Captura, preprocessament i emmagatzematge de dades massives

PID_00250687

Francesc Julbe
Jordi Conesa Caralt
Jordi Casas Roma
M.ª Elena Rodríguez González

Temps mínim de dedicació recomanat: 5 hores

Universitat Oberta de Catalunya

Índex

Introducció 5

Objectius 6

1. Captura de dades massives 7
1.1. Dades estàtiques 7
1.1.1. Apache Sqoop 8
1.2. Dades en *streaming* 9
1.2.1. *Streams* de dades 10
1.2.2. Enviament i recepció de dades en *stream* 12
1.2.3. Introducció a Apache Kafka 15

2. Preprocessament de dades massives 20
2.1. Tasques de preprocessament de dades 20
2.1.1. Neteja de dades 20
2.1.2. Transformació de dades 21
2.2. Arquitectures de preprocessament 22

3. Emmagatzematge de dades massives 24

4. Sistemes de fitxers distribuïts 25
4.1. Què és HDFS? 26
4.2. Arquitectura 27
4.2.1. *NameNode* i *DataNode* 27
4.2.2. Espai de noms (*namespace*) 29
4.3. Fiabilitat 32
4.3.1. HDFS *heartbeats* 33
4.3.2. Reajustament de blocs de dades 34
4.3.3. Permisos d'usuaris, fitxers i directoris 35
4.4. Interfície HDFS 35
4.4.1. Línia de comandaments 35
4.4.2. Projectes de l'ecosistema 37

5. Bases de dades NoSQL 41
5.1. Què és NoSQL? 42
5.1.1. Característiques de les bases de dades NoSQL 43
5.2. Models de dades NoSQL 44
5.2.1. Models de dades NoSQL d'agregació 46
5.2.2. Models de dades NoSQL en graf 49
5.2.3. Exemple pràctic 50

5.3. Avantatges i inconvenients de les bases de dades NoSQL 57

Resum 60

Glossari 61

Bibliografia 62

Introducció

En aquest mòdul tractarem les particularitats dels processos de captura, preprocessament i emmagatzematge de dades en entorns de dades massives (*big data*).

La primera part d'aquest mòdul agrupa els processos de captura i preprocessament de dades, mentre que a la segona part ens endinsarem en l'emmagatzematge de dades massives.

Iniciarem la primera part amb una anàlisi del procés de captura de dades, fent especial èmfasi en la captura de dades dinàmiques o en temps real (*streaming*). Veurem què són les dades en *streaming* i els components que intervenen en la seva creació, la seva distribució i el seu consum. Posteriorment, veurem més detalladament com es pot realitzar la distribució de dades per facilitar-ne l'escalabilitat i garantir-ne l'alta disponibilitat.

A continuació, veurem les tasques associades al preprocessament de dades i les arquitectures bàsiques que donen suport a aquestes tasques. Igual que al punt anterior, prestarem especial atenció a les dades en *streaming*, que per la seva casuística requereixen solucions més complexes i apartades dels processos analítics «tradicionals».

La segona part del mòdul comprèn dos grans blocs: d'una banda, l'emmagatzematge mitjançant sistemes de fitxers distribuïts, de l'altra, l'emmagatzematge utilitzant bases de dades NoSQL.

Pel que fa als sistemes de fitxers distribuïts, veurem la seva arquitectura i les seves funcionalitats bàsiques. Tot i que existeixen múltiples sistemes de fitxers distribuïts, en aquest material ens centrarem en el sistema de fitxers distribuïts de Hadoop (*Hadoop distributed file system*, HDFS).

Finalitzarem el segon bloc del mòdul introduint les bases de dades NoSQL. Concretament, veurem els diferents models existents i presentarem els principals avantatges i inconvenients de cadascun d'aquests models.

Objectius

Als materials didàctics d'aquest mòdul trobarem les eines indispensables per a assimilar els objectius següents:

1. Conèixer els processos de captura de dades massives, a més d'algunes de les eines més representatives de l'entorn de dades massives.
2. Comprendre les tasques associades al preprocessament de dades, a més de les principals arquitectures associades.
3. Descobrir els principals sistemes d'emmagatzematge en entorns de dades massives.
4. Conèixer les principals característiques d'un sistema de fitxers distribuït, a més dels mecanismes que permeten la fiabilitat i escalabilitat de les dades.
5. Saber el funcionament bàsic del sistema de fitxers distribuït HDFS, incloent-hi els comandaments bàsics.
6. Aprendre quins són els principals models de bases de dades NoSQL existents i els seus principals avantatges i inconvenients.
7. Ser capaç d'escollir un bon sistema d'emmagatzematge massiu a partir de les definicions i els requeriments de les dades.

1. Captura de dades massives

El primer pas de qualsevol procés d'anàlisi de dades consisteix a capturar-les, és a dir, hem de ser capaços d'obtenir les dades de les fonts que les produeixen o emmagatzemen per, posteriorment, poder emmagatzemar-les i analitzar-les.

En aquest sentit, el procés de captura de dades es converteix en un element clau en el procés d'anàlisi de dades, siguin massives o no. No serà possible, en cap cas, analitzar dades si no hem estat capaços de capturar-les quan estaven disponibles.

Existeixen dos grans blocs quan parlem de captura de dades, segons la naturalesa de la seva producció:

- **Dades estàtiques**: són dades que ja estan emmagatzemades a algun lloc, ja sigui en format de fitxers (per exemple, fitxers de text, JSON, XML, etc.) o en bases de dades.

- **Dades dinàmiques o en *streaming***: són dades que es produeixen de forma contínua, i que han de ser capturades durant un llindar limitat de temps, ja que no estaran disponibles passat un cert període de temps.

Dades estructurades

Les dades estructurades són les que segueixen un patró igual per a tots els elements que, a més, es coneix *a priori*. Per exemple, les dades d'un full de càlcul presenten els mateixos atributs per a cada fila.

Dades semiestructurades

Les dades semiestructurades són un tipus de dades que no contenen una estructura fixa predefinida *a priori*, però conté etiquetes o altres marcadors per a separar els elements semàntics i fer complir jerarquies de registres i camps de les dades. Per exemple, els documents JSON o HTML.

Dades no estructurades

Les dades no estructurades són aquelles que no segueixen cap tipus de patró conegut *a priori*. Per exemple, dos documents de text o imatges.

1.1. Dades estàtiques

Les dades estàtiques estan emmagatzemades perdurablement, és a dir, a llarg termini. Per tant, la captura d'aquestes dades és similar al procés de captura que s'ha fet en els processos de mineria «tradicionals».

L'objectiu principal és accedir a aquestes dades i portar-les als sistemes d'emmagatzematge del nostre entorn analític, que, com veurem més endavant, poden ser sistemes de fitxers distribuïts o bases de dades NoSQL, en el cas de dades massives.

Principalment, les dades estàtiques poden presentar-se en el seu origen de dues maneres:

- dades contingudes en fitxers no estructurats (per exemple, fitxers de text), semiestructurats (com, per exemple, fitxers XML) o estructurats (com, per exemple, fitxers separats per coma o fulls de càlcul);

- dades contingudes en bases de dades, bé de tipus relacional (com, per exemple, MySQL o PostgreSQL) o bé de tipus NoSQL (Cassandra o Neo4J, entre molts d'altres).

A partir d'una connexió de dades amb la font indicada, sigui un fitxer o una base de dades, es procedirà a fer una lectura de les dades de manera seqüencial, integrant les dades processades al nostre magatzem analític per analitzar-les posteriorment.

Hi ha múltiples eines que faciliten aquest tipus de tasques, algunes d'específiques i altres que engloben tot el procés d'extracció, transformació i càrrega de dades (*extract, transform, load*, ETL). A més, hi ha eines específiques per a tractar amb dades massives, mentre que n'existeixen d'altres que, tot i que poden gestionar volums relativament grans de dades, no estan dissenyades específicament per a treballar en entorns de dades massives.

Un exemple d'eina completa d'ETL pot ser el paquet ofimàtic (*suite*) Data Integration* de Pentaho, que permet fer les connexions a diferents fonts de dades i crear el *pipeline* de transformació de les dades, en cas que sigui necessari.

*** http://bit.ly/Zg49Di**

D'altra banda, podem trobar eines específiques per a entorns de dades massives com, per exemple, Apache Sqoop, que està especialitzada a transferir dades massives des d'orígens estructurats cap als sistemes d'emmagatzematge d'Apache Hadoop.

1.1.1. Apache Sqoop

Apache Sqoop** (SQL-to-Hadoop) és una eina d'importació i exportació de bases de dades relacionals. Sqoop està dissenyat per a la transferència eficient de dades entre una base de dades relacional, com MySQL o Oracle, i un magatzem de dades Hadoop, inclosos HDFS, Hive i HBase, i viceversa. Automatitza la major part del procés de transferència de dades, llegint la informació de l'esquema directament des del sistema gestor de la base de dades. Després, Sqoop fa servir MapReduce per a importar i exportar les dades cap a Hadoop i des d'ell.

**** http://sqoop.apache.org/**

Sqoop proporciona flexibilitat per mantenir les dades en el seu estat de producció mentre es copien a Hadoop, de manera que estiguin disponibles per a fer-ne una anàlisi posteriorment, sense modificar la base de dades de producció.

El funcionament de Sqoop és simple, només hem d'indicar-li l'origen de les dades i la destinació dins del sistema d'emmagatzematge de Hadoop per procedir a la importació de dades en l'entorn analític.

Exemple: importació de dades de MySQL a HDFS

Als apartats posteriors d'aquest mòdul veurem els detalls d'implementació i funcionament del sistema de fitxers distribuït HDFS, però podem obviar-ne els detalls per presentar un breu exemple d'importació de dades mitjançant l'eina Sqoop.

Suposem que disposem d'una base de dades MySQL, que està treballant en el port 3306 amb una base de dades anomenada *exemple1*, que conté la taula «sampledata» amb la informació que ens interessa transferir a Hadoop per a fer-ne una anàlisi posteriorment.

El fragment de codi següent mostra l'ordre que executa Sqoop, indicant la cadena de connexió a la base de dades MySQL, que és l'origen de dades. El paràmetre opcional `-m 1` indica que aquest procés ha d'utilitzar una única tasca de Map.

```
/srv/sqoop$ sqoop import --connect jdbc:mysql://localhost:3306/exemple1
--username root --table sampledata -m 1

17/09/15 22:47:35 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
17/09/15 22:47:35 INFO manager.MySQLManager: Preparing to use a MySQL
streaming resultset.
17/09/15 22:47:35 INFO tool.CodeGenTool: Beginning code generation
17/09/15 22:47:36 INFO manager.SqlManager: Executing SQL statement:
SELECT t.* FROM `average_price_by_state` AS t LIMIT 1

(output truncated)

17/09/15 22:47:53 INFO mapreduce.ImportJobBase: Transferred 200.4287 KB
in 15.3718 seconds (13.0387 KB/sec)
17/09/15 22:47:53 INFO mapreduce.ImportJobBase: Retrieved 3272 records.
```

En aquest exemple, hem especificat que la importació hauria de fer servir una única tasca de Map, ja que la nostra taula no conté una clau principal, que és necessària per a dividir i fusionar múltiples tasques de Map. Atès que hem fet aquesta especificació, hauríem d'esperar un únic arxiu al sistema HDFS, tal com podem veure:

```
/srv/sqoop$ hadoop fs -head average_price_by_state/part-m-00000

2012,AK,Total Electric Industry,17.88,14.93,16.82,0.00,null,16.33
2012,AL,Total Electric Industry,11.40,10.63,6.22,0.00,null,9.18
2012,AR,Total Electric Industry,9.30,7.71,5.77,11.23,null,7.62
2012,AZ,Total Electric Industry,11.29,9.53,6.53,0.00,null,9.81
2012,CA,Total Electric Industry,15.34,13.41,10.49,7.17,null,13.53
2012,CO,Total Electric Industry,11.46,9.39,6.95,9.69,null,9.39
2012,CT,Total Electric Industry,17.34,14.65,12.67,9.69,null,15.54
2012,DC,Total Electric Industry,12.28,12.02,5.46,9.01,null,11.85
2012,DE,Total Electric Industry,13.58,10.13,8.36,0.00,null,11.06
2012,FL,Total Electric Industry,11.42,9.66,8.04,8.45,null,10.44
```

La instrucció anterior mostra les primeres files del fitxer *part-m-00000*, en què podem veure les deu primeres files de les dades importades de la base de dades MySQL.

El procés d'importació a HBase o Hive és similar a l'exemple que acabem de veure.

1.2. Dades en *streaming*

Les dades en *streaming* són dades que es generen (i publiquen) contínuament. Normalment aquest tipus de dades les generen múltiples agents de manera concurrent i solen ser d'una grandària petita (de l'ordre de kilobytes).

Exemples de dades en *streaming* són les dades enviades per sensors que es troben al carrer, dades borsàries, dades cardiovasculars enviades per un rellotge

intel·ligent, dades sobre les activitats que duu a terme un usuari d'una aplicació mòbil o dades sobre xarxes socials (clics, m'agrada, compartir informació, etc.).

L'existència d'aquest tipus de dades no és nova. De fet, les tecnologies de bases de dades han anat proposant solucions basades en dades en *streaming* des de fa dècades. No obstant això, la irrupció de noves tecnologies que permeten generar i enviar més dades i amb més freqüència (internet de les coses i ciutats intel·ligents, per exemple), les millores en els sistemes distribuïts, la irrupció dels microserveis, l'evolució dels sistemes analítics i l'adopció d'una cultura analítica massivament han provocat un ressorgiment de l'interès en els fluxos de dades (*streams*).

Aquest material presentarà una breu introducció sobre el tema, es proposarà una definició de *stream* de dades, es mostrarà com poden enviar-se les dades des dels punts de creació als punts de consum i s'estudiarà com poden preparar-se les dades per a donar respostes a les necessitats analítiques, introduint algunes de les tecnologies d'enviament i de processament més rellevants.

1.2.1. *Streams* de dades

Un *stream* de dades es pot veure com un conjunt de parelles ordenades (D, t), en què D és un conjunt de dades d'interès i t és un nombre real positiu. Les diferents parelles estan ordenades en funció del valor de t.

La variable t de cada parella indica l'ordre en què es van produir o es van enviar les dades. Així doncs, existirà una funció d'ordre sobre t que indiqui quines dades es van generar o es van enviar en primer lloc.

Seguint aquesta representació, una seqüència de temperatures enviada per un sensor situat al carrer podrà veure's de la manera següent:

```
(<25 graus, 70% humitat>, 1483309623)
(<25 graus, 70% humitat>, 1483309624)
(<26 graus, 60% humitat>, 1483309625)
(<15 graus, 90% humitat>, 1483309626)
```

en què *<graus, humitat>* indica les dades d'interès (els nivells de temperatura i humitat) i el nombre indica la data en què es van recollir aquestes mesures utilitzant temps Unix.

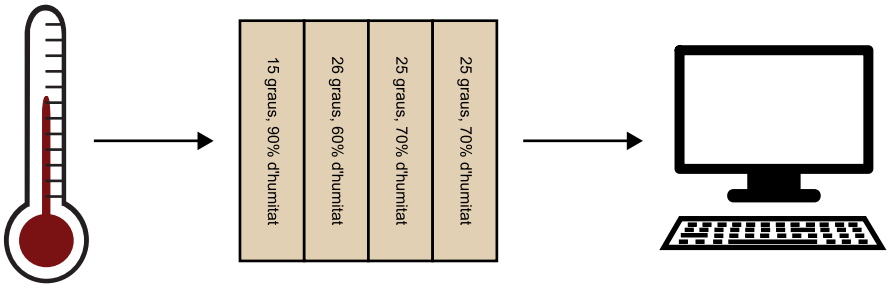
En funció de si la *t* ve determinada per un moment en el temps (un sensor mesura la temperatura cada mil·lisegon, per exemple) o per l'ordre causal de determinats esdeveniments (un internauta ha eliminat un producte de la seva cistella de la compra) estarem parlant de *streams* originats pel **temps** o per **esdeveniments**.

Temps Unix

Per a representar una data en temps Unix s'utilitza un nombre natural. El valor d'aquest nombre indica els segons transcurreguts des de la mitjanit de l'1 de gener de 1970.

Normalment, quan es parla de *streams* s'utilitza una visualització orientada a cues, en què es representen les dades de manera consecutiva en una llista en funció de quan es van originar (el seu valor *t*). De vegades, s'obvia el valor de la *t* en la representació i es mostra només l'ordre causal de les dades. Per tant, l'*stream* anterior podria representar-se com es mostra a la figura 1.

Figura 1. Representació de l'*stream* de dades generat per un sensor de temperatura



A l'exemple de la figura 1 s'ha assumit que el sistema tindrà un consumidor, és a dir, un servei que consultarà les dades que es vagin publicant a l'*stream* per a realitzar alguna tasca, com, per exemple, monitorar els valors rebuts i fer saltar una alarma quan la temperatura o la humitat estiguin fora d'uns rangs de control prefixats.

Utilitzar sistemes d'enviament i processament en *streaming* és beneficiós en escenaris on es generen noves dades contínuament. Són sistemes d'interès general que es troben presents en tot tipus de negoci. Des d'un punt de vista analític, les possibilitats del processament de *streams* són moltes i molt variades. Les més simples són la recol·lecció i el processament de *logs* (registres), la generació d'informes o quadres de comandament i la creació d'alarmes en temps real. Les més complexes inclouen anàlisis de dades més complexes, com l'ús d'algorismes d'aprenentatge automàtic o de processament d'esdeveniments.

Des d'un punt de vista funcional, podríem dividir els sistemes en *streaming* en tres components:

1) Productors: són els sistemes encarregats de generar i enviar les dades contínuament. El sistema productor de dades pot ser molt variat, des de complexos sistemes que generen informació borsària o meteorològica fins a senzills sensors distribuïts pel territori que mesuren i envien informació sobre la temperatura o la pol·lució. És comú que existeixin diferents productors en un mateix sistema. A l'exemple de la figura 1 només hi ha un productor: el sensor que

s'encarrega de mesurar la temperatura i la humitat de l'ambient i d'enviar els valors del mesurament.

2) Consumidors: són els sistemes encarregats de recollir i processar les dades produïdes. El nombre de consumidors associat a un *stream* de dades pot ser superior a un. Aquests sistemes poden tenir múltiples objectius; en el context analític, per exemple, aquests sistemes poden realitzar-se des de simples agregacions de dades fins a processos complexos per a transformar i enriquir les dades amb l'objectiu de poblar altres sistemes i crear altres *streams* de dades. En el cas de l'exemple tenim tan sols un consumidor: un computador que recull les temperatures i humitats i comprova si estan dins d'un rang acceptable.

3) Missatgeria: és el sistema que s'encarrega de transmetre les dades, des del seu productor fins als seus potencials consumidors. Aquest sistema pot ser molt simple i utilitzar connexions directes per a enviar les dades entre productors i consumidors. No obstant això, es tendeixen a utilitzar sistemes de missatgeria més complexos que garanteixin una alta tolerància a fallades i més escalabilitat.

1.2.2. Enviament i recepció de dades en *stream*

Aquest subapartat tracta sobre els sistemes de missatgeria. Aquests sistemes permeten l'intercanvi de dades entre diferents ordinadors.

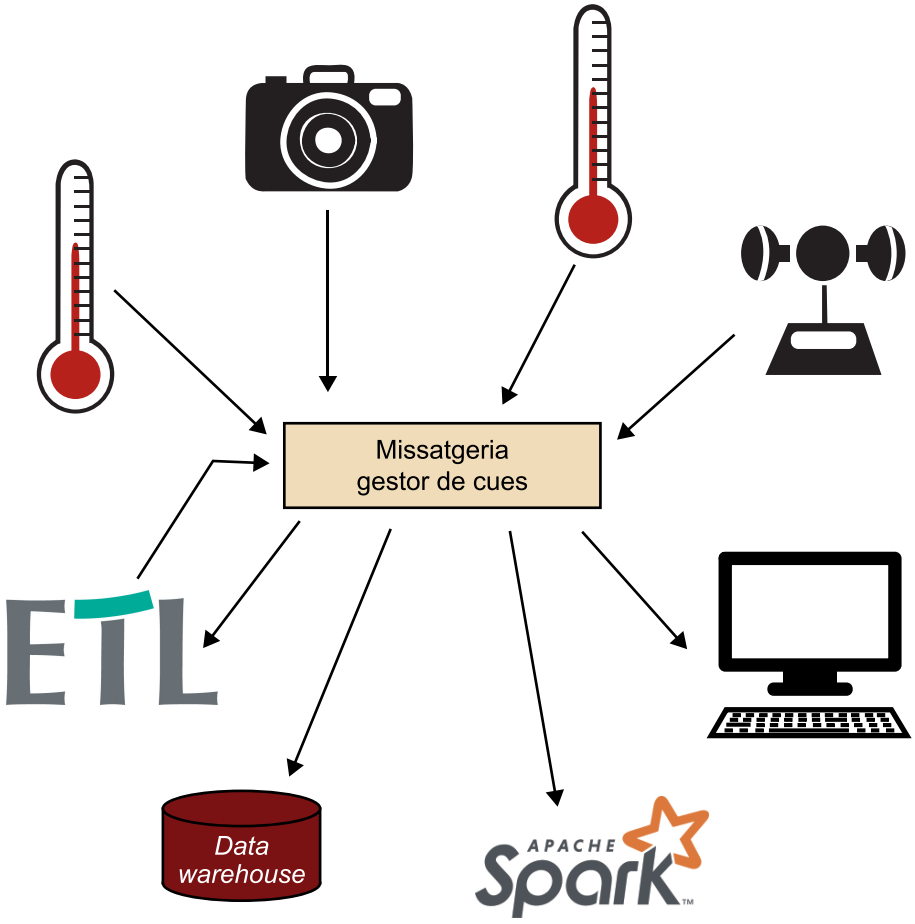
Quan volem compartir dades entre diferents dispositius podem optar per diferents alternatives. Potser l'alternativa més senzilla seria utilitzar una arquitectura client-servidor. En un entorn client-servidor les dades es generen en el client (el productor) i s'envien directament al servidor (el consumidor). Recordem que en el cas general aquest sistema podrà tenir múltiples productors i consumidors, per tant, hauríem d'implementar un sistema client-servidor amb tants clients com productors hi hagi i tants servidors com consumidors hi hagi.

Aquest sistema és molt simple, però presenta diversos problemes. El més rellevant és la seva difícil evolució, ja que en afegir (o eliminar) nous productors (o consumidors), haurem de modificar tots els programes servidor (o client) per establir les connexions directes entre els nous elements de la xarxa. Un altre problema d'aquesta aproximació és la seva baixa tolerància a fallades (en casos de caigudes de xarxa o de servidors els missatges enviats podrien perdre's irremeiablement) i la seva difícil escalabilitat (atès que no hi ha una infraestructura pròpia per a la gestió de l'enviament i la recepció dels missatges; per millorar el rendiment dels enviaments hauríem de millorar la velocitat de la xarxa, ja que afegir nous nodes al sistema no hi provocaria cap millora).

Una alternativa a l'arquitectura client-servidor seria utilitzar sistemes de missatgeria per a realitzar l'enviament de les dades en *stream*. Aquests sistemes afegeixen una estructura de xarxa intermèdia per desacoblar els sistemes cre-

adors dels sistemes consumidors. Aquests sistemes també proporcionen més disponibilitat, escalabilitat i tolerància a fallades. A la figura 2 podem veure un exemple de sistema de missatgeria.

Figura 2. Exemple d'utilització d'un sistema de missatgeria



El sistema de la figura 2 es compon de quatre productors i quatre consumidors. Els productors són de diferent tipus: dos sensors de temperatura, una càmera fotogràfica i un anemòmetre (sensor de vent). La informació recol·lectada per aquests sistemes s'envia al sistema de gestió de cues. Aquest és l'encarregat de garantir que la informació enviada (els missatges) arribi als seus potencials consumidors. Aquests consumidors podrien ser, per exemple, un magatzem de dades (*data warehouse*) que emmagatzemi la informació recol·lectada fins al moment, un quadre de comandament, un sistema Spark per a fer analítiques en temps real i un procés d'extracció, transformació i càrrega que agregui i refini les dades. Aquest sistema ETL podria generar noves dades, com per exemple estimar la temperatura de determinats punts geogràfics en funció de les temperatures rebudes i afegir aquesta nova informació en un nou *stream* de dades. En aquest cas, el procés ETL es comportaria també com a productor per un altre *stream* de dades.

Els sistemes de missatgeria segueixen bàsicament dos models: el de **cues** i el de **publicació/subscripció**. En tots dos sistemes els missatges es distribueixen temàticament en diferents *streams* (ja sigui en cues o en temes). Una cua pot

tenir assignats un conjunt de consumidors vàlids; cada missatge de la cua s'envia a un d'ells. En el cas del sistema de publicació/subscripció, els missatges d'un tema es difonen a tots els consumidors subscrits al tema.

El sistema intermedi de missatgeria és el que es denomina *message oriented middleware* (o MOM) i pot ser programari o maquinari. Aquest sistema assumeix la responsabilitat de transferir els missatges entre aplicacions, interconnectant diferents elements d'un sistema distribuït i facilitant la comunicació entre ells. Aquest programari intermediari (*middleware*) haurà de gestionar els problemes d'interoperabilitat de dades que pot haver-hi, a causa de l'heterogeneïtat entre els diferents productors (diferents protocols de comunicació o tipus de dades incompatibles) i l'heterogeneïtat entre productors i consumidors. També haurà de proporcionar mesures per a promoure una alta disponibilitat en el procés d'intercanvi d'informació.

Els avantatges d'utilitzar aquest tipus de sistemes són els següents:

- **Permet l'enviament de dades de manera asíncrona**: les dades rebudes es guarden en una cua i es van processant a mesura que el consumidor els sol·licita.

- **Permet desacoblar els diferents components del sistema distribuït**: es podran afegir o eliminar creadors o consumidors d'informació sense necessitat de modificar o ajustar els nodes del sistema distribuït.

- **Permet oferir una alta disponibilitat**: facilita la implementació de mesures per a garantir que els missatges no es perdin davant caigudes de xarxa o de nodes, com podrien ser la replicació de les dades enviades o la definició de protocols per a garantir que les dades d'un *stream* s'eliminin solament quan hagin estat consumides per un nombre mínim de consumidors.

- **Facilita l'escalabilitat**: facilita l'escalabilitat horitzontal i permet, per exemple, distribuir un *stream* en diferents nodes de la xarxa.

Existeixen diferents protocols de missatgeria que poden utilitzar-se per a realitzar l'enviament de missatges entre productors i consumidors. Els més populars en el context de les dades massives i de l'internet de les coses són potser AMQP* (*advance message queuing protocol*), MQTT** (*message queue telemetry transport*) i STOMP*** (*simple streaming text oriented messaging protocol*).

* https://www.amqp.org
** http://mqtt.org
*** https://stomp.github.io

Aquests protocols són adoptats per diferents sistemes de missatgeria que implementen una capa de programari intermediari per a l'enviament i la recepció de missatges. Alguns dels més populars són: Apache Kafka, Apache ActiveMQ* i RabbitMQ.**

* https://www.amqp.org
** https://www.rabbitmq.com

A continuació, amb l'objectiu de mostrar el funcionament d'aquest tipus de sistemes, introduirem Apache Kafka.

1.2.3. Introducció a Apache Kafka

Apache Kafka*** és un sistema de missatgeria distribuït i replicat de codi obert. Té el seu origen en LinkedIn, on es va crear per gestionar els fluxos d'informació entre les seves diferents aplicacions.

*** https://kafka.apache.org

Kafka s'executa en un clúster, que pot estar compost per un o més nodes. A Kafka, els *streams* poden fragmentar-se i distribuir-se entre diferents nodes, fet que permet l'escalabilitat horitzontal. També permet la replicació dels missatges enviats per proveir d'una alta disponibilitat en cas de fallades a la xarxa o en els nodes del clúster. De fet, Kafka proporciona les garanties següents:

1) Els missatges enviats per un productor s'afegiran a l'*stream* en l'ordre d'enviament.

2) Els consumidors poden obtenir els missatges en l'ordre en què van ser enviats.

3) Si definim N com el nombre de rèpliques d'un determinat *stream*, el sistema garantirà la disponibilitat de les dades sempre que el nombre de nodes que fallin no superi el valor de $N - 1$; és a dir, sempre que quedi almenys una rèplica en funcionament.

Les dades enviades a Kafka s'emmagatzemen en registres. Aquests registres s'assignen a diferents *streams* de dades en funció del seu contingut (categoria o propòsit). Cadascun d'aquests *streams* de dades o categories es denomina *topic*. Els registres es componen d'una tripleta de la forma <*clau, valor, timestamp*>.

Una altra característica interessant de Kafka és que, a diferència d'altres sistemes de missatgeria, el consum d'una dada per part d'un productor no elimina la dada de l'*stream*. Les dades enviades a Kafka tenen data de caducitat (es pot configurar el temps que han d'estar disponibles) i es mantenen al sistema durant aquest temps. Quan s'arriba a la data límit, les dades s'eliminen. Això permet millorar-ne la disponibilitat, ja que els consumidors poden llegir dades en qualsevol moment, fins i tot quan altres consumidors ja hagin consumit la dada. D'altra banda, traspassa la decisió sobre quines dades llegeix cada consumidor. El consumidor que consumeix dades d'un *stream* ha de saber quines dades ha llegit, quines dades li interessa llegir i en quin ordre (a Kafka un consumidor pot consumir les dades en un ordre diferent de l'ordre d'arribada).

Arquitectura d'Apache Kafka

Els components principals que podem trobar-nos a Kafka són els següents, tal com es mostra a la figura 3:

- **Productors**: són els sistemes que envien dades a Kafka. Aquestes dades s'envien en forma de registre i podran ser publicades en un o més *topics*.

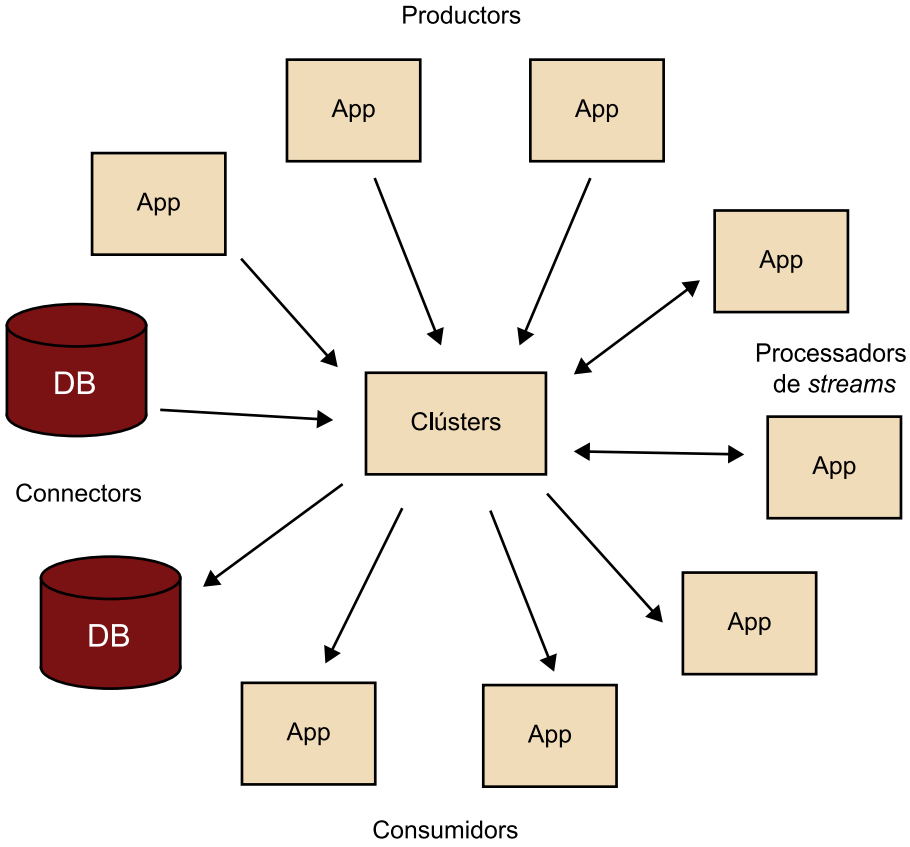
- **Clústers**: el sistema encarregat de la recepció, fragmentació, distribució, replicació i enviament de missatges. Pot estar compost de diferents nodes, cadascun d'ells denominat *broker*.

- **Consumidors**: són els sistemes de destinació de l'*stream* de dades. Es poden subscriure a un o més *topics* per a obtenir-ne els missatges.

- **Connectors**: permeten crear consumidors (o productors) reutilitzables que consumeixin (o publiquin) dades en un o més *topics* a partir de bases de dades o aplicacions existents.

- **Processadors de *streams***: permeten crear aplicacions per a preprocessar i enriquir les dades d'un *stream* de dades. Aquests sistemes actuen com a consumidors d'un o més *topics*, realitzen operacions sobre les dades proporcionades i escriuen les dades resultants en un o més *topics*.

Figura 3. Principals components de Kafka



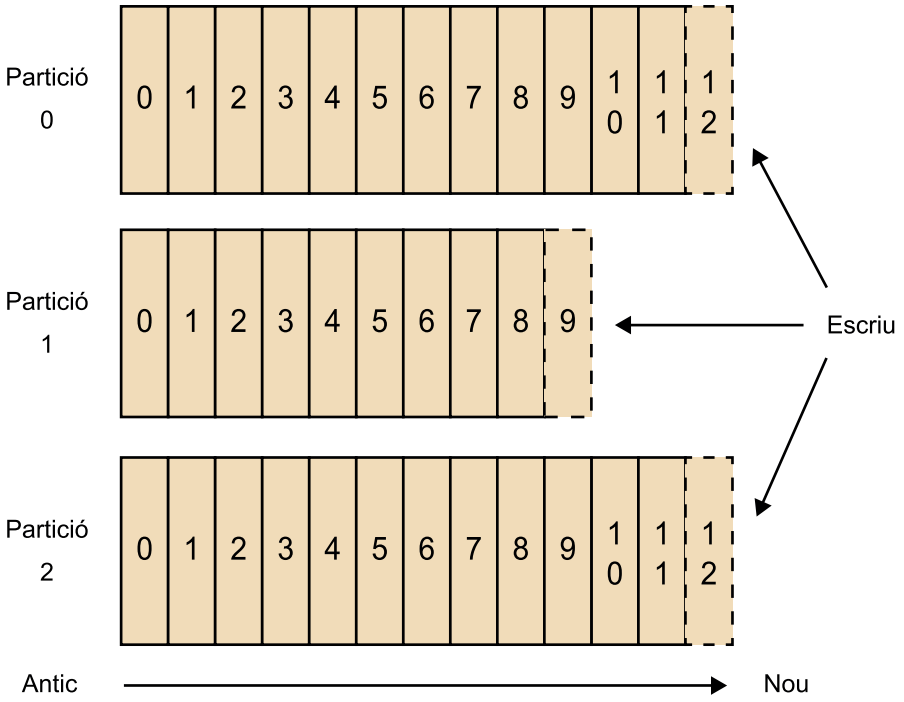
Representació interna dels *streams* a Kafka

Tal com s'ha comentat, les dades es distribueixen en *topics* segons la seva categoria. Per tant, quan un productor envia dades a Kafka, haurà d'indicar a quin

topic corresponen. D'altra banda, els *topics* poden estar lligats a zero, a un o a més consumidors.

Com es pot veure a la figura 4, els *topics* es distribueixen en diferents particions disjuntes. Aquesta distribució es realitza per facilitar l'escalabilitat horitzontal, la distribució de càrrega i la paralel·lització.

Figura 4. Distribució d'un *topic* en tres particions



A la figura 4 es pot veure un exemple en què s'ha dividit un *topic* en tres particions. Cada partició és una seqüència de registres immutable (una vegada afegit un registre a la seqüència no pot modificar-se). En cada partició els registres van afegint-se de manera seqüencial. Cada registre té un nombre anomenat *offset* (la seva posició dins de la seqüència) que l'identifica unívocament dins de la partició.

Per defecte, escollir en quina partició ha d'emmagatzemar-se cada registre és tasca dels productors. Per tant, aquests seran els responsables de garantir una correcta distribució de dades en les diferents particions d'un *topic*.

Quan un consumidor vol consultar les dades d'un *stream* ha de saber el *topic* i la partició en els quals s'han assignat les dades d'interès. Per indicar quines dades es volen consumir, el consumidor ha d'indicar també l'*offset* dels registres que desitja obtenir. Tal com s'ha comentat anteriorment, el consum d'un registre no eliminarà el registre del *topic*.

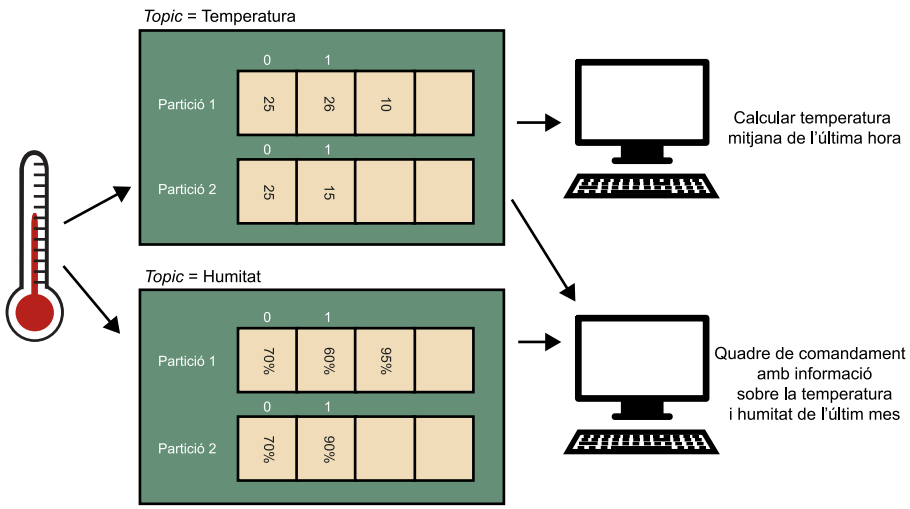
A títol d'exemple ara veurem com es gestionarien les dades següents en *streaming* utilitzant Apache Kafka. Suposem que les dades es generen cada segon segons les mesures de temperatura i humitat d'un sensor situat al carrer i que

s'envien via *streaming* a dos consumidors: un que calcula la temperatura mitjana de l'última hora i un altre que mostra un quadre de comandament amb informació de temperatura i humitat. Les dades que s'enviaran serien les següents:

```
1 (<25 graus, 70% humitat>, 1483309623)
2 (<25 graus, 70% humitat>, 1483309624)
3 (<26 graus, 60% humitat>, 1483309625)
4 (<15 graus, 90% humitat>, 1483309626)
5 (<10 graus, 95% humitat>, 1483309627)
```

Segons aquestes dades, s'ha decidit utilitzar dos *topics* diferents: un per a enviar dades sobre temperatura (anomenat *temperatura*) i un altre per a enviar dades sobre humitat en l'ambient (anomenat *humitat*). Suposem també que decidim utilitzar dues particions per a cada *topic* i que el que fem és distribuir mesures consecutives a particions diferents. Per tant, enviaríem els valors de temperatura 25 (línia 1), 26 (línia 3) i 10 (línia 5) a la partició 1 del *topic temperatura*, i els valors 25 (línia 2) i 15 (línia 4) a la partició 2 del *topic temperatura*. D'altra banda, els valors 70 (línia 1), 60 (línia 3) i 95 (línia 5) s'enviarien a la partició 1 del *topic humitat* i la resta de valors a la partició 2. Els estats de les particions després d'enviar les dades d'exemple serien els mostrats a la figura 5.

Figura 5. Exemple d'ús d'Apache Kafka



Respecte als consumidors, el primer consumidor només necessita dades de temperatura per calcular la temperatura mitjana. Per tant, només consultarà dades del *topic temperatura*. El segon consumidor haurà de proporcionar informació sobre la temperatura i la humitat i, per tant, haurà de consultar tots dos *topics*. Recordeu que els consumidors hauran de saber el *topic*, la partició i l'*offset* de les dades que es consultaran en cada moment.

Distribució i replicació

A Kafka les particions són la unitat de distribució i replicació.

Les particions es distribuiran entre diferents servidors del clúster, col·locant les particions d'un *topic* en diferents nodes sempre que sigui possible.

Les particions es replicaran en diferents servidors segons un factor de replicació indicat per l'usuari. La gestió de rèpliques es realitzarà seguint una estratègia *master-slave* asíncrona en la qual una partició actua com a líder. La partició capdavantera s'encarrega de gestionar totes les peticions de lectura i escriptura. Hi pot haver zero particions secundàries o més (*followers*, segons la terminologia Kafka). Aquestes particions secundàries replicaran de manera asíncrona les escriptures del líder i tindran un rol passiu: no podran rebre peticions de lectura/escriptura dels consumidors/productors. En cas que la partició líder caigui, una de les particions secundàries la substituirà.

En un clúster Kafka, sempre que sigui possible, cada node actuarà com a líder d'una de les seves particions amb l'objectiu de garantir que la càrrega del sistema estigui ben balancejada.

2. Preprocessament de dades massives

El propòsit fonamental de la fase de preprocessament o preparació de dades és manipular i transformar les dades brutes (*raw data*) perquè el contingut d'informació sigui coherent, homogeni i consistent. Aquest procés és molt important quan es recol·lecta informació de múltiples fonts de dades.

2.1. Tasques de preprocessament de dades

El preprocessament de dades engloba totes aquelles tècniques d'anàlisi i manipulació de dades que permet millorar la qualitat d'un conjunt de dades, de manera que les tècniques de mineria de dades que s'aplicaran posteriorment puguin obtenir-ne un millor rendiment.

En aquest sentit, podem agrupar les diferents tasques de preprocessament en dos grans grups, cadascun dels quals implica múltiples subtasques que depenen, en gran part, del tipus de dades que estem capturant i integrant en els sistemes d'informació analítics. Aquests són:

- tasques de neteja de dades (*data cleansing*)
- tasques de transformació de dades (*data wrangling*)

2.1.1. Neteja de dades

Les tasques de neteja de dades agrupen els processos de detectar i corregir (o eliminar) registres corruptes o imprecisos d'un conjunt de dades. Concretament, s'espera que aquestes tasques siguin capaces d'identificar parts incompletes, incorrectes, inexactes o irrellevants de les dades, per després reemplaçar, modificar o eliminar les dades que presentin problemes.

Després de la neteja, un conjunt de dades ha de ser coherent amb altres conjunts de dades similars en el sistema. Les inconsistències detectades o eliminades poden haver estat causades originàriament per errors en les dades originals (ja siguin errors humans o errors produïts en sensors o altres automatismes), per corrupció en la transmissió o l'emmagatzematge, o per diferents definicions de diccionari de dades de les diferents fonts de dades emprades.

Una de les primeres tasques que ha de fer el procés de neteja és la **integració de dades** (*data integration*). L'objectiu d'aquesta tasca és resoldre problemes de representació i codificació de les dades, amb la finalitat d'integrar correctament dades provinents de diferents plataformes per crear informació homogènia. En aquest sentit, aquesta tasca és clau en entorns de dades massives, on generalment recol·lectem informació des de múltiples fonts de dades, que poden presentar diversitat de representació i codificació de les dades.

Exemple: integració de dates

Un exemple molt clar de la tasca d'integració de dades es produeix quan recol·lectem informació sobre el moment en què s'han produït certs esdeveniments. Existeixen multitud de formats per a representar un moment concret en el temps.

Per exemple, podem representar el moment exacte en què s'ha produït un esdeveniment de les maneres següents:

- A partir d'una cadena de text que representa una data, en un format concret, com pot ser «YYYY-MM-DD hh:mm:ss». En aquest format s'indica la data a partir de l'any, el mes, el dia, l'hora, els minuts i els segons. A més, cal afegir-hi el tema de la zona horària.

- A partir del nombre de segons des de la data de l'1 de gener de l'any 1970, que és conegut com a UNIX *timestamp*. Aquest sistema és molt emprat en els sistemes operatius UNIX i derivats (per exemple, Linux) i identifica un moment del temps concret a partir d'un valor numèric. Per exemple, el valor 1502442623 equival a la data d'11 d'agost del 2017 a les 9 hores, 10 minuts i 23 segons.

Per tant, quan integrem dades de diferents fonts cal homogeneïtzar les dates perquè les anàlisis posteriors es puguin executar satisfactòriament.

El procés real de neteja de dades pot implicar l'eliminació d'errors tipogràfics o la validació i correcció de valors enfront d'una llista coneguda d'entitats. La validació pot ser estricta (com rebutjar qualsevol adreça que no tingui un codi postal vàlid) o difusa (com corregir registres que coincideixin parcialment amb els registres existents coneguts).

Una pràctica comuna de neteja de dades és la millora de dades, on les dades es completen en agregar informació relacionada. Per exemple, annexant adreces amb qualsevol número de telèfon relacionat amb aquesta adreça.

D'altra banda, la neteja de dades pròpiament dita inclou tasques «tradicionals» dins del procés de mineria de dades, com pot ser l'eliminació de valors extrems (*outliers*), la resolució de conflictes de dades, eliminació de soroll, valors perduts o absents, etc.

2.1.2. Transformació de dades

La transformació de dades és el procés de transformar i «mapejar» dades «en brut» en un altre format amb la intenció de fer-lo més apropiat i consistent per a les posteriors operacions d'anàlisi.

Això pot incloure l'agregació de dades, així com molts altres usos potencials. A partir de les dades extretes en cru (*raw data*) de l'origen de dades, es poden formatar les dades per posteriorment emmagatzemar-les en estructures de dades predefinides o, per exemple, utilitzar algorismes simples de mineria de dades o tècniques estadístiques per generar agregats o resums que posteriorment seran emmagatzemats al repositori analític.

Dins de les tasques de transformació de dades s'inclouen, entre d'altres, la consolidació de les dades de manera apropiada per a l'extracció d'informació, la sumarització de dades o les operacions d'agregació.

Una altra tasca important dins d'aquest grup és la reducció de les dades (*data reduction*), que inclou tasques com la selecció de dades rellevants per a les anàlisis posteriors o diferents vies per a la reducció de dades, com la selecció de característiques, selecció d'instàncies o discretització de valors.

2.2. Arquitectures de preprocessament

Hi ha dos factors clau que determinaran l'estructura o arquitectura del preprocessament de dades. Aquests factors són:

- La tipologia del procés de captura de les dades. En aquest sentit és important determinar si el sistema treballarà amb dades estàtiques o dades dinàmiques (en *streaming*).

- La complexitat de les operacions de neteja i transformació que s'hauran d'aplicar a les dades abans d'emmagatzemar-les al repositori analític.

Segons aquests factors i d'altres , trobem dos models principals d'estructura o arquitectura de preprocessament:

- preprocessament durant el procés de captura de les dades
- preprocessament independent del procés de captura de les dades

Al primer model (figura 6), les dades es netegen i transformen durant el procés de captura de dades i s'emmagatzemen una vegada netejades i transformades. Podem dir que els processos de captura i preprocessament es realitzen en sèrie, però sense requerir un sistema d'emmagatzematge intermedi.

Per contra, al segon model (figura 7), les dades són capturades i emmagatzemades en un sistema d'emmagatzematge intermedi, conegut com a *staging area*, en què les dades són emmagatzemades temporalment. Les tasques de preprocessament agafen les dades d'aquests sistemes d'emmagatzematge intermedi i realitzen les operacions oportunes sobre les dades; posteriorment emmagatzemen les dades als repositoris analítics finals, on s'emmagatzemen de forma duradora perquè posteriorment es puguin analitzar.

Figura 6. Esquema d'una arquitectura sense *staging area*

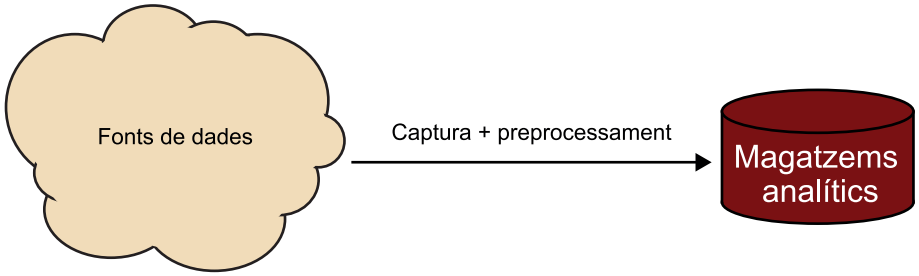
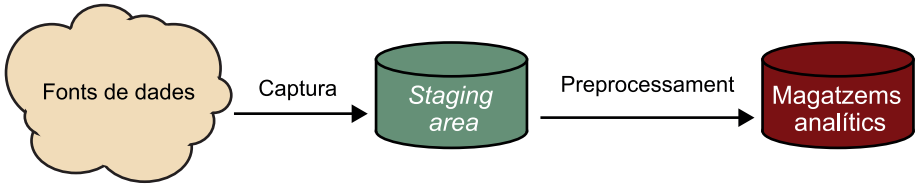


Figura 7. Esquema d'una arquitectura que incorpora *staging area*



El principal avantatge de l'arquitectura amb *staging area* és la independència entre els processos de captura i preprocessament de dades. És a dir, els dos processos queden «desacoblats», amb la qual cosa evitem que un pugui influenciar negativament en el desenvolupament de l'altre.

Això està especialment indicat en el cas de treballar amb dades en *streaming*, ja que es poden produir pics de producció de dades que arribin a col·lapsar el sistema de preprocessament, especialment en el cas que aquest hagi de realitzar alguna tasca de certa complexitat. Per contra, si s'usa una arquitectura de *staging area*, aquesta «absorbeix» el pic de creixement de dades, mentre que el preprocessament de dades continua independentment.

També és especialment interessant en el cas de realitzar operacions de preprocessament de certa complexitat que podrien arribar a provocar la saturació del procés de captura i provocar la pèrdua de dades, atès que el sistema podria no ser capaç de processar i emmagatzemar la informació al ritme que es produeix.

3. Emmagatzematge de dades massives

Un dels aspectes en els quals les dades massives han tingut l'impacte més directe és el de l'emmagatzematge de dades. Molts escenaris que requereixen el maneig de grans volums de dades obliguen les empreses i institucions a adoptar solucions de emmagatzematge capaces de guardar una gran quantitat de dades alhora que ofereixen altes prestacions, escalabilitat i fiabilitat enfront de qualsevol problema i/o error.

Una vegada les dades han estat capturades i preprocessades cal emmagatzemar-les perquè puguin ser analitzades posteriorment. Fins i tot en els sistemes basats en temps real (*streaming*) se sol emmagatzemar una còpia de les dades paral·lelament al seu processament, ja que en cas contrari es perdrien les dades una vegada processades.

En aquest sentit, l'emmagatzematge de dades massives presenta dues opcions principals:

- sistemes de fitxers distribuïts
- bases de dades, principalment de tipus NoSQL

Els sistemes de fitxers distribuïts es basen en el funcionament dels sistemes de fitxers tradicionals, gestionats pel sistema operatiu, però en aquest cas s'emmagatzemen les dades de forma transversal en un conjunt heterogeni d'ordinadors.

Vegeu també

A l'apartat 4 veurem els sistemes de fitxers distribuïts, i posarem especial esment en els sistemes més coneguts i utilitzats actualment.

D'altra banda, les bases de dades NoSQL són una opció d'emmagatzematge massiu i distribuït molt emprat per qualsevol empresa o institució que treballi amb dades massives.

Vegeu també

Veurem una introducció a les bases de dades NoSQL i n'estudiarem els diversos tipus i les seves característiques principals a l'apartat 5, tot i que una descripció detallada de totes elles queda fora de l'abast d'aquests materials.

4. Sistemes de fitxers distribuïts

Podem identificar, a grans trets, dos grans sistemes d'emmagatzematge d'altes prestacions.

En primer lloc, existeixen els sistemes d'accés compartit a disc o *shared-disk file system*, com l'*storage-area network* (SAN). Aquest sistema permet que diversos ordinadors puguin accedir directament al disc a nivell de bloc de dades, i que sigui el node client el que tradueixi els blocs de dades a fitxers.

Hi ha diferents aproximacions arquitectòniques a un sistema d'arxius de disc compartit. Alguns distribueixen informació d'arxius en tots els servidors d'un clúster (totalment distribuïts). Uns altres utilitzen un servidor de metadades centralitzat, però en tots dos casos s'aconsegueix el mateix resultat: permetre que tots els servidors tinguin accés a les dades en un dispositiu d'emmagatzematge compartit.

Metadada

Les metadades són dades que en descriuen d'altres. En general, un grup de metadades es refereix a un grup de dades que descriuen el contingut informatiu d'un objecte conegut com a recurs.

El problema principal d'aquests sistemes és que el seu model d'emmagatzematge requereix un subsistema de disc extern relativament car (per exemple, Fibre Channel / iSCSI) a més de commutadors, adaptadors, etc. No obstant això, permet que les fallades de disc es controlin al subsistema extern.

Una altra aproximació, molt més utilitzada actualment, és un sistema d'emmagatzematge distribuït o sistemes d'arxius distribuïts, en què els nodes no comparteixen l'accés a dades a nivell de bloc, però proporcionen una vista unificada, amb un espai de noms (*namespace*) global. La diferència resideix en el model utilitzat per a l'emmagatzematge de blocs subjacent. A diferència del model anterior, cada node té el seu propi emmagatzematge a nivell de bloc de dades. L'abstracció d'aquests blocs als mateixos fitxers es gestiona a un nivell superior.

Aquests sistemes usualment es construeixen utilitzant *commodity hardware* o maquinari estàndard, de menor cost (com discos SATA/SAS). La seva escalabilitat és superior als sistemes de disc compartit i, tot i que pot oferir pitjors latències, existeixen estratègies per compensar aquest problema, com l'ús extensiu de memòries cau (caches), replicació, etc.

En aquest apartat ens centrarem en el sistema d'arxius distribuït més popular actualment, conegut com a HDFS.

4.1. Què és HDFS?

El sistema d'arxius distribuïts Hadoop (d'ara endavant HDFS, *Hadoop distributed filesystem*) és un subprojecte d'Apache Hadoop* i forma part de l'ecosistema d'eines de dades massives que proporciona Hadoop.

* https://hadoop.apache.org

HDFS és un sistema d'arxius altament tolerant a fallades dissenyat per funcionar amb l'anomenat maquinari de baix cost (*commodity hardware*), que permet reduir els costos d'emmagatzematge significativament. Proporciona accés d'alt rendiment a grans volums de dades i és adequat per a aplicacions de processament distribuït, tal com el seu nom indica.

Commodity hardware

La computacció de «baix cost» implica l'ús d'un gran número de components de computació ja disponibles per a la computació paralel·la, amb l'objectiu d'obtenir la major quantitat de computació útil a baix cost.

Té moltes similituds amb altres sistemes d'arxius distribuïts, però al seu torn presenta diferències importants:

- Segueix un model d'accés *write-once-read-many*. Això és, restringint rigorosament l'escriptura de dades a un escriptor o *writer* es relaxen els requisits de control de concurrència. A més, un arxiu una vegada creat, escrit i tancat no necessita ser canviat, la qual cosa simplifica la coherència de les dades i hi permet un accés d'alt rendiment. Els bytes sempre s'annexen al final d'un flux, i està garantit que els fluxos de bytes siguin emmagatzemats en l'ordre escrit.

- Una altra propietat interessant d'HDFS és el desplaçament de la capacitat de computació a les dades en lloc de moure les dades a l'espai de l'aplicació. Així, un càlcul sol·licitat per una aplicació és molt més eficient si s'executa prop de les dades en les quals opera. Això és especialment cert quan la grandària del conjunt de dades és enorme. Així es minimitza la congestió de la xarxa i augmenta el rendiment global del sistema. HDFS proporciona interfícies perquè les aplicacions es moguin més a prop d'on són les dades.

I entre les principals característiques d'HDFS podem citar les següents:

- És tolerant a fallades, les detecta i aplica tècniques de recuperació ràpida i automàtica. Les fallades al maquinari solen ocórrer amb certa freqüència i, tenint en compte que una instància d'HDFS pot consistir en centenars o milers de màquines i que cadascuna emmagatzema part de les dades del sistema d'arxius, existeix una elevada probabilitat que no tots els nodes funcionin correctament en tot moment. Per tant, la detecció de fallades i la seva recuperació ràpida i automàtica és un dels objectius clau d'HDFS, que defineix la seva arquitectura i disposa, a més, de mecanismes de reemplaçament en cas de fallades greus del sistema HDFS (*hot swap*), com es descriurà més endavant. A més, és un sistema fiable gràcies al manteniment automàtic de múltiples còpies de dades per a maximitzar-ne la integritat.

- Accés a dades per mitjà de Streaming MapReduce. Les aplicacions que s'executen a HDFS necessiten tenir accés de *streaming* als seus conjunts de

dades, ja que són múltiples els escenaris i les eines de dades massives que ofereixen processament en *streaming* sobre dades a HDFS. Així mateix, ha d'estar perfectament integrat als sistemes de processament *batch*, com per exemple MapReduce. Així, HDFS no ofereix un bon rendiment per treballar de manera interactiva, ja que està dissenyat per donar alt rendiment d'accés a grans volums de dades en lloc de baixa latència d'accés a elles.

- HDFS ha estat dissenyat per a ser fàcilment portable d'una plataforma a una altra, la qual cosa el fa compatible en entorns de maquinari heterogeni (*commodity hardware*) i diferents sistemes operatius. HDFS està implementat amb llenguatge Java, de manera que qualsevol màquina que admeti la programació Java pot executar-lo. Això facilita l'adopció generalitzada d'HDFS com una plataforma d'elecció per a un gran conjunt d'aplicacions. A més, aquest factor incrementa la seva escalabilitat, ja que la integració d'un nou node en un clúster HDFS imposa poques restriccions.

- Les aplicacions que s'executen sobre dades a HDFS solen treballar amb grans conjunts de dades, de l'ordre de GB a TB o més. HDFS està dissenyat per a treballar sobre grandàries d'arxiu grans, de manera que optimitzi tots els paràmetres relacionats amb el rendiment, com el moviment de dades entre nodes i entre els mateixos *racks*, i maximitzi la localització de la computació allà on són les dades.

- Proporciona diverses interfícies perquè les aplicacions treballin sobre les dades emmagatzemades, així que maximitza la facilitat d'ús i l'accés.

4.2. Arquitectura

HDFS no és un sistema de fitxers regular com podria ser ext3* o NTFS,** que estan construïts sobre el *kernel* del sistema operatiu, sinó que és una abstracció sobre el sistema de fitxers regular. HDFS és un servei que s'executa en un entorn distribuït d'ordinadors o clústers seguint una arquitectura de mestre (al qual anomenarem *NameNode* d'ara endavant) i esclau (*DataNode* d'ara endavant). En un clúster sol haver-hi un sol *NameNode* i diversos *DataNodes*, un per ordinador al clúster.

*** http://bit.ly/2Cvkeuz**
**** http://bit.ly/2EkhQY1**

4.2.1. *NameNode* i *DataNode*

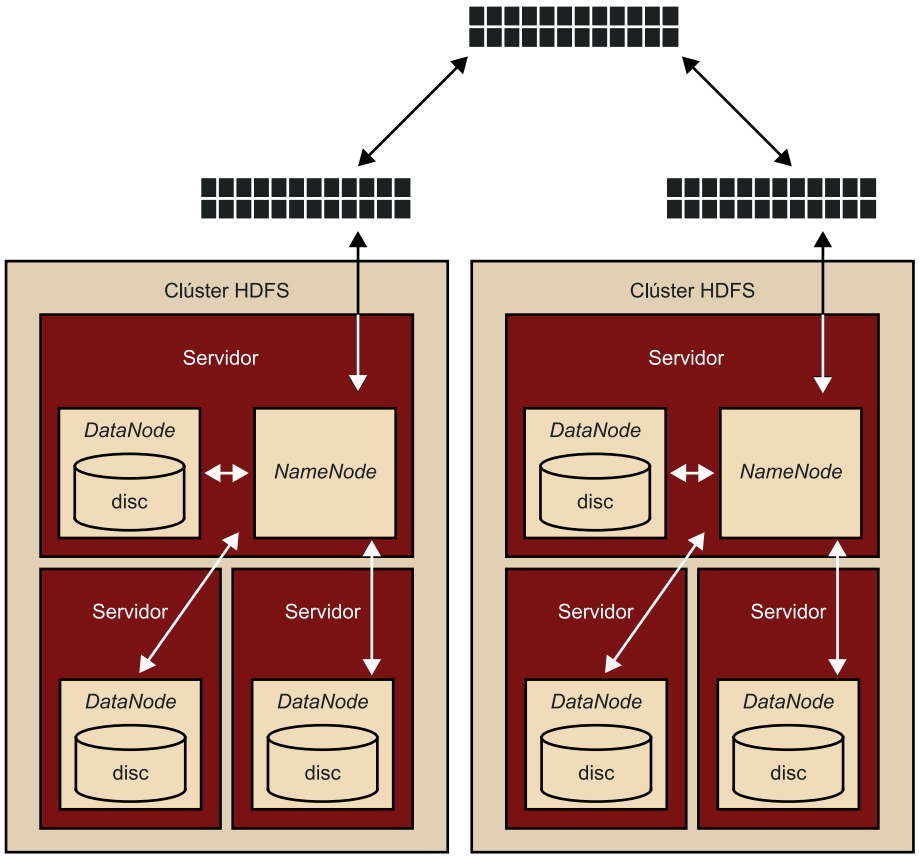
El *NameNode* gestiona les operacions del *namespace* del sistema d'arxius com obrir, tancar i renomenar arxius i directoris, i regula l'accés del client als arxius. També és responsable de l'assignació de les dades al seu corresponent *DataNode*.

Els *DataNode* manegen les sol·licituds de lectura i escriptura dels clients de l'HDFS i són responsables de gestionar l'emmagatzematge del node en el qual

resideixen, així com crear, eliminar i duplicar les dades d'acord amb les instruccions del *NameNode*.

Els *NameNodes* i els *DataNodes* són components de programari dissenyats per executar-se d'una manera desacoblada entre ells als nodes del clúster. Tal com mostra la figura 8, l'arquitectura típica consta d'un node en el qual s'executa el servei *NameNode* i possiblement el servei *DataNode*, mentre que en cadascuna de les altres màquines del clúster s'executa el *DataNode*.

Figura 8. Arquitectura típica d'un clúster HDFS



Els *DataNodes* consulten contínuament al *NameNode* noves instruccions. No obstant això, el *NameNode* no es connecta al *DataNode* directament, simplement respon a aquestes consultes. Al mateix temps, el *DataNode* manté un canal de comunicacions obert, de manera que el codi client o altres *DataNodes* poden escriure o llegir dades quan siguin requerides en altres nodes.

Tots els processos de comunicació HDFS es basen en el protocol TCP/IP. Els clients HDFS es connecten a un port TCP (protocol de control de transferència) obert al *NameNode* i s'hi comuniquen utilitzant un protocol basat en RPC (*remote procedure call*). Al seu torn, els *DataNodes* es comuniquen amb el *NameNode* usant un protocol propietari.

RPC

En computació distribuïda, la crida a un procediment remot (*remote procedure call*, RPC) és un programa que utilitza un ordinador per a executar codi en una altra màquina remota sense haver de preocupar-se per les comunicacions entre ambdues.

4.2.2. Espai de noms (*namespace*)

HDFS admet una organització d'arxius jeràrquica (arbres de directoris) similar a la majoria dels sistemes d'arxius existents. Un usuari o una aplicació poden crear directoris i emmagatzemar arxius dins d'aquests directoris, i poden crear, eliminar i moure arxius d'un directori a un altre. Tot i que l'arquitectura d'HDFS no ho impedeix, encara no és possible crear enllaços simbòlics (*symbolic links*). El *NameNode* és el responsable de mantenir l'espai de noms del sistema d'arxius i qualsevol canvi en aquest espai és registrat.

Organització de dades

El rendiment d'HDFS és millor quan els fitxers que es volen gestionar són grans. Es divideix cada arxiu en blocs de dades, típicament de 64 MB (totalment configurable); per tant, cada arxiu consta d'un o més blocs de 64 MB. HDFS intenta distribuir cada bloc en *DataNodes* separats.

Cada bloc requereix de l'ordre de 150 bytes* d'*overhead* al *NameNode*. Així, si el nombre de fitxers és molt elevat (poden arribar a ser milions) és possible arribar a saturar la memòria del *NameNode*.

* http://bit.ly/2DDlVEq

Taula 1. Impacte pel que fa a la gestió de memòria del *NameNode* en la gestió d'un sol fitxer d'1 GB de grandària enfront de particionar 1 GB en mil fitxers d'1 MB

Estrategia	Metadades	Recursos
1 fitxer d'1 GB	1 entrada en el registre de nombres, 16 blocs de dades, 3 rèpliques	49 entrades
1.000 fitxers d'1 MB	1.000 entrades en el registre de nombres, 1.000 blocs de dades, 3 rèpliques	4.000 entrades

Els arxius de dades es divideixen en blocs i es distribueixen mitjançant el clúster en el moment de ser carregats. Cada fitxer es divideix en blocs (de la grandària de bloc definit) que es distribueixen amb el clúster.

Quan es distribueixen les dades entre diferents màquines del clúster, s'elimina el *single point of failure* (SPOF) o únic punt de fallada. En el cas de fallada d'un dels nodes no es perd un fitxer, ja que es pot recompondre a partir de les seves rèpliques. Aquesta característica també facilita el processament en paral·lel, perquè diferents nodes poden processar blocs de dades individuals d'una manera concurrent.

Exemple: procés de creació d'un fitxer en HDFS

La figura 9 mostra a manera d'exemple com un fitxer donat es divideix en quatre blocs (B1, B2, B3 i B4) i en el moment de ser carregat a l'HDFS es reparteix entre els diferents nodes. S'emmagatzemen fins a tres còpies de cada bloc a causa de la replicació de dades (que es descriurà en apartats posteriors).

Com ja s'ha introduït anteriorment, la grandària del bloc sol ser de 64 MB o 128 MB, tot i que és altament configurable per mitjà dels paràmetres de configuració, que es troben al fitxer *hdfs-site.xml*. A continuació, es mostra un exemple de configuració d'aquest valor:

```
<property>
        <name>dfs.blocksize</name>
        <value>134217728</value>
</property>
```

Figura 9. Distribució de blocs de dades en un clúster HDFS entre els diferents *DataNodes*

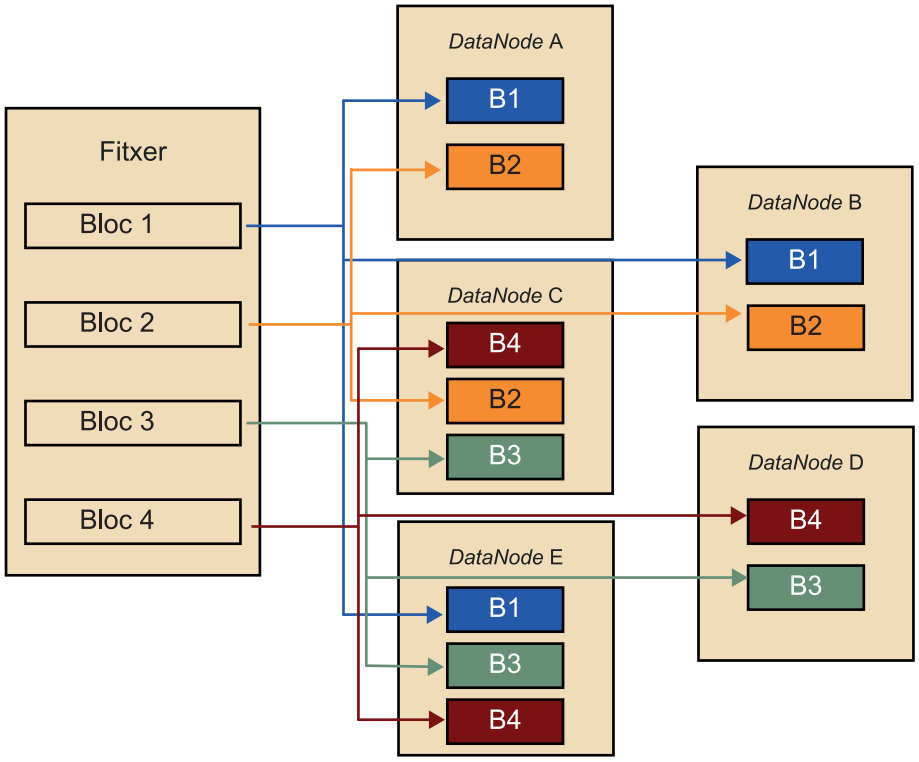


Figura 9

Noteu que a la figura es creen tantes còpies com rèpliques tingui definides el sistema.

Així doncs, treballar amb fitxers grans comporta diversos avantatges:

- Millor rendiment del clúster mitjançant l'*streaming* de grans quantitats de dades.
- Millor rendiment del clúster que eviti costoses càrregues d'arrencada per a quantitats de dades petites.
- Millor escalabilitat del *NameNode* associada amb la càrrega de metadades de menys fitxers però de més grandària. Si el *NameNode* treballa amb fitxers petits la seva operativa és més costosa i pot conduir a problemes de memòria.
- Més disponibilitat, ja que l'estat del sistema d'arxius HDFS està contingut a la memòria.

Replicació de dades

Com s'ha indicat anteriorment, HDFS és tolerant a fallades mitjançant la replicació de blocs de dades. Una aplicació pot especificar el nombre de rèpliques d'un arxiu en el moment en què es crea, i aquest nombre es pot canviar poste-

riorment. El *NameNode* també és el responsable de gestionar la replicació de dades.

HDFS utilitza un model d'assignació de blocs replicats a diferents nodes, però també pot distribuir blocs a diferents *racks* d'ordinadors (la crida *rack-aware replica placement policy*), la qual cosa el fa molt competitiu enfront d'altres solucions d'emmagatzematge distribuït, ja no solament enfront de problemes de fiabilitat sinó també d'eficiència en l'ús de la interconnectivitat de la xarxa interna del clúster.

En efecte, els grans entorns HDFS normalment operen per mitjà de múltiples instal·lacions d'ordinadors i la comunicació entre dos nodes de dades en instal·lacions diferents sol ser més lenta que a la mateixa instal·lació. Per tant, el *NameNode* intenta optimitzar les comunicacions entre els *DataNodes* identificant la seva ubicació pel seu *rack ID* o identificador de *rack*. Així doncs, HDFS és conscient de la ubicació de cada *DataNode*, la qual cosa li permet optimitzar la tolerància a fallades (*DataNodes* amb rèpliques en *racks* diferents) i balancejar també el trànsit de xarxa entre *racks* diferents.

El factor de replicació es defineix al fitxer de configuració *hdfs-site.xml* i l'exemplifiquem a continuació:

```
<property>
    <name>dfs.replication</name>
    <value>3</value>
</property>
```

Exemple: emmagatzematge d'un fitxer

Imaginem un cas en què un usuari vol emmagatzemar dos fitxers (*file1.log* i *file2.log*) en el sistema HDFS d'un clúster de cinc nodes (A, B, C, D i E).

HDFS divideix cada fitxer en blocs de dades. Donada la grandària dels fitxers exemple, suposarem que *file1.log* es divideix en tres blocs, B1, B2 i B3. Mentre que *file2.log* es divideix en dos blocs més (B4 i B5). Els blocs es distribueixen entre els diferents nodes i, a més, suposarem un factor de replicació 3, de manera que de cadascun dels blocs es fan tres còpies. Així, el bloc B1 s'emmagatzema en els nodes A, B i D; el bloc B2 en els blocs B, D i E. Es pot veure la distribució completa a continuació.

Correspondència entre fitxer i blocs:

- *file1.log*: B1, B2 i B3
- *file2.log*: B4 i B5

Correspondència entre blocs i nodes:

- B1: A, B i D
- B2: B, D i E
- B3: A, B i C
- B4: E, B i A
- B5: C, E i D

El *NameNode* és el que emmagatzema tota la informació relativa als fitxers, els seus blocs corresponents i la seva localització al clúster. La correspondència o

el mapeig (*mapping*) de cada fitxer amb els seus blocs respectius està disponible en memòria i també en disc, per a la seva recuperació en cas de problemes; no obstant això, el mapeig entre blocs de dades i els nodes en què estan emmagatzemats només està disponible en memòria, en què els *DataNodes* són els que notifiquen al *NameNode* els blocs sota la seva supervisió en un procés anomenat *heartbeating* (que es descriu més endavant i que sol tenir un valor d'uns deu segons).

Exemple: lectura d'un fitxer

En el cas d'haver d'accedir als fitxers anteriorment descrits, suposem que el client demana al *NameNode* el fitxer *file2.log* i la resposta és la llista dels blocs de dades que el componen (B4 i B5 en aquest cas). Tot seguit, el client li pregunta on trobar-los i el *NameNode* li respon amb els nodes en què es troben.

En aquest cas:

- B4: nodes A, B i E
- B5: nodes C, E i D

El *NameNode* retorna al client una llista amb els nodes ordenats per proximitat a ell, seguint el criteri que presentem a continuació:

1) es troba a la mateixa màquina,
2) es troba al mateix *rack* (*rack awarenes*),
3) es troba en algun dels nodes restants.

El client obté l'arxiu sol·licitant cadascun dels blocs directament des dels nodes en els quals estan emmagatzemats i seguint els criteris de proximitat ja esmentats. Així, el client intentarà recuperar el bloc des del primer node de la llista (el node més proper al client, tal com el *NameNode* li ha reportat), i si aquest node no està disponible el client ho intentarà des del segon, i si tampoc no està disponible ho intentarà amb el tercer, etc. Les dades es transfereixen directament entre el *DataNode* i el client, sense involucrar-hi el NameNode, així la comunicació entre el client i el *NameNode* és mínima, amb un trànsit de xarxa baix, i tot el procés optimitza tant com sigui possible l'ús d'ample de banda de la xarxa de comunicacions.

4.3. Fiabilitat

La fiabilitat és un dels objectius més importants d'un sistema d'arxius com HDFS. Així, en primer lloc és necessari detectar els problemes, ja sigui al *NameNode*, els *DataNodes* o les fallades pròpies de la xarxa.

Disponibilitat (*secondary NameNode*)

Com es pot veure, el servei *NameNode* ha d'estar actiu contínuament, ja que si s'atura el clúster estarà inaccessible. Quines mesures es prenen per evitar que això passi? HDFS disposa de dues maneres de funcionament:

- mode d'alta disponibilitat
- mode clàssic

Al **mode d'alta disponibilitat** hi ha un *NameNode* actiu, anomenat principal o primari, i un segon *NameNode* en mode *stand-by*, anomenat *mirror*, que

pren el relleu del *NameNode* principal en cas de fallada, en un procediment anomenat *hot swap*.

Al **mode clàssic** hi ha un *NameNode* principal o primari i un *NameNode* secundari que té funcions de manteniment i optimització, però no de *backup*. El *NameNode* secundari és un servei que s'executa en mode *daemon* (és a dir, un dimoni sempre actiu), que s'encarrega d'algunes tasques de manteniment pel *NameNode* com les que descrivim a continuació:

- Manté una imatge del mapeig de fitxers amb els seus respectius blocs de dades i la seva ubicació als *DataNodes* (anomenat *FsImage*).

- Els canvis al sistema d'arxius no s'integren immediatament a l'arxiu d'imatge; en el seu lloc, l'arxiu d'imatge principal s'actualitza en mode *lazy* (és a dir, només sota demanda).

- Els canvis al sistema d'arxius es graven en uns arxius separats coneguts com a *EditLogs*, anàlegs a un punt de control o *breakpoints* en un registre de transaccions d'una base de dades relacional. A l'*EditLog* es registra persistentment cada transacció que s'aplica a les metadades del sistema d'arxius HDFS. Si els arxius *EditLog* o *FsImage* es danyen, la instància d'HDFS a la qual pertanyen deixa de funcionar. Per tant, un *NameNode* admet múltiples còpies dels arxius *FsImage* i *EditLog*, i qualsevol canvi en qualsevol d'aquests arxius es propaga a totes les còpies. Quan un *NameNode* es reinicia, utilitza l'última versió consistent d'*FsImage* i *EditLog* per inicialitzar-se. De forma periòdica es fusionen els *EditLog* en l'arxiu d'imatge principal *FsImage*, típicament una vegada per hora, la qual cosa accelera el temps de recuperació si falla el *NameNode*. Quan s'inicialitza un *NameNode*, llegeix l'arxiu *FsImage* juntament amb els *EditLogs* i hi aplica les transaccions i la informació d'estat que estan registrades en aquests arxius.

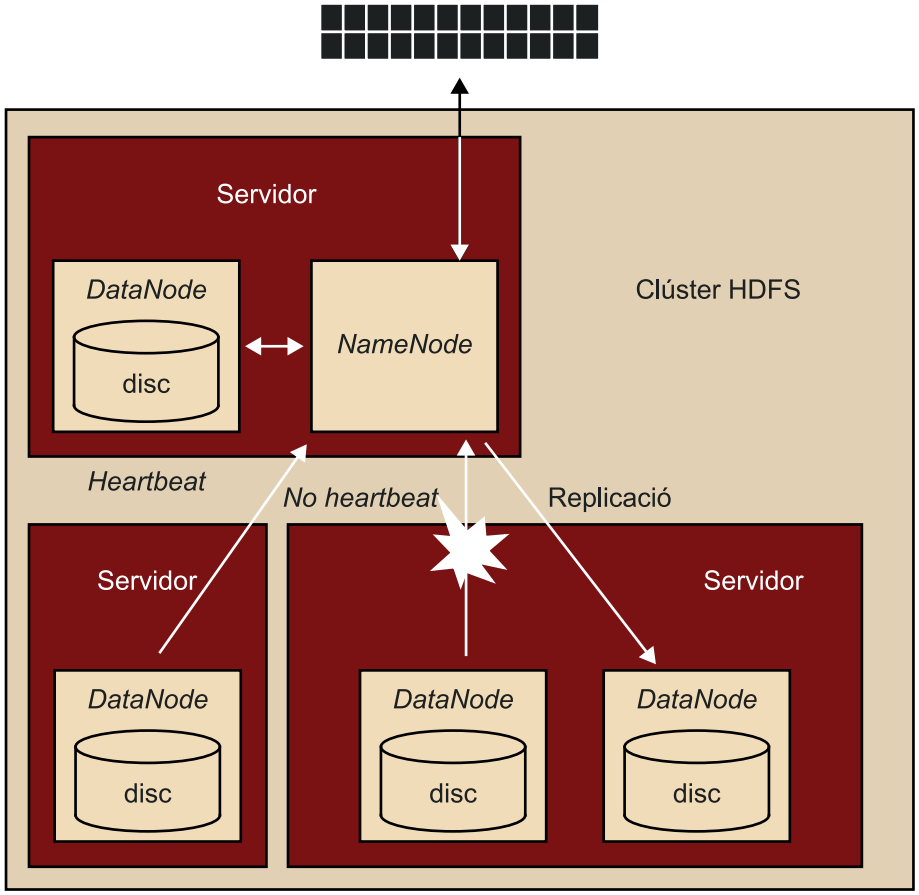
Malgrat tot això, el *NameNode* secundari no és un node de *backup* que reemplaça el *NameNode* en cas de fallada, de manera que en escenaris d'alta disponibilitat no se sol fer servir.

4.3.1. HDFS *heartbeats*

Existeixen multitud de situacions que poden causar pèrdua de connectivitat entre el *NameNode* i els *DataNodes*. Així doncs, per detectar aquests problemes HDFS utilitza els *heartbeats* (pulsacions) que verifiquen la connectivitat entre tots dos. Cada *DataNode* envia missatges periòdicament (els *heartbeats*) al seu *NameNode* i aquest pot detectar la pèrdua de connectivitat si deixa de rebre'ls. El *NameNode* marca com *DataNodes* caiguts aquells que no responen als *heartbeats*, i para d'enviar-los més sol·licituds. Les dades emmagatzemades en un node caigut ja no estan disponibles per a un client HDFS des d'aquest node, que s'elimina efectivament del sistema. Si la caiguda d'un node fa que el factor de replicació dels blocs de dades caigui per sota del seu valor mínim,

el *NameNode* inicia el procés de creació d'una rèplica addicional per retornar el factor de replicació a un estat normal. Aquest procés s'il·lustra a la figura 10.

Figura 10. Diagrama de replicació a causa d'una fallada al *heartbeat*



4.3.2. Reajustament de blocs de dades

Els blocs de dades HDFS no sempre es poden col·locar uniformement entre els *DataNodes*, la qual cosa significa que l'espai utilitzat per a un d'aquests nodes o més pot estar infrautilitzat. Per evitar que això passi, HDFS realitza tasques d'equilibrat de blocs entre nodes utilitzant diversos models:

- Un model pot moure blocs de dades d'un *DataNode* a un altre automàticament si l'espai lliure en aquest node cau massa baix.

- Un altre model consisteix a crear dinàmicament rèpliques addicionals i reajustar altres blocs de dades en un clúster si es produeix una pujada sobtada en la demanda d'un arxiu donat.

Un motiu habitual per a reajustar blocs és l'addició de nous *DataNodes* a un clúster. En col·locar nous blocs, el *NameNode* segueix diversos criteris per a la seva assignació:

- polítiques de replicació, descrites anteriorment;
- uniformitat en la distribució de dades entre nodes;
- reducció d'utilització d'ample de banda pel trànsit de dades a la xarxa interna.

Així, el reajustament de blocs de dades del clúster d'HDFS és un mecanisme que s'utilitza per a sostenir la integritat de les seves dades.

HDFS també utilitza mecanismes per validar les dades del sistema HDFS i emmagatzema al seu torn sumes de comprovació (o *chekcsums*) en arxius separats i ocults al mateix espai de noms que les dades reals. Així, quan un client recupera arxius de l'HDFS, pot verificar que les dades rebudes coincideixin amb la suma de comprovació emmagatzemada en l'arxiu associat.

Checksum

Checksum és una funció *hash* que té com a propòsit principal detectar canvis accidentals en una seqüència de dades per protegir-ne la integritat, verificant que no hi hagi discrepàncies entre els valors obtinguts en fer una comprovació inicial i una altra final després de la transmissió.

4.3.3. Permisos d'usuaris, fitxers i directoris

HDFS implementa un model de permisos per a arxius i directoris que té molt en comú amb el model POSIX (*portable operating system interface*).

Per exemple, cada arxiu i directori està associat a un propietari i a un grup, de manera que admet permisos de lectura (r – *read*), escriptura (w – *write*) i execució (x – *execution*). Degut a que no hi ha un concepte d'execució d'arxius dins d'HDFS, el permís d'execució *x* té un significat diferent. En aquest cas, l'atribut *x* indica el permís per accedir a un directori fill d'un directori pare determinat. El propietari d'un arxiu o directori és la identitat del procés client que el va crear i el grup és el grup del directori pare.

4.4. Interfície HDFS

Per accedir al servei HDFS disposem de diverses alternatives. A continuació, presentem les més utilitzades.

4.4.1. Línia de comandaments

HDFS organitza les dades, de cara a l'usuari, en fitxers i directoris de manera semblant als sistemes de fitxers més habituals, de manera que l'usuari pot consultar el contingut dels directoris mitjançant comandaments semblants als dels sistemes operatius de la família Unix/Linux, fent servir el prefix «hdfs dfs» o «hadoop fs» abans de cada ordre (són equivalents).

A continuació, mostrarem alguns ordres bàsics d'ús d'HDFS. No obstant això, per poder conèixer totes les opcions disponibles, n'hi ha prou a invocar l'ordre següent:

```
> hdfs dfs -help
```

Per poder consultar el contingut dels diferents directoris del sistema de fitxers farem servir el comandament `ls`, que llista els continguts dels directoris. Una diferència important amb el sistema de fitxers UNIX és que no existeix l'opció `cd`, amb la qual cosa no és possible moure'ns pel sistema de fitxers. L'ordre següent:

```
> hdfs dfs -ls
```

mostra els continguts del directori de l'usuari al sistema (que anomenarem *user* en aquests exemples). Així, un ordre equivalent seria:

```
> hdfs dfs -ls /home/user
```

Per llistar els continguts d'un directori donat anomenat <directori>, farem servir l'ordre:

```
> hdfs dfs -ls <directori>
```

Per crear un directori, farem servir l'ordre:

```
> hdfs dfs -mkdir <nom-directori>
```

És important remarcar que no es pot crear més d'un directori alhora. Si volem crear un directori dins d'un altre directori que no existeix, cal crear el primer abans de crear el segon:

```
> hdfs dfs -mkdir /directori1/directori2
```

Si <directori1> no existeix, caldrà crear primer aquest directori per poder crear a dins el segon:

```
> hdfs dfs -mkdir /directori1
> hdfs dfs -mkdir /directori1/directori2
```

De manera similar a altres sistemes de fitxers, es poden donar o restringir permisos d'accés i lectura de directoris i de fitxers mitjançant l'ordre `chmod`:

```
> hdfs dfs -chmod rwx /directori
```

en què <rwx> pot ser especificat en format numèric (per exemple, si <rwx> és igual a 777, es donaran permisos de lectura, escriptura i execució a tots els usuaris del sistema). És important tenir en compte que per accedir a directoris en HDFS, el directori ha de tenir permisos d'execució *x*, com s'ha indicat en apartats anteriors.

Una vegada creat un directori, hem de ser capaços de carregar arxius del sistema de fitxers local als directoris dins d'HDFS. Amb aquesta finalitat utilitzarem l'ordre `put`:

```
> hdfs dfs -put <fitxer-origen> <directori-desti-hdfs>
```

Per recuperar arxius del sistema HDFS al sistema de fitxers local farem servir l'ordre `get`:

```
> hdfs dfs -get <directori-origen-hdfs> <directori-desti>
```

També és possible copiar i moure directoris mitjançant l'ordre `cp` per a copiar i `mv` per a moure i reanomenar fitxers.

```
> hdfs dfs -cp <directori-origen-hdfs> <directori-desti-hdfs>
> hdfs dfs -mv <directori-origen-hdfs> <directori-desti-hdfs>
```

Ordres avançats

És possible combinar i descarregar diversos fitxers del sistema HDFS al sistema local en un sol fitxer mitjançant l'ordre `getmerge`:

```
> hdfs dfs -getmerge <origen-hdfs-1> <origen-hdfs-2>
        ... <fitxer-desti-local>
```

Aquest ordre crearà un sol fitxer amb una combinació dels fitxers d'origen de l'HDFS.

4.4.2. Projectes de l'ecosistema

Treballar sobre el sistema HDFS mitjançant la línia d'ordres és molt habitual per la seva similitud al funcionament de la línia d'ordres d'UNIX. No obstant això, l'ecosistema Hadoop ofereix també un conjunt d'eines que d'alguna manera permeten treballar sobre el sistema de fitxers HDFS i que ofereixen funcionalitats diverses.

Flume

Apache Flume* és una eina que té la principal funció de recollir, agregar i moure grans volums de dades provinents de diferents fonts cap al repositori HDFS en temps gairebé real. És útil en situacions en les quals les dades es creen de manera regular o espontània, de manera que a mesura que noves dades apareixen, ja siguin *logs* o dades d'altres fonts, són emmagatzemades en el sistema distribuït automàticament.

* https://flume.apache.org

L'ús d'Apache Flume no es limita a l'agregació de dades des de *logs*. Com que les fonts de dades són configurables, es pot utilitzar Flume per a recollir dades des d'esdeveniments lligats al trànsit de xarxa, xarxes socials, missatges de correu electrònic o a gairebé qualsevol tipus de font de generació de dades dinàmica.

Podem dividir l'arquitectura de Flume en els components següents:

- **Font externa**. Es tracta de l'aplicació o el mecanisme, com un servidor web o una consola d'ordres des de la qual es generen esdeveniments de dades que seran recollides per la font.

- **Agent**. És un procés Java que s'encarrega de recollir esdeveniments des de la font externa en un format recognoscible per Flume i passar-los transaccionalment al canal. Si una tasca no s'ha completat, a causa del seu caràcter transaccional, es reintenta per complet i s'eliminen resultats parcials.

- **Canal**. Un canal actuarà de magatzem intermedi entre l'agent i l'embornal (descrit a continuació). L'agent serà l'encarregat d'escriure les dades al canal i hi romandran fins que l'embornal o un altre canal els consumeixin. El canal pot ser la memòria, de manera que les dades s'emmagatzemen en la memòria volàtil (RAM), o poden ser emmagatzemades en disc, usant fitxers com a magatzem de dades intermèdies o una base de dades.

- **Embornal (*sink*)**. Estarà a càrrec de recollir les dades des del canal intermedi dins d'una transacció i de moure'ls a un repositori extern, a una altra font o a un canal intermedi. Flume és capaç d'emmagatzemar dades en diferents formats HDFS com, per exemple, text: SequenceFiles*, JSON** o Avro.***

*** http://bit.ly/2pZH2Qc**
**** http://www.json.org**
***** https://avro.apache.org**

Sqoop

Apache Sqoop**** (SQL To Hadoop) és una eina que té el propòsit d'intercanviar dades entre bases de dades relacionals (RBDMS) i HDFS. Fa ús del model MapReduce per realitzar la còpia de dades des d'una base de dades relacional, però fa servir solament tasques *map* en què el nombre de *mappers* per a realitzar la conversió és configurable. Sqoop analitza el nombre de files que s'importaran i divideix la tasca entre els diferents *mappers*, generant codi Java (que es pot conservar per a usos posteriors) per a cada taula que importarem.

****** http://sqoop.apache.org**

De la mateixa manera que les dades poden importar-se a l'HDFS, també poden exportar-se a una base de dades des d'un repositori HDFS.

L'eina disposa d'una consola per a realitzar les tasques d'importació i exportació del tipus:

```
> sqoop import –connect jdbc:mysql:<params>
```

Ofereix compatibilitat amb gairebé qualsevol tipus de base de dades mitjançant una connexió JDBC genèrica. No obstant això, també disposa de l'anomenat mode directe, en el qual el rendiment és molt millor gràcies a l'ús d'utilitats pròpies de cada servidor de base de dades.

Tot i que Sqoop pot executar-se en paral·lel, com més gran sigui el paral·lelisme, més gran serà la càrrega sobre la base de dades, així que aquest paral·lelisme ha de dimensionar-se adequadament.

Fins ara, Sqoop era una eina «client», és a dir, el client mateix s'encarregava de connectar-se a la base de dades (amb usuari i contrasenya) i a l'HDFS alhora per a realitzar la transferència. Així, va aparèixer una millora amb Sqoop2, en la qual es defineix una nova arquitectura client-servidor. El client, aquesta vegada, només necessita accés al servidor, de manera que no té accés directe a la base de dades i permet a l'administrador del sistema configurar Sqoop, limitant els recursos que es poden utilitzar, establint una política de seguretat, etc.

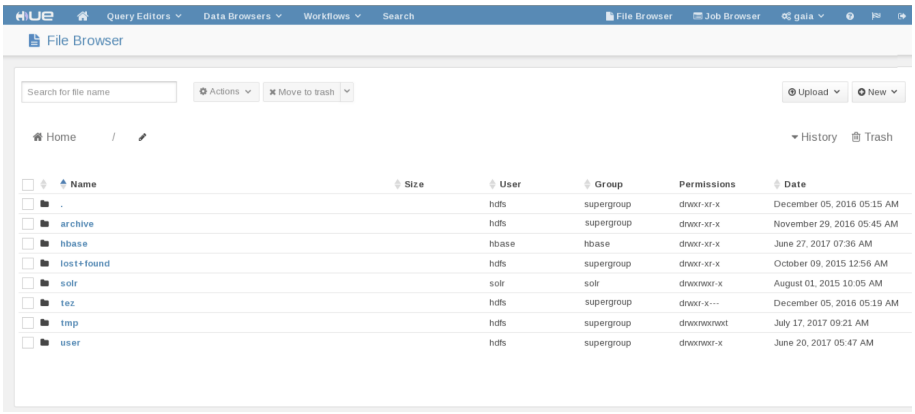
Hue

Hue (acrònim de Hadoop User Experience*) és una aplicació web que permet veure i administrar els directoris i arxius al sistema HDFS. Mitjançant HUE es poden crear, modificar (només directoris, no fitxers), moure, veure, pujar, descarregar i eliminar fitxers i directoris.

* http://gethue.com

Més enllà de les funcionalitats d'exploració del sistema HDFS, a més permet gestionar el *job scheduler* i *job manager* (servei YARN, que forma part de l'ecosistema Hadoop), ja que observa quins són les tasques en execució en el clúster. També s'integra amb altres eines de consulta sobre el sistema de fitxers com Impala i Hive. Es pot veure el panell principal d'HUE a la figura 11.

Figura 11. Panell principal de l'aplicació HUE (al menú superior es poden veure totes les seves funcionalitats)



Altres eines

Podem distingir altres eines:

- El sistema HDFS es pot muntar com qualsevol altre sistema de fitxers mitjançant l'eina FUSE,* la qual cosa una vegada muntat permet a l'usuari navegar pel sistema de fitxers amb els comandaments habituals `ls`, `cd`, etc.

  * http://bit.ly/2Cha3wu

- HttpFS** és un servidor que proporciona una porta d'enllaç HTTP REST que suporta totes les operacions del sistema d'arxius HDFS (lectura i escriptura).

  ** http://bit.ly/2C2EGBR

  HttpFS es pot utilitzar per a transferir dades entre clústers que executen versions diferents de Hadoop (superar problemes de versions d'RPC) o per accedir a dades a HDFS en un clúster darrere d'un servidor de seguretat (el servidor HttpFS actua com una porta d'enllaç i és l'únic sistema que es permet creuar el tallafocs al clúster).

  HttpFS es pot utilitzar per accedir a dades a HDFS utilitzant les utilitats HTTP (com `curl` i `wget`) i les biblioteques HTTP d'altres llenguatges.

5. Bases de dades NoSQL

Les bases de dades relacionals són àmpliament utilitzades en tot tipus de sistemes i organitzacions des de la dècada dels setanta. Els sistemes gestors de bases de dades relacionals són altament sofisticats i disten molt dels primers sistemes implementats en les albors del model relacional. Actualment, els sistemes relacionals estan àmpliament estesos, utilitzen llenguatges d'accés i manipulació de dades estàndards i estan molt optimitzats. Els avantatges dels sistemes relacionals són molts i molt variats; a continuació en presentem alguns:

- proporcionen un model de dades formal;
- proporcionen un llenguatge de consulta i una manipulació estàndard, l'última versió de la qual data del 2016;*
- proporcionen interfícies d'accés comuns i interoperables: ODBC, JDBC, etc.;
- proporcionen un sistema transaccional que garanteix un alta consistència de les dades;
- les dades emmagatzemades a les bases de dades relacionals són altament interoperables, ja que segueixen el mateix model de dades;
- la corba d'aprenentatge necessària per aprendre a utilitzar una base de dades relacional és mínima, ja que els diferents sistemes gestors de bases de dades utilitzen el mateix llenguatge per accedir i manipular les dades (SQL). Això garanteix també que hi hagi una gran quantitat de professionals habilitats per a treballar amb bases de dades relacionals eficientment;
- proporciona eines d'optimització molt potents i facilita la personalització del model físic de les dades per optimitzar encara més les operacions de dades en funció de les característiques dels problemes que es volen tractar;
- existeixen diferents extensions al sistema relacional per a suplir algunes de les seves deficiències en segons quins tipus de contextos: extensió geogràfica, extensió XML, etc.

*** ISO/IEC 9075-1:2016: vuitena revisió de l'estándard d'SQL (vegeu http://bit.ly/2lx9nYW).**

Tenint en compte la utilitat dels sistemes relacionals, la pregunta que se'ns planteja és: les bases de dades relacionals són la solució definitiva? Hem d'utilitzar-les per a resoldre tot tipus de problemes? La resposta, evidentment, és no.

Amb l'explosió d'internet i dels telèfons intel·ligents, entre d'altres, les bases de dades relacionals no sempre poden donar resposta a projectes que requereixen l'emmagatzematge d'un gran volum de dades (en general distribuïdes) i el seu tractament en temps real. És per això que en l'última dècada han aparegut altres tipus de base de dades, les anomenades bases de dades NoSQL i NewSQL, que pretenen suplir les bases de dades relacionals en aquells entorns en què el seu ús no és adequat.

En aquest apartat ens centrarem en les bases de dades NoSQL. En particular, començarem descrivint el concepte de NoSQL i parlant una mica dels seus orígens i característiques. Més endavant, identificarem els models de dades que suporta, per després descriure en més detalli cadascun d'ells i mostrar, mitjançant un exemple, com representar les dades en els diferents models de dades.

5.1. Què és NoSQL?

El terme NoSQL apareix per primera vegada l'any 1998 per denominar una base de dades creada per Carlo Strozzi que no utilitzava el llenguatge SQL i que té poc en comú amb les bases de dades NoSQL actuals. A partir del 2009 es va popularitzar aquesta denominació per referir-se a una nova generació de bases de dades no relacionals i altament distribuïdes que van aparèixer durant la primera dècada del 2000.

Tot i que moltes de les organitzacions que van desenvolupar i popularitzar aquesta nova tecnologia són ara grans corporacions (per exemple Google, Facebook o Amazon), la tecnologia NoSQL ha estat sempre lligada a l'auge de les *startup* i al moviment del codi obert. De fet, la majoria de bases de dades NoSQL són de codi obert.

És important destacar que la denominació *NoSQL* és polèmica. Si revisem els fets, i amb això els problemes que van portar al desenvolupament de les bases de dades NoSQL, no estan directament relacionats amb el llenguatge SQL, tal com el nom NoSQL pot suggerir. El fet que aquests nous models de bases de dades s'anomenin NoSQL és accidental. Per tant, sent rigorosos, hem de prendre el terme *NoSQL* simplement com una etiqueta que identifica un conjunt de bases de dades de nova aparició que no segueixen el model de dades relacional, i no com un acrònim que signifiqui «No s'usa SQL», «No SQL», «No solament SQL», ni res semblant.

Durant els últims anys, l'auge d'internet ha propiciat l'aparició d'aplicacions web (o serveis web) que han de garantir, idealment, dues propietats davant les fallades a la xarxa o als servidors de l'aplicació: la garantia que les dades estiguin disponibles tot i que la xarxa caigui o hi hagi servidors del sistema inoperatius, i la garantia que les dades de l'aplicació continuaran sent consistents (correctes) en cas de caiguda del sistema. En aquest context, l'any 2000 es va presentar la conjectura de Brewer, que més tard l'any 2002 es va

demostrar i va passar a ser el teorema CAP. Aquest teorema enumera tres propietats desitjables en un sistema distribuït: la consistència, la disponibilitat i la tolerància a particions (els seus acrònims formen el terme CAP). El teorema demostra que, en un sistema distribuït, solament dues d'aquestes propietats poden ser garantides simultàniament. És a dir, és impossible garantir les tres alhora. Això ha donat pas a la creació de diferents sistemes gestors de bases de dades que permeten satisfer diferents combinacions d'aquestes tres propietats i així poder donar solució a diferents problemes de dades eficientment.

En resum, la necessitat d'emmagatzemar i gestionar grans volums de dades, altament relacionats entre ells, en entorns distribuïts i en un ecosistema on les aplicacions han de respondre ràpidament, de manera continuada i per a tothom, ha causat que neixin noves generacions de bases de dades.

Les primeres bases de dades NoSQL que es coneixen són BigTable de Google (2003) i Marklogic (2005), tot i que l'any 1989 ja hi havia bases de dades amb característiques semblants. No obstant això, només s'utilitza el terme NoSQL per a bases de dades creades a partir de l'any 2000. Des de llavors hi ha hagut una gran proliferació de bases de dades NoSQL. Avui dia existeixen prop de dues-centes bases de dades NoSQL amb característiques molt variades i amb alts nivells de sofisticació.

Enllaç d'interès

Podeu veure una llista dels sistemes gestors de bases de dades NoSQL (i relacionals) ordenats per popularitat al lloc web de dbengines: https://db-engines.com.

5.1.1. Característiques de les bases de dades NoSQL

Les característiques principals de les bases de dades NoSQL són les següents:

- No ofereixen SQL com a llenguatge estàndard: les bases de dades NoSQL engloben diversos tipus de bases de dades, cadascun amb el seu llenguatge de consulta específic. En alguns casos el llenguatge utilitzat pot ser semblant a SQL, incloent-hi algunes extensions específiques per NoSQL. Això facilita l'accés a aquestes bases de dades a gent que sàpiga utilitzar SQL. La gran majoria de les bases de dades NoSQL es poden utilitzar mitjançant una API particular de programació, ja que aquest és l'ús més generalitzat.

- L'esquema de dades és flexible o no tenen un esquema predefinit (*schemaless*): es pot començar a afegir dades sense definir-ne prèviament l'esquema, com es fa al model relacional. Això permet molta més flexibilitat a l'hora de tractar dades heterogènies però dificulta la seva programació, de forma que la gestió de l'esquema (interpretació de les dades) es fa explícitament en el codi dels programes que accedeixen a la base de dades. El terme *schemaless* és una mica abusiu en aquest context perquè, encara que és cert que no existeix un esquema explícit, sempre hi ha un esquema implícit que s'usa i que condiciona com es distribuiran les dades en les bases de dades NoSQL. Un dels riscos que l'esquema quedi implícit és que es compromet la independència de les dades, un dels pilars sobre els quals es fonamenta el desenvolupament de les bases de dades relacionals.

- Les propietats ACID de les transaccions (atomicitat, consistència, aïllament i definitivitat) no sempre es garanteixen per complet: això es fa amb l'objectiu de millorar el rendiment i augmentar la disponibilitat. De fet, les bases de dades NoSQL que promouen l'alta disponibilitat acostumen a seguir un altre model transaccional denominat BASE.

- Permeten reduir, en part, els problemes de falta de concordança (*impedance mismatch*) entre les estructures de dades usades als programes i les bases de dades. És a dir, el format en què es guarda la informació en les bases de dades és proper al format utilitzat als programes que hi accedeixen.

- La majoria d'elles estan especialment dissenyades per a créixer, generalment en horitzontal (mitjançant la fragmentació de les dades/esquema i l'existència de còpies idèntiques de les dades en múltiples servidors).

- Generalment són distribuïdes —amb diferents models de distribució (*peer-to-peer*, *master-slave*, etc.)— i de codi obert (amb llicències dobles o basades en suport).

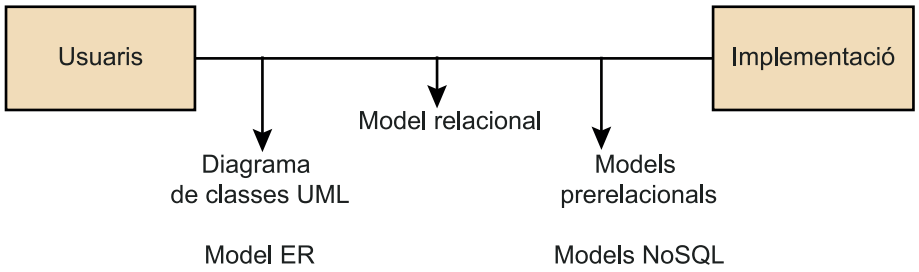
5.2. Models de dades NoSQL

Un model és un esquema que ens permet analitzar, conceptualitzar i representar una part de la realitat que ens interessa comprendre amb l'objectiu d'extreure'n coneixement. Els models s'utilitzen en totes les disciplines científiques i en enginyeria i, per tant, també en informàtica.

Des d'una perspectiva d'enginyeria del programari, un model conceptual de dades (o simplement, un model conceptual) constitueix una descripció d'una part del món real (o domini d'aplicació) que ens interessa analitzar i conceptualitzar.

Per construir models conceptuals se solen usar models que es basen en llenguatges gràfics, atès que s'orienten a persones. La seva construcció sorgeix com a conseqüència d'un procés d'abstracció a partir de l'observació del món real que ens interessa modelar. Com a exemples de models que ens permeten realitzar aquesta tasca, tenim els diagrames de classes d'UML i el model *entity-relationship* (o model entitat-interrelació, també conegut simplement com a model ER). UML es basa en el paradigma d'orientació a objectes i és més expressiu que el model ER. Amb els diagrames de classes d'UML podem representar classes d'objectes del món real que tenen propietats comunes i diferents tipus de relacions entre classes (per exemple, relacions d'herència, associacions i diversos tipus de relacions part-tot). A la figura 12 podem observar diferents models de dades, classificats segons el seu nivell d'abstracció i amb la indicació, per a cadascun d'ells, de si està més orientat a l'usuari o a la implementació final.

Figura 12. Diferents models de dades classificades segons el seu nivell d'abstracció



Una característica fonamental dels models conceptuals és que s'aïllen de possibles tecnologies de representació. Malgrat això, un model conceptual ha de ser finalment implementat, amb l'objectiu que pugui ser utilitzat pels nostres programes d'aplicació. Quan la tecnologia d'implementació és una base de dades, el model conceptual s'ha de transformar d'acord al model de dades en el qual es basa la base de dades triada.

Des de la perspectiva de bases de dades, un model de dades constitueix el conjunt d'elements que ens proporciona el sistema gestor de la base de dades per a implementar (o representar) el nostre model conceptual, i inclou (almenys des d'una perspectiva clàssica) els elements següents:

1) estructures de dades,

2) operacions per a consultar i modificar les dades,

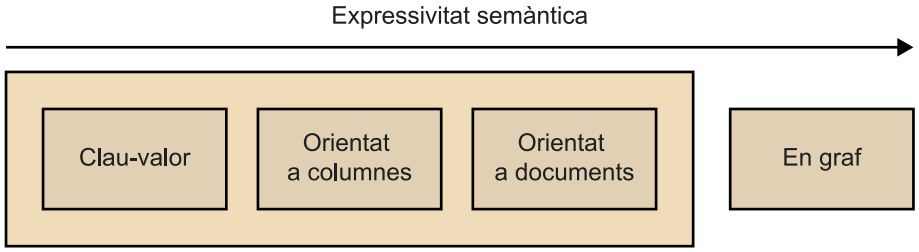
3) mecanismes per a definir restriccions d'integritat que les dades han de complir.

A manera d'exemple, el model relacional inclou la relació com a element d'estructuració, llenguatges com l'àlgebra relacional i SQL per a consultar i modificar les dades i la possibilitat de definir restriccions d'integritat (per exemple, claus primàries i foranes).

El model relacional no és l'únic model de dades, també existeixen els models prerelacionals (basats en els models jeràrquics i en xarxa) i els models de dades NoSQL. Una característica comuna a tots aquests models és que estan més propers (pel que fa al model relacional) al món de les representacions informàtiques.

Sota el paraigua NoSQL hi ha principalment dues famílies de models de dades, el model de grafs i els models d'agregació (figura 13). El model de grafs s'orienta a representar dominis d'aplicació on s'estableixen múltiples i complexes interrelacions entre els objectes del món real que desitgem representar. Per la seva banda, els models d'agregació inclouen tres models: el clau-valor, el documental i l'orientat a columnes. En conseqüència, tenim quatre models de dades que s'acostumen a utilitzar per a classificar les diferents implementacions de bases de dades NoSQL disponibles al mercat.

Figura 13. Models de dades NoSQL més comuns



5.2.1. Models de dades NoSQL d'agregació

Els models d'agregació es basen en el concepte d'agregat. Un agregat és una col·lecció d'objectes del món real relacionats que desitgem tractar com una unitat independent (o atòmica) des d'un punt de vista de significat a l'efecte dels punts següents:

1) Accés i manipulació: la unitat mínima d'intercanvi de dades entre els nostres programes i el sistema gestor de la base de dades és l'agregat.

2) Consistència i control de concurrència: assegurar que la definitivitat dels canvis sobre l'agregat i el manteniment de la seva integritat se circumscriuen a l'àmbit de l'agregat. Si és necessari mantenir restriccions d'integritat entre diferents agregats, la responsabilitat d'assegurar la consistència serà dels programes i no de la base de dades.

3) Com a unitat de distribució i replicació: si la base de dades està distribuïda, l'agregat s'usa com a unitat de distribució i replicació de les dades.

Si considerem els tres elements que des d'un punt de vista teòric tot model de dades ha de proveir, és important destacar que cada model d'agregació ofereix estructures (en alguns casos, mínimes) per a emmagatzemar les dades. En relació amb les operacions per a manipular i consultar les dades, cada fabricant ofereix, en general, el seu propi llenguatge (recordem la falta d'estandardització de les tecnologies NoSQL). Finalment, els models d'agregació deleguen el manteniment de restriccions d'integritat principalment als programes que accedeixen a la base de dades.

A continuació, detallarem cadascun dels models d'agregació.

Models d'agregació en clau-valor

El model clau-valor té el seu origen en la base de dades BerkeleyDB, que a dia d'avui és propietat d'Oracle. Altres exemples de bases de dades NoSQL basades en aquest model són DynamoDB, Redis i Riak.

Es tracta del model d'agregació més simple, atès que l'agregat constitueix un parell (clau, valor), on el valor conté les dades de l'agregat i la clau permet

identificar-lo unívocament; per tant, és la que s'utilitza per a identificar i consultar els agregats. El sistema gestor de la base de dades desconeix l'estructura interna associada a l'agregat (l'element valor). Això no significa necessàriament que l'agregat no tingui estructura, sinó que solament serà compresa pels programes que manipulen els agregats.

Com que l'agregat és opac per l'SGBD, les consultes hauran de realitzar-se per clau. És convenient que la clau de cada agregat tingui un significat dins del domini d'interès que es modelarà. Aquest seria el cas, per exemple, de claus com a noms d'usuari, adreces de correu electrònic, coordenades cartesianes per a l'emmagatzematge de dades de geolocalització, etc. Assignar claus aleatòries o que puguin oblidar-se fàcilment pot ser problemàtic i dificulta enormement l'accés a dades prèviament emmagatzemades.

Aquest conjunt de característiques base està representada amb diferent profunditat en les diferents bases de dades NoSQL de tipus clau-valor. Per exemple, Riak i Redis permeten definir una estructuració mínima de l'agregat i representar explícitament relacions entre agregats. Recordem que, en NoSQL, les bases de dades solen ser aproximacions als models de dades. Això significa que poden tant no implementar-los íntegrament, com afegir funcionalitats extra. Aquest fet pot provocar que algunes bases de dades NoSQL es puguin classificar de diferents maneres, segons la font d'informació consultada.

Models d'agregació orientats a documents

Aquest model es considera un cas particular del model clau-valor, però a diferència del model clau-valor, en el model documental els agregats (que reben el nom de document) tenen una estructura interna. Aquesta estructura interna simplifica el desenvolupament d'aplicacions, però redueix la flexibilitat del model clau-valor.

L'estructuració interna pot ser aprofitada pel sistema gestor de la base de dades i pels llenguatges que ofereixen aquestes bases de dades. Així, per exemple, els documents es poden recuperar mitjançant la seva clau o mitjançant el valor que prenen els seus atributs. També és possible accedir a parts del document i crear índexs que ajudin a recuperar eficientment els documents emmagatzemats en la base de dades. Els documents es poden agrupar en col·leccions.

Encara que els documents puguin tenir una estructura interna, a diferència del model relacional, no és necessari definir-la per endavant, sinó que serà implícita i dependrà de com estan estructurades les dades en els documents. En conseqüència, diferents documents que representin el mateix concepte del món real (per exemple, les dades de dues persones) poden tenir estructures totalment diferents o variacions de la mateixa estructura.

La majoria de bases de dades basades en el model documental es caracteritzen per emmagatzemar documents en format JSON, però també s'admeten altres formats, com seria el cas d'XML.

Algunes de les bases de dades NoSQL més conegudes que implementen el model documental són MongoDB, CouchDB, Marklogic i RethinkDB.

Models d'agregació orientats a columnes

Els models d'agregació orientats a columnes estan basats en el model de dades BigTable de Google. Malgrat això, avui dia, les bases de dades adherides a aquest model han evolucionat i poden presentar diferències significatives amb el model de dades motivador. Les bases de dades que segueixen aquest model també reben el nom de magatzems per a grans dades (*big data stores*) i magatzems de registres extensibles (*extensible record stores*).

Conceptualment, podem veure aquest model com a bidimensional (una matriu), en què cada fila de la taula representa un agregat i és accessible a partir d'una clau. Fins ara no hi ha cap novetat respecte als models d'agregació vists anteriorment. No obstant això, en aquest model, les dades dels agregats (és a dir, de cadascuna de les files de la taula) s'organitzen en columnes.

Per tant, un agregat és un conjunt de columnes, en què cada columna està formada per una tripleta composta pel nom de la columna, el valor de la columna i una marca de temps (*timestamp*) que indica quan es va afegir la columna a la base de dades.

Diferents columnes poden agrupar-se en una nova estructura, anomenada normalment família de columnes. Una família de columnes té un nom i una semàntica molt definida. Habitualment, dins d'una agregació, una família de columnes representa un concepte de l'agregació. Per exemple, entenent un estudiant com un agregat, tres de les seves famílies de columnes podrien ser, respectivament, les seves dades personals (sexe, data de naixement, nom), el seu domicili (carrer, codi postal, municipi, província i país) i els estudis previs realitzats per l'estudiant (assignatures cursades prèviament, resultats, any de superació, universitat on es van cursar, etc.).

Una característica d'aquest model és la facilitat que proveeix per a accedir a un subconjunt dels atributs d'un agregat (recordem que les dades estan organitzades per columnes). Això permet, per exemple, recuperar dades de manera eficient, solament amb els atributs rellevants a cada moment. Una altra característica d'aquest model és que, en comparació amb el model relacional, no totes la files d'una mateixa taula han de tenir el mateix conjunt de columnes. El model per columnes pot ser una mica confús al principi, però és també molt versàtil i ofereix moltes possibilitats.

Exemples de bases de dades que segueixen el model d'agregació orientat a columnes són Cassandra, HBase i Amazon SimpleDB.

5.2.2. Models de dades NoSQL en graf

El model en graf difereix totalment dels models d'agregació. En aquest model les dades no s'emmagatzemen mitjançant agregats, sinó mitjançant grafs.

Com a la resta de models de dades NoSQL, no hi ha un model en graf estàndard. En el nostre cas suposarem que el model en graf respon a un graf de propietats etiquetat. Sota aquesta assumpció, els models en graf estan compostos de nodes, arestes, etiquetes i propietats:

Què és un graf?

Un graf és una representació abstracta d'un conjunt d'objectes. Els objectes dels grafs es representen mitjançant vèrtexs (també anomenats nodes) i arestes. Els tipus, les propietats i els algorismes sobre grafs s'estudien en una branca de les matemàtiques anomenada matemàtica discreta.

Els **nodes** o **vèrtexs** són elements que permeten representar conceptes generals o objectes del món real. Vindrien a ser l'equivalent a les relacions en el model relacional. No obstant això, hi ha una diferència important: en el model relacional les relacions permeten representar conceptes («actor», per exemple) i les seves files permeten representar instàncies dels mateixos, és a dir, objectes del món real («Groucho Marx», per exemple). Per tant, els conceptes i els objectes del món real es representen mitjançant construccions diferents. No obstant això, en el model en graf, els conceptes i els objectes del món real es poden representar de la mateixa forma: mitjançant nodes. És possible que no hi hagi una diferenciació semàntica a nivell de model.

Les **arestes** o **arcs** són relacions dirigides que ens permeten relacionar nodes. Les arestes representen relacions entre objectes del món real i serien l'equivalent a les claus foranes en el model relacional o a les associacions en els diagrames de classes d'UML.

Les **etiquetes** són cadenes de text que es poden assignar als nodes i a les arestes per facilitar la seva lectura i proveir més semàntica.

Les **propietats** són parelles <clau, valor> que s'assignen tant a nodes com a arestes. La clau és una cadena de caràcters, mentre que el valor pot respondre a un conjunt de tipus de dades predefinides. Les propietats serien l'equivalent als atributs en el model relacional.

Tot i que els nodes poden emprar-se per a representar conceptes i objectes del món real, alguns sistemes gestors de bases de dades NoSQL en graf, com per exemple Neo4J, utilitzen les etiquetes per a identificar els conceptes als quals pertanyen els objectes del domini. Així doncs, si tenim el node que representa l'assignatura d'aquest curs, aquest node pot etiquetar-se amb l'etiqueta «Assignatura» per indicar que el node és de tipus assignatura. En aquests materials s'utilitzarà aquesta estratègia per a representar els conceptes de la base de dades.

La característica principal del model en graf és que les relacions estan explícitament representades en la base de dades. Això simplifica la recuperació d'elements relacionats de forma molt eficient. Aquesta eficiència és major que en models de dades relacionals, on les relacions entre dades estan representades de forma implícita (claus foranes) i requereixen executar operacions de combinació (en anglès *join*) per calcular-les.

A més, la definició de relacions és més rica en el model en graf, a causa que poden assignar-se propietats directament a les arestes. Això permet que les relacions entre dades continguin propietats i que proveeixin una representació més natural, clara i eficient.

El model en graf imposa poques restriccions d'integritat. Bàsicament dues, i la segona és conseqüència de la primera:

1) No és possible definir arestes (arcs) sense nodes origen i/o destinació.

2) Els nodes només poden eliminar-se quan queden orfes. És a dir, quan no hi hagi cap aresta que hi entri o en surti.

Igual que els altres models de dades NoSQL vistos, aquest model és *schemaless*. Per tant, no és necessari definir un esquema abans de començar a treballar amb una base de dades en graf. No obstant això, en aquest cas, hi ha part de l'esquema implícitament definit en els grafs. Per exemple, els arcs que representin el mateix tipus de relació acostumen a utilitzar una etiqueta comuna per a indicar el seu tipus i semàntica (igual que els nodes). Com a conseqüència de la característica anterior, aquest model permet afegir nous tipus de nodes i arestes fàcilment, i sense realitzar canvis en l'esquema. Tot i que és convenient fer-ho de manera ordenada i planificada, per evitar afegir tipus de relacions o nodes redundants o mal etiquetats.

Els sistemes de gestió de bases de dades que implementen un model en graf acostumen a proporcionar llenguatges de més alt nivell que els basats en models d'agregació. Per exemple, podem trobar l'ús de Cypher* en Neo4J, que és un llenguatge declaratiu semblant a SQL, i Gremlin.**

Exemples de bases de dades NoSQL basades en aquest model són Neo4j, OrientDB, AllegroGraph i TITAN.

*** http://bit.ly/2ElTZY1**
**** http://bit.ly/1pGhRK6**

Cypher

Cypher és un llenguatge de consulta i manipulació de grafs creat suportat per Neo4j.

Gremlin

Gremlin és un llenguatge de domini de gestió de grafs suportat per diversos sistemes gestors de bases de dades en graf.

5.2.3. Exemple pràctic

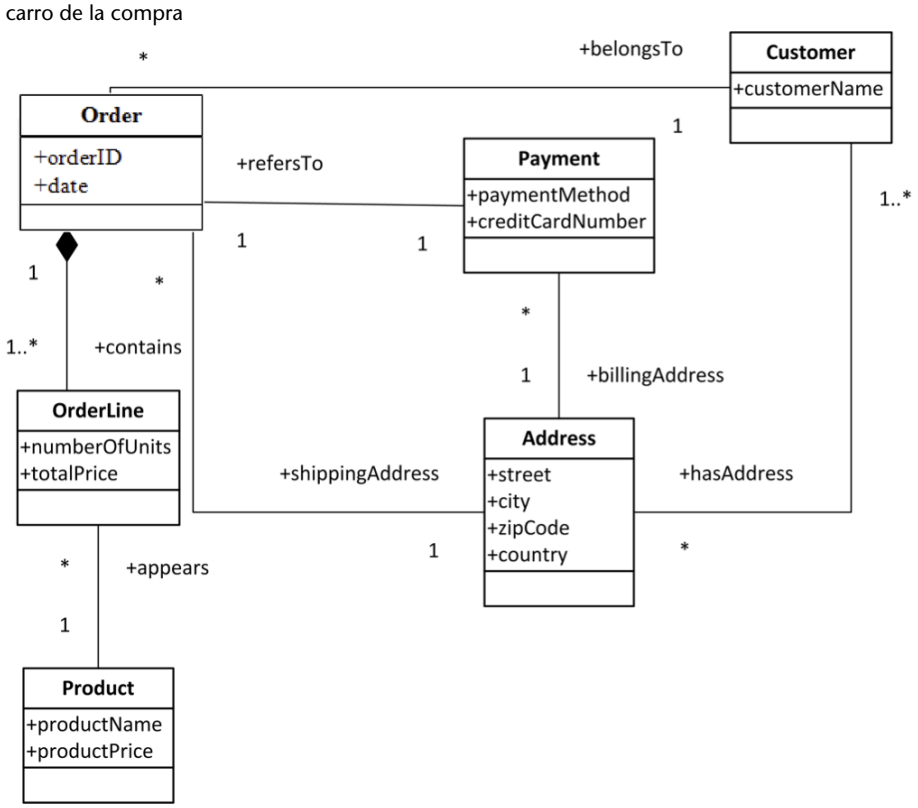
A continuació, presentarem un model conceptual i explicarem com representar-lo en els diferents models NoSQL presentats. L'objectiu és mostrar, mitjançant un exemple, com es distribueixen les dades en els diferents tipus de bases de dades NoSQL.

L'exemple que us proposem constitueix una simplificació del carro de la compra que s'utilitza en aplicacions d'*e-commerce* (figura 14).

En primer lloc, tenim clients (*customer*). Dels clients podem tenir registrades entre zero i múltiples adreces postals (*address*). Els clients també poden haver efectuat entre zero i múltiples comandes (*order*). Guardem el nom de tots els clients. Que un client no tingui comandes (o adreça postal) es pot relacionar, per exemple, amb el fet que el client alguna vegada s'hagi registrat al sistema, però que, pels motius que sigui, no ha completat cap procés de compra.

Les adreces queden caracteritzades pel carrer, ciutat, codi postal i país. No es guarden adreces que no pertanyin a cap client i podem tenir clients que comparteixin domicili. Una comanda es realitza en una data. Les comandes tenen una adreça postal d'enviament i dades associades per a efectuar el pagament (*payment*). Al seu torn, cada pagament té una adreça associada per a l'enviament de la factura i dades sobre com s'efectuarà el pagament (tipus i nombre de targeta).

Figura 14. Esquema conceptual en UML que indica les dades necessàries per a gestionar un carro de la compra

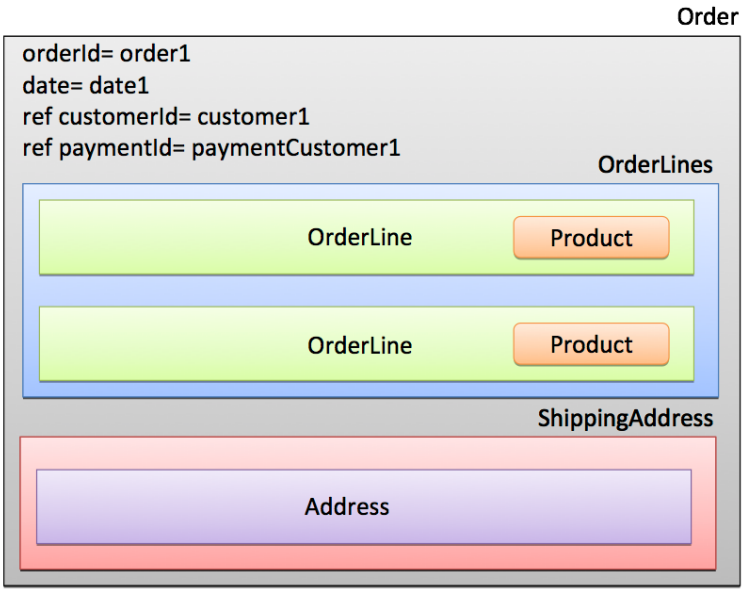


Cada comanda és un objecte compost que inclou, almenys, una línia de comanda (*orderline*). El fet que sigui un objecte compost s'indica mitjançant el diamant negre que a UML representa una relació de composició. Aquesta relació és d'un tipus de relació part-tot, la semàntica del qual implica que les línies de la comanda no tenen sentit sense l'objecte en el qual s'agreguen (la comanda). En altres paraules, cada línia de comanda solament pot formar part d'una comanda i, si la comanda s'elimina, també s'eliminaran les seves línies. Cada línia de comanda fa referència a un producte (*product*). També inclou quantes unitats s'adquireixen del producte i el preu total. Dels productes, se'n guarda el nom i preu unitari. Hi ha productes que no apareixen en cap línia de comanda (i, en conseqüència, tampoc en cap comanda), per exemple, perquè o no han estat adquirits per ningú o perquè encara no s'han posat en oferta.

Imaginem que sobre aquest model conceptual volem conèixer la informació relativa a una comanda concreta a l'efecte de gestionar el seu enviament. Aquesta informació serà més fàcil o difícil de recuperar en funció del tipus de base de dades triada per a implementar aquest model conceptual.

Per recuperar la informació requerida, seria possible dissenyar un agregat que incorpori totes les dades necessàries (figura 15). Aquest agregat (que representa l'objecte compost demanat) inclou els atributs propis de la comanda (el seu identificador i la data en la qual es realitza) i les referències als agregats que representen les dades de pagament i del client que efectua la comanda (aquestes referències ens permetrien navegar a aquests agregats). Addicionalment, cada comanda agrega tota la informació sobre les línies de comanda (i els productes als quals fan referència), a més de l'adreça postal d'enviament.

Figura 15. Disseny del carro de la compra mitjançant agregats



Atès que l'agregat és la unitat mínima d'accés i manipulació, obtindríem tota la informació desitjada de forma eficient mitjançant una sola operació. Pel que fa a una base de dades relacional, ens estem estalviant les operacions de combinació. Això és a causa del fet que un model d'agregació es basa en l'aplicació de tècniques de desnormalització.

Si la base de dades estigués distribuïda, l'agregat s'hauria utilitzat com a unitat de distribució de les dades, garantint d'aquesta manera que estaria emmagatzemat de manera conjunta en alguna de les bases de dades que conformen el sistema distribuït.

El disseny base de l'agregat pot ser el mateix per a les representacions dels models clau-valor, documental i de columnes. Malgrat això, haurà d'adaptar-se en cada cas, com es mostra a continuació.

Representació mitjançant un model de tipus clau-valor

En el cas d'escollir un model clau-valor utilitzarem l'agregat proposat com a valor i escollirem l'atribut identificador de comanda com a atribut clau (figura 16), ja que és un atribut que identifica de manera unívoca les comandes i és conegut. Si algú l'oblida, simplement hauria de recuperar informació —escrita o en qualsevol format— sobre la comanda per recuperar-la.

Figura 16. Disseny del carro de la compra utilizant un model clau-valor

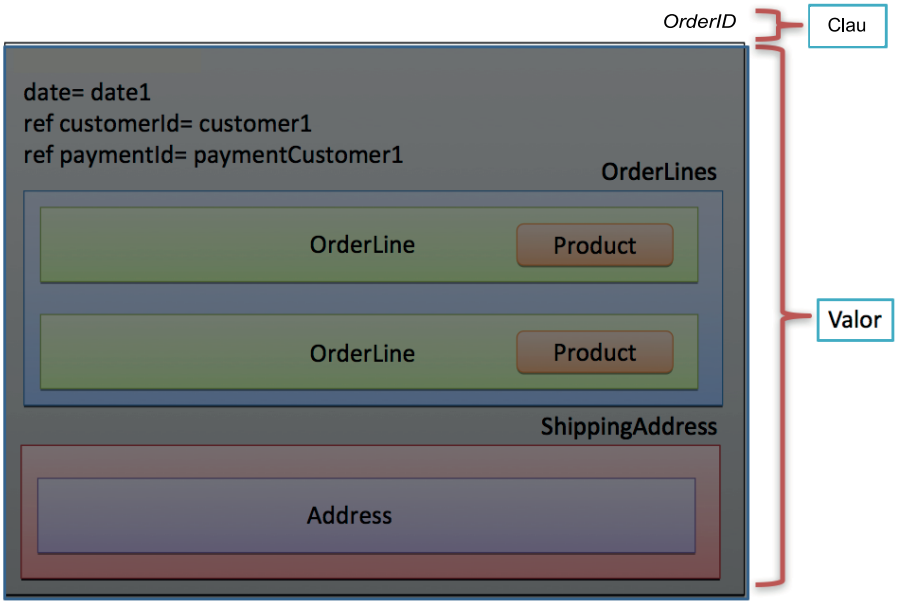


Figura 16

El contingut de l'agregat (tot i tenir una estructura) no serà visible per al sistema gestor de la base de dades, com s'indica mitjançant una capa negre.

A l'efecte de la base de dades, el valor de l'agregat serà opac. És a dir, la base de dades no serà conscient de quines dades conté l'agregat ni de com estan estructurades. Per tant, serà l'aplicació que consulta la base de dades qui haurà d'interpretar l'agregat i donar-li forma. Per a això, l'aplicació haurà de tenir informació sobre quines dades estan emmagatzemades en l'agregat i amb quina estructura. Amb aquesta informació, l'aplicació serà capaç d'interpretar i treballar amb les dades emmagatzemades. El fet que el valor de l'agregat sigui opac per a la base de dades proporciona més flexibilitat per a emmagatzemar les dades, ja que es podria, si fos convenient, emmagatzemar documents binaris (el PDF d'una factura, per exemple) com a valor d'un agregat per a alguns registres del sistema.

Noteu que en un model clau-valor pur, si es volen definir relacions entre diferents agregats ha de fer-se implícitament. És a dir, afegir dins de l'agregat les claus que referencien els agregats relacionats. Per explotar aquestes relacions, els programes que consultin l'agregat hauran d'interpretar les seves dades, extreure les claus dels agregats relacionats i consultar aquests agregats en la base de dades mitjançant les noves claus. Un procés poc eficient que desaconsella l'ús de models clau-valor quan el nombre de relacions en les dades sigui elevat.

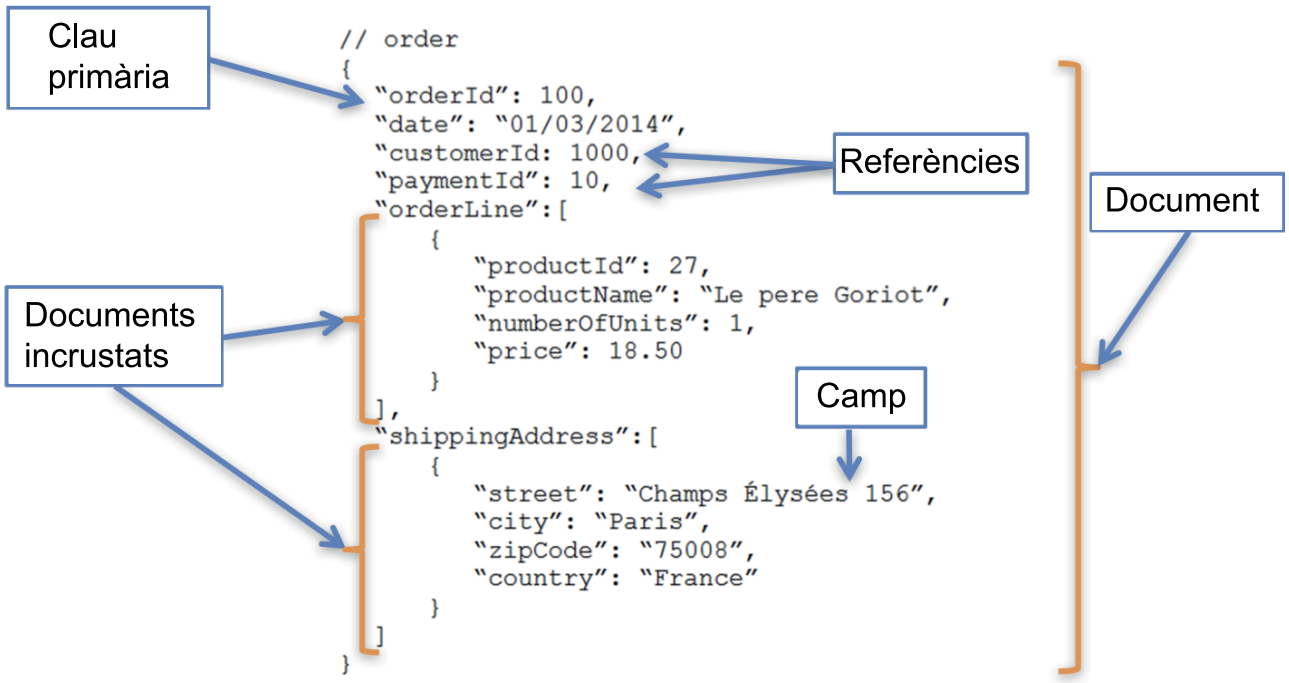
Representació mitjançant un model de tipus documental

En el cas d'utilitzar un model d'agregació documental, les diferents comandes s'emmagatzemarien en una col·lecció, que en el nostre cas hem denominat *order*.

Cada comanda estarà representada per un document en format JSON (figura 17). El document d'exemple conté una clau primària (*_id*) amb valor *order1*, un camp que permet indicar la data en què s'efectua la comanda i un parell de referències que apunten a altres documents (agregats) que contenen in-

formació del client que va realitzar la comanda i de les dades del pagament. Aquestes referències podrien haver-se definit d'una altra manera, ja que podria haver inclòs tota la informació del client i de les dades de pagament dins del document actual. No obstant això, s'ha preferit utilitzar referències i gestionar les dades de clients i de pagament mitjançant documents apart per exemplificar aquesta opció. A la comanda també podem veure exemples de (sub)documents incrustats com, per exemple, les línies de la comanda i l'adreça d'enviament de la comanda.

Figura 17. Disseny del carro de la compra utilitzant un model documental



Noteu que el fet d'utilitzar referències a altres agregats i no incloure totes les dades rellevants dins de l'agregat pot ralentir les consultes sobre l'agregat. No obstant això, fer-ho pot ser convenient en el cas que les dades dels agregats referenciats apareguin en molts agregats i es modifiquin amb assiduïtat, ja que així es redueix el nombre d'actualitzacions que s'han de realitzar cada vegada que es modifica el seu valor.

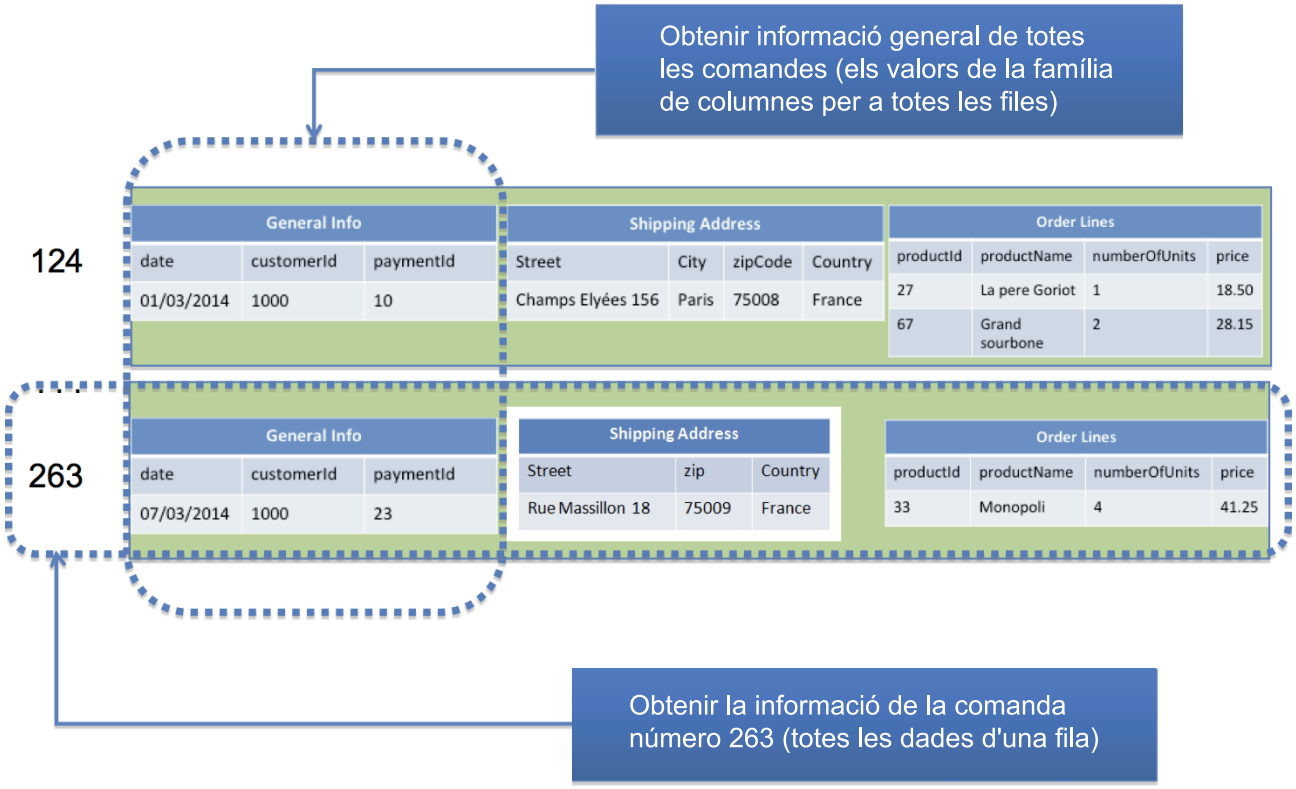
Representació mitjançant un model orientat a columnes

En aquest cas, igual que en els anteriors, s'ha triat el camp *orderId* com a identificador d'agregat. Posteriorment, s'han agrupat les columnes en diferents famílies sobre la base del seu significat. Cada família de columnes té un nom que la identifica. S'han creat tres famílies de columnes, una per a representar la informació general de la comanda (composta per les columnes data, identificador de client i identificador de pagament), una altra per a representar l'adreça de lliurament (composta per les columnes carrer, ciutat, codi postal i país) i l'última per a representar les línies de comanda, compostes pels iden-

tificadors de producte, els noms de productes, el nombre d'unitats i el preu final.

A la figura 18 podem veure els agregats de dues comandes representades mitjançant l'esquema proposat. Els dos agregats de l'exemple responen a dues comandes d'un mateix client.

Figura 18. Disseny del carro de la compra mitjançant un model orientat a columnes



Les columnes representen els atributs de la comanda, com poden ser la data, el client, o el domicili on s'ha enviat. Podem veure que el nombre de columnes dels agregats pot ser diferent. El primer agregat té una columna ciutat (*city*), mentre que el segon agregat no la té. Una altra particularitat és que les columnes poden tenir diferent nombre de valors en diferents agregats, tal com podem veure al primer agregat, que té dos valors per a la columna *productId*, mentre que el segon agregat només en té una. Això també passa a les famílies de columnes. Noteu que algunes famílies de columnes tenen només un valor (informació general i adreça de lliurament), mentre que d'altres poden emmagatzemar múltiples valors (línies de producte).

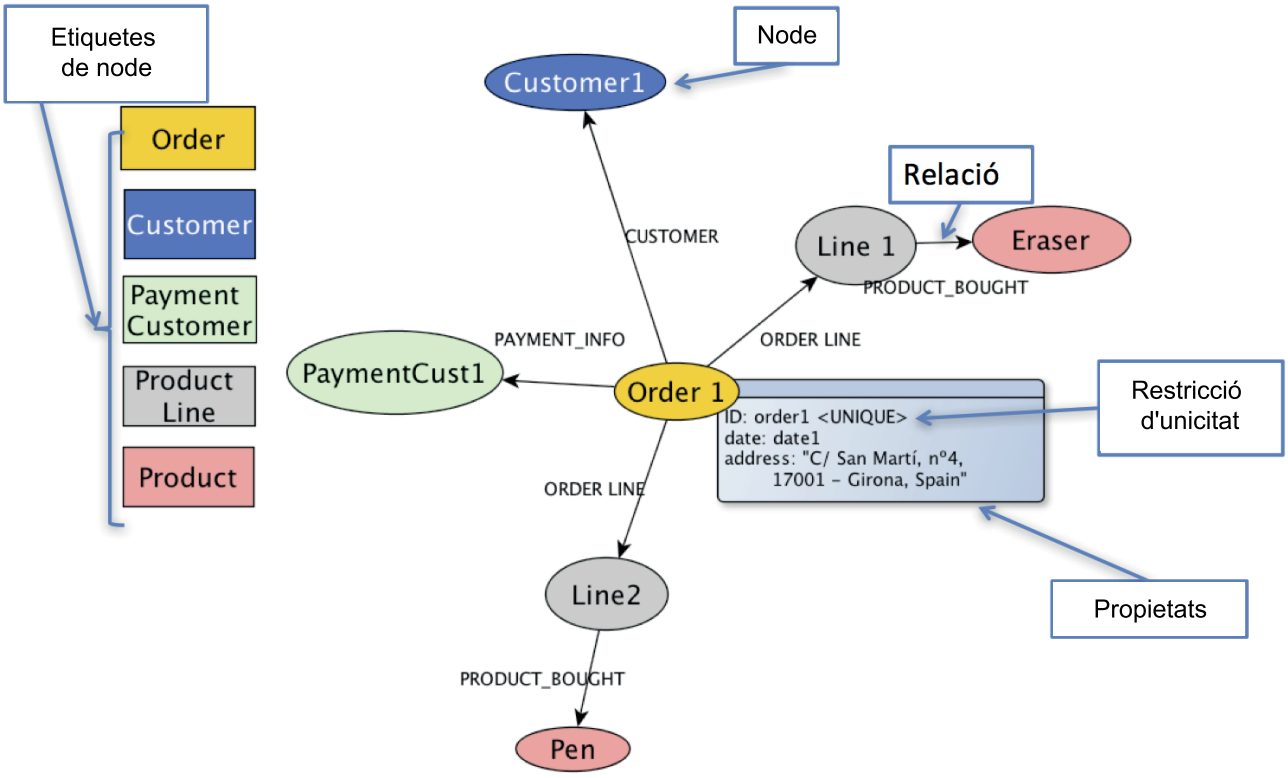
En un model d'agregació de columnes es pot accedir a una fila (o agregat) concret a partir de la seva clau (per exemple, es podria accedir a l'agregat del segon demanat a partir del seu identificador, amb valor 263). D'altra banda, també és possible obtenir informació sobre una família de columnes per a tots els agregats (per exemple, es podrien obtenir els valors de la família de columnes *general info* per a tots els agregats. Evidentment, també pot obtenir-se informació mitjançant la combinació de les dues operacions anteriors.

Representació mitjançant un model en graf

La representació de les comandes utilitzant un model en graf diferirà molt de les representacions d'agregats vistes fins ara, ja que l'objectiu d'aquest model no és representar les dades de manera agregada, sinó de manera dispersa però relacionada.

A la figura 19 podem veure una possible representació de les comandes mitjançant una base de dades NoSQL en graf. Les comandes s'emmagatzemen com a nodes i cada comanda constitueix un node diferent. En aquest cas, podem veure com el node *order 1* determina la comanda que hem estat utilitzant a l'exemple. Vegeu també que s'ha utilitzat una etiqueta anomenada *order* per a indicar que el node *order 1* és de tipus demanat. Una vegada definit el node de la comanda i el seu tipus, es poden indicar les seves dades mitjançant propietats. A l'exemple podem veure que s'han definit tres propietats: l'identificador de la comanda (*ID*), la data de la comanda (*date*) i l'adreça d'enviament (*address*). A més, en aquest cas, hem decidit definir a la propietat ID una restricció d'unicitat; per tant, no es permetran dues propietats *ID* amb el mateix valor en nodes de tipus *order*. Les línies de comanda s'han representat mitjançant un conjunt de nodes i relacions, però també podrien haver-se representat mitjançant propietats. Expressar-ho d'una manera o una altra constitueix una decisió de disseny de la base de dades i haurà d'escollir-se en funció de l'ús esperat de la base de dades.

Figura 19. Disseny del carro de la compra mitjançant un model en graf



El client que va realitzar la comanda s'ha definit mitjançant un node *customer 1* de tipus *customer*. Per indicar el fet que *customer 1* és el client que va realitzar la comanda, s'ha creat una relació amb nom *customer* entre els nodes *order 1* i *customer 1*. De manera semblant s'ha creat un node per a les dades de pagament anomenat *paymentCust 1* i una relació per a indicar que aquestes dades de pagament són relatives a la comanda *order 1*. Noteu que la nomenclatura utilitzada per als nodes i les relacions són diferents: els primers s'indiquen amb la primera lletra en majúscula i els segons amb totes les lletres en majúscula.*

*** La nomenclatura utilitzada en aquest exemple és la que fa servir el sistema gestor de bases de dades Neo4j.**

Respecte a les línies de comanda, s'ha creat un tipus de node anomenat *product line* i s'han definit dos nodes d'aquest tipus per a les línies de comanda: *line 1* i *line 2*. Aquestes línies s'han relacionat amb la comanda mitjançant dues relacions, ambdues anomenades *order line*. Fixeu-vos en el fet que encara que diferents nodes del mateix tipus continguin noms diferents, les relacions comparteixen nom. Això és perquè el nom de la relació s'utilitza a Neo4j per a identificar-ne el tipus. És a dir, com que les relacions entre *order 1* i *line 1* i entre *order 1* i *line 2* tenen el mateix nom (aquest nom és *order line*), això significa que són del mateix tipus (*order line*). Això ens resultarà d'utilitat més endavant a l'hora de consultar el graf, ja que ens permetrà realitzar consultes que satisfacin un patró, com, per exemple: «Dona'm totes les línies de comanda del producte Eraser». El patró que hi ha darrere d'aquesta consulta seria: «Els nodes de tipus *product line* relacionats mitjançant una relació *order line* amb un node el nom del qual és *Eraser*». Els nodes del graf que verifiquin aquest patró satisfaran la consulta anterior.

Els productes que apareixen en cada línia de comanda s'han definit mitjançant un nou tipus de node denominat *product*. En particular, s'han definit dos nodes d'aquest tipus: *Eraser* i *Pen*. Per indicar que aquests nodes apareixen en una línia de comanda s'han creat dues relacions de *product bought* que els relacionen amb les línies de comanda en què apareixen.

En aquest exemple, hem creat una versió simplificada del possible graf resultant, obviant alguns nodes i propietats que podrien haver-se afegit. Tot i estar incomplet, el nombre de nodes i relacions creats en l'exemple és elevat. En un cas real el nombre d'elements del graf creix ràpidament. Per aquest motiu, serà necessari fer un bon disseny del graf per a identificar quins conceptes han de ser representats mitjançant nodes, quins mitjançant relacions i quins mitjançant propietats. Així mateix, és important fer una bona gestió d'índexs per optimitzar la consulta de les dades a partir del valor de les seves propietats.

5.3. Avantatges i inconvenients de les bases de dades NoSQL

A continuació, enumerem breument els avantatges i inconvenients de les bases de dades NoSQL respecte les bases de dades relacionals. Cal destacar que, donada la gran diversitat que existeix en les bases de dades NoSQL, és impossible enumerar avantatges i inconvenients que siguin certes en el 100% de totes elles. El que sigui vàlid per a algunes (d'agregació) pot no ser vàlid per

a d'altres (en graf). A més, cal tenir en compte que el moviment NoSQL va incorporant més tipus de dades amb característiques pròpies: *event oriented*, multimodel, etc.

Si parlem de bases de dades NoSQL d'agregació, els avantatges més rellevants són el suport de la distribució i replicació de dades de manera més natural, el tractament més eficient de grans volums de dades, més tolerància a fallades, la possibilitat d'escalar horitzontalment i de permetre emmagatzemar la informació de manera desnormalitzada (això també pot ser un inconvenient) —de manera que així també es permet tenir preparades les dades en la base de dades per al seu posterior consum—, la possibilitat d'oferir un esquema de dades més flexible, capacitats de corregir problemes sense la intervenció de l'administrador de la base de dades (caigudes de servidors, per exemple) i que, com que hi ha diferents tipus de models, permeten adaptar-se molt bé a diferents problemes.

A les bases de dades NoSQL en graf, els avantatges són clars: suporten més naturalment informació relacionada, permeten representar més eficientment relacions i consultar dades relacionades de forma més fàcil i eficient.

Els problemes principals de les bases de dades NoSQL són, potser, la falta d'estàndards als seus llenguatges de manipulació i consulta de dades, la seva ràpida i caòtica evolució, la dificultat de programar i mantenir aquests sistemes (conceptualment sembla que la flexibilitat d'esquema pot ser un veritable problema per a l'evolució dels programes, tot i que hem d'esperar un temps per veure fins a quin punt i com se soluciona), la falta de suport en alguns casos, la baixa consistència que ofereixen (bàsicament les d'agregació) i la reticència al canvi que poden trobar-se a les empreses. Un altre punt feble de les bases de dades NoSQL fonamental és la seguretat de les dades. D'una banda, perquè no són productes tan madurs com els seus homòlegs relacionals i, d'una altra, perquè com més distribució i replicació hi ha, més potencials forats de seguretat existeixen i més punts d'accés a les dades disponibles per a possibles atacants.

Un altre problema de les bases de dades deriva de la seva gran variabilitat i de l'elevat nombre de productes disponibles, actualment més de dos-cents. Això pot provocar dispersió, una evolució de productes més lenta, més dificultat per a trobar professionals qualificats per a cada producte, una corba d'aprenentatge més acusada, dificultats d'interoperabilitat i compatibilitat i una percepció d'immaduresa. Sembla coherent pensar que en el futur hauria d'haver-hi una racionalització dels productes NoSQL.

Hi ha alguns aspectes que apareixen a la bibliografia de NoSQL però que, sota el punt de vista de l'autor, no poden classificar-se actualment com a avantatges ni com a inconvenients. Són els aspectes següents:

- La falta de maduresa de la tecnologia NoSQL: actualment existeixen productes molt potents i madurs al mercat i que han estat utilitzats en casos

d'èxit impactants i clars. En són exemples Cassandra, MongoDB, Riak, DynamoDB o Neo4j. D'altra banda, a diferència d'anys enrere, actualment la majoria d'eines d'intel·ligència de negoci suporten sistemes NoSQL.

- L'estalvi a causa de la seva gratuïtat i disponibilitat de codi: el fet que la majoria sigui de codi lliure pot ser un avantatge, però també un inconvenient, i no garanteix que el cost d'implantació o manteniment sigui menor, ja que el cost total de propietat pot ser més elevat. A més, alguns dels sistemes gestors de bases de dades NoSQL tenen una versió professional de pagament que pot fer que la seva implantació sigui més costosa que la d'una base de dades relacional.

- Dificultat de relacionar dades (o realitzar *joins*): si es fa un bon disseny i s'utilitzen diferents models de bases de dades de manera combinada per a representar i manipular les dades d'una manera eficient, la qual cosa es diu persistència políglota,* les dades ja estaran preparades per ser consumides i això ho facilitarà.

*** Més informació al vídeo següent: https://vimeo.com/121661296**

Persistència políglota

Fa referència a l'estratègia d'utilitzar la base de dades més adequada per a cada problema. En problemes complexos, pot implicar utilitzar més d'una base de dades de manera coordinada.

Resum

En aquest mòdul didàctic hem presentat les tasques i arquitectures associades a la captura, el preprocessament i l'emmagatzematge de dades massives (*big data*).

A la primera part d'aquest mòdul hem discutit sobre els processos de captura i preprocessament de dades massives, fent especial èmfasi en el concepte de dades generades contínuament (*streaming data*) i com poden capturar-se, distribuir-se i emmagatzemar-se en entorns distribuïts.

En aquest sentit, s'ha aprofundit en l'enviament de dades en *streaming* des dels productors als seus potencials consumidors. De tots ells, s'ha posat especial focus en els sistemes de missatgeria, que ofereixen més escalabilitat i disponibilitat. Com a exemple d'aquests sistemes, s'ha mostrat el funcionament bàsic d'Apache Kafka.

A la segona i última part dels materials s'han tractat amb més detall els sistemes que permeten emmagatzemar dades massives. Concretament, hem vist els sistemes de fitxers distribuïts, la seva arquitectura i les seves funcionalitats bàsiques i ens hem centrat en el sistema de fitxers distribuïts de Hadoop (*Hadoop Distributed File System*, HDFS).

Hem finalitzat aquest mòdul introduint les bases de dades NoSQL. Concretament, hem revisat els diferents models existents i els principals avantatges i inconvenients de cadascun d'ells.

Glossari

checksum *f* Vegeu **suma de verificació**.

commodity maquinari *sust.* La computació de «baix cost» implica l'ús d'un gran nombre de components de computació ja disponibles per a la computació paral·lela amb la intenció d'obtenir la major quantitat de computació útil a baix cost.

Cypher *sust.* Llenguatge de consulta i manipulació de grafs creat suportat per Neo4j.

dades estructurades *m pl* Dades que segueixen un patró igual per a tots els elements i que, a més, és conegut *a priori*. Per exemple, les dades d'un full de càlcul presenten els mateixos atributs per a cada fila.

dades no estructurades *m pl* Dades que no segueixen cap tipus de patró conegut *a priori*. Per exemple, dos documents de text o imatges.

dades semiestructurades *m pl* Forma de dades que no conté una estructura fixa predefinida *a priori*, però que conté etiquetes o altres marcadors per a separar els elements semàntics i fer complir jerarquies de registres i camps de les dades. Per exemple, els documents JSON o HTML.

graf *m* Representació abstracta d'un conjunt d'objectes. Els objectes dels grafs es representen mitjançant vèrtexs (també anomenats nodes) i arestes. Els tipus, les propietats i els algorismes sobre grafs s'estudien en una branca de les matemàtiques denominada matemàtica discreta.

Gremlin *sust.* Llenguatge de domini de gestió de grafs que està suportat per diversos sistemes gestors de bases de dades en graf.

invocació a procediment remot *f* En computació distribuïda, programa que utilitza un ordinador per a executar codi en una altra màquina remota sense haver de preocupar-se per les comunicacions entre ambdues.
en Remote procedure call

metadada *m* Dada que descriu altres dades. En general, un grup de metadades es refereix a un grup de dades que descriuen el contingut informatiu d'un objecte que es denomina recurs.

persistència políglota *f* Es refereix a l'estratègia d'utilitzar la base de dades més adequada per a cada problema. En problemes complexos pot implicar utilitzar més d'una base de dades de manera coordinada.

remote procedure call *f* Vegeu **invocació a procediment remot**.
sigla **RPC**

suma de verificació *f* És una funció *hash* que té com a propòsit principal detectar canvis accidentals en una seqüència de dades per a protegir-ne la integritat, i que verifica que no hi hagi discrepàncies entre els valors obtinguts fent-ne una comprovació inicial i una altra de final després de la transmissió.
en checksum

temps UNIX Per a representar una data en temps UNIX s'utilitza un nombre natural. El valor d'aquest nombre indica els segons transcorreguts des de la mitjanit de l'1 de gener del 1970.

Bibliografia

Apache Kafka (2017). *Documentation*. [Data de consulta: setembre 2017]. <https://kafka.apache.org/documentation>

Atzeni, P; Jensen, C. S.; Orsin, G.; Ram, S. (2013). «The relational model is dead, SQL is dead, and I don't feel so good myself». *SIGMOD Record* (vol. 42, núm. 2, pàg. 64-68).

Bengfort, B.; Kim, J. (2016). *Data Analytics with Hadoop*. Boston: O'Reilly Media.

Canal Vimeo de bases de dades dels EIMT de la UOC [canal en línia]. UOC. <https://vimeo.com/channels/basesdedatos/videos>

Cattell, R. (2010). *Scalable SQL and NoSQL Data Stores* [document en línea]. [Data de consulta: juny 2017]. <http://cattell.net/datastores/Datastores.pdf>

Celko's, J. (2013). *Complete Guide to NoSQL*. Boston: Elsevier.

Dunning, T.; Friedman, I. (2015). *Real-World Hadoop*. Boston: O'Reilly Media.

Garofalakis, M.; Gehrke, J.; Rastogi, R. (2016). *Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications)*. Berlín: Springer.

Golab, L.; Özsu, M. T. (2003). «Issues in data stream management». *ACM Sigmod Record* (vol. 32, núm. 2, pàg. 5-14).

Kleppmann, M. (2016). *Making Sense of Stream Processing: The Philosophy Behind Apache Kafka and Scalable Stream Data Platforms*. Boston: O'Reilly Media.

Popescu, A. *NoSQL and Polyglot Persistence* [document en línia]. [Data de consulta: agost 2017]. <http://bit.ly/1ggTT4k>

Redmond, I.; Wilson, J. (2012). *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Raleigh, NC: The Pragmatic Bookshelf.

Robinson, I.; Webber J.; Eifren, I. (2013). *Graph Databases*. Boston: O'Reilly.

Sadalage, P. J.; Fowler, M. (2013). *NoSQL Distilled. A brief Guide to the Emerging World of Polyglot Persistence, Pearson Education* [document en línia]. [Data de consulta: agosto 2017]. <http://bit.ly/1koKhBZ>

Sitto, K.; Presser, M. (2015). *Field guide to Hadoop: an introduction to Hadoop, its ecosystem, and aligned technologies*. Boston: O'Reilly Media.

Stonebraker, M.; Çetintemel, O. (2005). «One Size fits all: An Idea whose time has come and gone». *Proceedings of the International Conference on Data Engineering* (pàg. 2-11).

White, Tom (2015). *Hadoop: The Definitive Guide, 4th Edition*. Boston: O'Reilly Media.

Zaharia, M.; Karau, H.; Konwinski, A.; Wendell, P. (2015). *Learning Spark: Lightning-Fast Big Data Analysis*. Boston: O'Reilly Media.