: "$diff" ms"
35                         t1=‘date +%s%N | cut -b1-13‘
36                         hive --auxpath /usr/local/hive/lib/hive-hbase-
                               handler-0.10.0.jar,/usr/local/hive/lib/hbase
```

```
                                -0.92.0.jar,/usr/local/hive/lib/zookeeper
                                -3.4.3.jar,/usr/local/hive/lib/guava-r09.jar,/
                                usr/local/hbase/lib/protobuf-java-2.4.0a.jar -
                                hiveconf hbase.master=master:60000 -f /home/
                                serginadalfrancesch/consultes/sql.sql -S >
                                sort.txt
37                          t2=‘date +%s%N | cut -b1-13‘
38                          diff=$((t2-t1))
39                          echo "    Prova 3: "$diff" ms"
40                      fi
41                done < $j
42          done
43          rkindex=‘expr $rkindex + 1‘
44  done
```