GESTIÓ DE DADES MASSIVES

Alberto Abelló[1] **i Besim Bilalli**[2]
1. *Professor agregat del Departament d'Enginyeria de Serveis i Sistemes d'Informació de la Universitat Politècnica de Catalunya. aabello@essi.upc.edu*
2. *Professor associat del Departament d'Enginyeria de Serveis i Sistemes d'Informació de la Universitat Politècnica de Catalunya. bbilalli@essi.upc.edu*

Resum: Aquest article pretén donar una visió general del que és la gestió de dades massives, la seva problemàtica i com s'han d'abordar les solucions. La principal dificultat és que no existeix una solució genèrica i s'ha de construir a mida de cada organització.

Paraules clau: ciència de dades, escalabilitat, distribució, paral·lelisme.

BIG DATA MANAGEMENT

Abstract: This article presents an overview of what big data management is, the problems it poses and how the solutions should be approached. In this respect, the main difficulty is that there is no generic solution so big data management processes must be tailored to each organization.

Keywords: data science, scalability, distribution, parallelism.

1. Introducció

Les dades han estat un objecte valuós durant molt de temps (principalment per a l'anàlisi i la visualització; vegeu, per exemple, les obres de Charles Joseph Minard al segle XIX (Friendly i Denis, 2001)), però avui en dia el seu valor s'ha multiplicat, perquè som capaços de generar-ne, emmagatzemar-ne i manipular-ne una quantitat immensa, gairebé il·limitada, que normalment es coneix com a *dades massives* (*big data*). Mitjançant les dades massives podem predir el proper huracà, la maniobra d'un conductor, el diagnòstic d'un pacient i els retards dels vols, entre d'altres. La indústria líder en l'anàlisi de dades és la de les telecomunicacions, que les usa massivament tant en l'optimització de la Xarxa com en la captació i retenció de clients (Carnelley i Schwenk, 2016), però les dades massives s'han convertit en un actiu potent i monetitzable i han transformat la nostra societat, en general, en la societat de la informació, fins al punt que no podem concebre la nostra existència sense les dades que es generen de forma continuada, de maneres molt diverses (per exemple, xarxes socials i tot tipus de sensors) i en múltiples formats (com ara, números, text, imatges i vídeos). L'any 2022, el 92,1 % de les grans empreses ja declaraven que estaven aconseguint beneficis de la seva inversió en dades massives i intel·ligència artificial (Davenport i Bean, 2022).

Efectivament, estem assistint a l'eclosió de nous models de negoci basats en l'anàlisi de dades massives, i veiem constantment notícies sorprenents de l'aplicació sobre elles d'aprenentatge automàtic i les més modernes tècniques estadístiques. Habitualment es coneix com a *ciència de dades* la conversió d'aquestes dades en coneixement directament aplicable a les nostres vides (incloent, entre altres tècniques, l'aprenentatge automàtic). No obstant això, malgrat l'aparent simplicitat del concepte, no és gens trivial i requereix especialistes i eines molt específiques no només de l'estadística i la matemàtica, sinó també d'àmbits tan diversos com la visualització, l'enginyeria del programari i, òbviament, la gestió i el processament de les mateixes dades.

En efecte, els grans resultats de l'aprenentatge automàtic no serien possibles sense la tecnologia que permet la gestió i el processament d'ingents quantitats de dades. La gran majoria d'investigadors i empreses consultores estimen que el 90 % de les dades actuals s'han generat en els darrers dos anys (Marr, 2018), i en molts àmbits la seva gestió i processament són tot un repte que porta la tecnologia al límit. Per exemple, un parc eòlic amb només 100 aerogeneradors genera aproximadament 17,5 terabytes de dades al dia. Emmagatzemar i processar aquesta magnitud de dades és tot un repte, però també un requisit indispensable per a la seva anàlisi posterior.

Conseqüentment, en aquest article ens fixarem especialment en aquest problema concret de la gestió de les dades. En aquest sentit, les dificultats de gestió que plantegen les dades massives s'acostumen a caracteritzar per 3V:

1) *Volum*, que fa referència a la quantitat de dades, en què hem passat de terabytes (TB, 10^{12}) a zettabytes (ZB, 10^{21}).

2) *Varietat*, que fa referència a la diversitat de tipus i formats de dades, en què, a més de dades estructurades, també estem recopilant dades no estructurades i semiestructurades.

3) *Velocitat*, que fa referència a la velocitat a la qual es reben les dades, en què essencialment estem considerant fluxos de generació contínua de desenes de milers de missatges per segon.

La gestió de dades massives tracta precisament de fer front de manera eficient a aquests tres reptes. Per abordar el problema del volum, una de les solucions més esteses és la distribució de dades i el processament paral·lel, normalment utilitzant tecnologies basades en el núvol. Per fer front a la varietat, cal automatitzar tant com sigui possible la integració i el processament de conjunts heterogenis i en constant evolució de múltiples i diverses fonts de dades. Finalment, per fer front a la velocitat, les irregularitats en el ritme d'arribada s'han d'aplanar fins a obtenir un flux de processament pràcticament constant mitjançant mecanismes de memòria intermèdia i les dades s'han de processar a partir d'algorismes de complexitat com a màxim lineal per poder proporcionar una anàlisi en temps real. Malauradament, avui dia no hi ha cap sistema únic que sigui capaç de fer front a totes aquestes dificultats i es requereix un alt grau d'expertesa i de coneixement multidisciplinari. Una mostra més d'aquesta realitat és el projecte ExtremeXP,[1] concedit recentment a la Universitat Politècnica de Catalunya i vint socis europeus més per a la creació d'un sistema que precisament faciliti i simplifiqui aquesta gestió de les dades per a la seva anàlisi posterior.[2]

2. La rellevància de la gestió de dades

Els projectes de ciència de dades per a l'anàlisi descriptiva, predictiva i prescriptiva de dades massives s'estan convertint en una norma i afecten tots els sectors econòmics i socials. Aquests projectes consisteixen en una sèrie de tasques de processament de les dades que podem identificar de manera genèrica en diferents fases d'adquisició (obtenció dels conjunts de dades), transformació (neteja i preparació de dades per adequar-les a les necessitats de l'algorisme d'entrenament), modelització (entrenament o creació d'un model estadístic), avaluació (quantificació de la bonança del model creat) i aplicació del model (utilització del model per resoldre un problema concret). Típicament, per motius de simplicitat i d'agilitat de desenvolupament, els científics de dades treballen en el seu ordinador de sobretaula i accedeixen a un cert conjunt de dades (sovint en format CSV[3] o JSON)[4] mitjançant el llenguatge de programació Python o R. Aquesta manera de treballar és perfectament acceptable per a una anàlisi preliminar i exploratòria, però clarament no serveix per posar en producció sistemes d'aprenentatge automàtic sobre dades massives.

Tal com s'explica a Sambasivan *et al*. (2021), un error molt comú és centrar-se a provar simplement diferents models fins a obtenir-ne un que s'ajusti tan bé com es pugui a les dades existents sense parar gaire atenció al seu processament (fet que es coneix com a *aproximació centrada en el model*). No obstant això, habitualment la diferència entre diferents models és mínima si les dades d'entrenament són exactament les mateixes. El que realment pot donar lloc a una millora substancial és centrar-nos en la millora de les dades d'entrenament i la seva qualitat (és a dir, una aproximació centrada en les dades). Les diferències entre les dues aproximacions es pot veure resumida en la taula 1. Malauradament, la segona opció és molt més complexa i costosa. Part de la complexitat i el cost ve del volum de dades i requereix treballar en un entorn distribuït i paral·lel en comptes de l'ordinador de sobretaula. L'altra part ve de la gestió dels múltiples fluxos de dades alternatius i complementaris que es puguin generar de manera sistemàtica i finalment passar de desenvolupament a producció amb totes les garanties d'un procés típic de creació de programari (*software*).

TAULA 1
Alternatives per a la millora de l'aprenentatge automàtic

Centrat en el model	*Centrat en les dades*
Utilitzar tantes dades com es pugui i provar molts models diferents.	Mantenir el model fix i provar d'ajustar les dades al problema.
Millorar el model de manera iterativa fins a eliminar el soroll de les dades.	Millorar la qualitat de les dades fins a millorar els resultats del model obtingut.
Bonança de l'ajust del model a les dades.	Bonança de la qualitat de les dades respecte a la realitat.

FONT: Elaboració pròpia a partir de les dades extretes de Sambasivan *et al.*, 2021.

Aquesta sistematització de la generació dels fluxos de dades ha d'ajudar a: 1) facilitar l'estandardització de les etapes utilitzades dins dels fluxos (per exemple, no oblidar la fase d'avaluació entre la de modelització i la seva aplicació); 2) evitar les etapes embullades entre si (com ara, utilitzant pràctiques de disseny modular); 3) gestionar les proves unitàries de les diferents etapes i la seva integració en el sistema en producció; 4) aplicar un marc o metodologia estàndard, que doni lloc a fluxos interoperables i reutilitzables, i 5) automatitzar (o almenys simplificar) el des-

1. «EXPerimentation driven and user eXPerience oriented analytics for eXtremely Precise outcomes and decisions».
2. *https://cordis.europa.eu/project/id/101093164*.
3. *Comma-separated values*, valors separats per comes.
4. *JavaScript object notation*, notació Javascript d'objectes.

plegament dels fluxos de dades un cop desenvolupats i testats adequadament. Tal com s'explica a Sculley *et al.* (2015), la ciència de dades és només una minúscula caixeta d'aprenentatge automàtic envoltada d'una immensa quantitat de mecanismes de gestió de dades en forma de canonades de programari (*pipelines*, en anglès) per on flueixen i es transformen les dades.

3. Necessitats de la gestió de dades massives

Tradicionalment, un sistema de gestió de bases de dades (SGBD) es presenta com un únic sistema complex capaç de servir diferents propòsits i proporcionar funcionalitats múltiples de manera integrada. En canvi, un sistema de gestió de dades massives està construït de forma modular en termes de diferents components de programari totalment independents que interactuen de diverses maneres per aconseguir un propòsit global. Aquesta arquitectura modular ve donada per les particularitats de les transformacions de dades de cada projecte i la manca de maduresa de les eines. Avui en dia, no existeix cap sistema al mercat capaç de processar un nombre indeterminat de fonts d'informació externes a l'organització, no estructurades, amb un flux constant de generació de dades i que, a més, sigui escalable. Les solucions exigeixen múltiples components arquitectònics específics i especialitzats que interactuen amb molts altres per superar els reptes de transformar les dades en coneixement útil per a l'organització.

No obstant això, les funcionalitats que esperem dels diferents mecanismes de gestió de dades al cap i a la fi corresponen a les que esperem d'un SGBD tradicional (amb èmfasi especial en la distribució de les dades i l'escalabilitat) i, per tant, de manera semblant a un SGBD distribuït, en un sistema de dades massives hem d'entendre, entre altres qüestions, com recull les dades, on les guarda, com es relacionen amb les metadades corresponents, com es processen de manera eficient, com es gestionen les rèpliques i se'n garanteix la consistència, etc. Analitzem ara algunes de les particularitats més rellevants d'aquests sistemes:

— *Paral·lelisme*. Amb l'objectiu de reduir el temps de resposta, proporcionant escalabilitat i alta disponibilitat, la idea és resoldre els reptes de dades massives dividint el problema en petites peces lògiques i tractar-les de manera independent en diferents processadors. Això implica que els conjunts de dades s'han de poder fragmentar en trossos independents que s'hauran de moure allà on tinguem recursos computacionals disponibles.

— *Computació al núvol*. Per motius econòmics, no en tenim prou amb l'escalabilitat, sinó que volem disposar d'elasticitat (activant o desactivant els recursos en funció de les necessitats concretes en un moment determinat), ubiqüitat, tolerància a fallades i la possibilitat de fer ús de maquinari de baix cost, però d'altes prestacions, amb una capacitat de processament i d'emmagatzemament suposadament infinita. La computació al núvol ens dona tot això de manera transparent per a l'usuari.

— *Multiarrendament*. La màgia del núvol s'aconsegueix amb la compartició de recursos a gran escala. Els usuaris no paguen la propietat del maquinari i el programari, sinó que paguen només pel seu ús en el moment que el necessitin. De fet, apareix el concepte *multiarrendament*, en què el proveïdor s'encarrega d'una part molt important de la gestió dels recursos que són compartits de manera senzilla, dinàmica i transparent per múltiples consumidors en forma de màquines virtuals.

— *Localització de dades*. Un cop les dades estan fragmentades, aquests fragments s'han d'assignar de manera transparent i dinàmica a les diferents màquines virtuals disponibles en cada moment. Per complicar-ho encara més i fer-ho com més eficient millor, el mateix fragment d'un conjunt de dades es pot col·locar en diverses màquines al núvol per tal de millorar-ne la fiabilitat (si una falla, encara es poden utilitzar les altres) i la localitat (oferint més possibilitats per executar les tasques).

— *Optimització de consultes*. Un cop tenim les dades fragmentades, distribuïdes i feta la rèplica en el núvol, n'hem de treure el màxim profit per al seu processament amb un temps de resposta mínim. Donada la complexitat del problema, no podem esperar que una persona pugui trobar la millor manera d'utilitzar els diferents recursos disponibles al núvol de forma coordinada per assolir aquest objectiu. És per això que existeixen optimitzadors de consultes que, aplicant heurístiques i treballant sota determinades hipòtesis, fan que aquesta tasca sigui factible en un temps raonable, maximitzant la paral·lelització de les tasques alhora que es minimitzen les comunicacions entre les màquines i l'efecte de les barreres de sincronisme, que fan que una tasca o procés hagi d'esperar fins que un altre acabi, i són el principal límit al paral·lelisme.

4. Eines de gestió de dades massives

Amb els nous reptes plantejats tant per les característiques de les dades com per la infraestructura necessària per gestionar-les, van sorgir noves eines per fer-hi front. Així, a principis de segle, va aparèixer el moviment NOSQL,[5] que mostrava la necessitat de trencar amb l'arquitectura monolítica dels SGBD relacionals tradicionals. Tanmateix, malgrat que el nom sembla indicar algun problema amb l'SQL (de l'anglès *structured query language*, llenguatge de consulta estructurat), el problema real dels SGBD relacionals no tenia res a veure amb aquest llenguatge de consultes, sinó amb: 1) simplificar les funcions innecessàries que en comprometien el rendiment; 2) permetre l'escalabilitat sobre el maquinari característic del núvol; 3) augmentar-ne la disponibilitat i la fiabilitat, i, alhora, 4) reduir la complexitat de la gestió del mateix sistema.

La hipòtesi d'aquest moviment era que no existeix un sistema que sigui el millor en tots els casos, sinó que, de-

5. Not Only SQL.

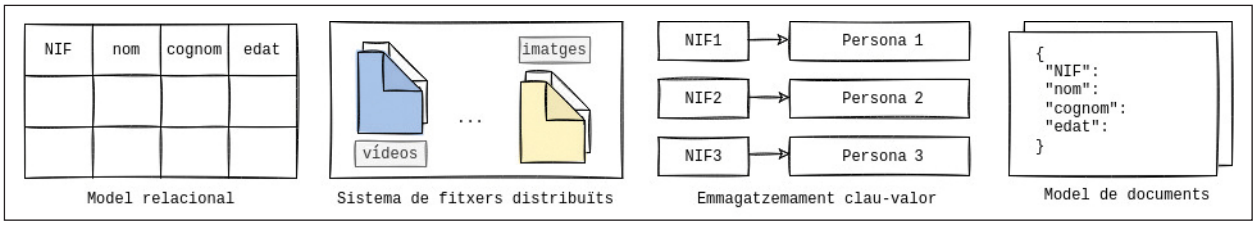

Figura 1. Models alternatius de gestió de dades.
Font: Elaboració pròpia.

penent del problema que tinguem, pot ser més adequat un sistema o un altre. Per exemple, per a sistemes de magatzem de dades (*data warehouse*) podem utilitzar SGBD com ara Vertica o Amazon Redshift; per gestionar dades de la web semàntica, podem utilitzar Virtuoso o GraphDB; per treballar amb dades científiques, Matlab o SciDB; etc. Això s'explica clarament a Athanassoulis *et al*. 2016, en què els autors conjecturen que un sistema només pot maximitzar dues de les tres característiques següents: l'ús de l'espai, la velocitat de lectura i la velocitat d'escriptura. Així doncs, com que un sistema concret no pot optimitzar totes tres característiques alhora, depenent de quina sigui la prioritat en el nostre cas d'ús, haurem de triar un sistema o un altre.

A continuació es veu, a tall d'exemple, alguns sistemes possibles que podem utilitzar per gestionar dades massives. Els seus models de dades estan esquematitzats a la figura 1. Tenint com a referència el model relacional que estructura les dades en forma de taules, podem tenir simplement: 1) fitxers emmagatzemats al sistema operatiu sense cap tipus d'estructura concreta, 2) dades estructurades en forma de parelles de clau-valor planes o 3) estructures de parelles clau-valor imbricades més complexes, que normalment s'anomenen *documents*.

4.1. *Sistemes de fitxers distribuïts*

Com ja s'ha explicat, la gestió de grans quantitats de dades requereix un processament distribuït i paral·lel. Tanmateix, no és gens evident com s'ha de gestionar al més baix nivell. El primer que hem de resoldre és com el sistema de fitxers gestiona els blocs de disc,[6] de manera que característiques com ara la tolerància a errors o la sincronització siguin transparents per als desenvolupadors.

A causa de la naturalesa d'un sistema de fitxers distribuïts, algunes de les opcions clàssiques en el disseny de sistemes de fitxers centralitzats ja no són vàlides. Quan la distribució és al núvol, els problemes no fan més que amplificar-se. Per exemple, els sistemes de dades massives estan pensats per emmagatzemar dades generades per aplicacions com ara registres del sistema, interaccions de xarxes socials o generació i intercanvi de continguts multimèdia (per exemple, vídeo). Per tant, no és estrany que la mida d'un sol fitxer estigui en el rang de GB o TB, i n'esperem un gran nombre. Se suposa que els fitxers petits (és a dir, kB o pocs MB) poques vegades haurien d'existir en aquests casos, i no cal optimitzar el sistema per fer front a la seva gestió.

En un sistema com aquest de fitxers distribuïts en el núvol, hem de permetre, primerament, que diversos clients afegeixin dades simultàniament al mateix fitxer i, en segon lloc, que moltes màquines diferents (centenars o milers) participin del mateix sistema de fitxers. Conseqüentment, donat el gran nombre de participants, és estadísticament molt probable que es produeixi algun error de maquinari. Així doncs, aquest tipus de situacions han ser monitorades i detectades, i quan es produeixin, s'han d'establir mecanismes de recuperació per revertir possibles inconsistències.

Tots aquests problemes (i més) ja els va resoldre Google amb el seu sistema de fitxers (Google File System). Arran de la publicació de les seves especificacions, el projecte Apache va llançar l'equivalent versió de codi obert amb el nom Hadoop Distributed File System (HDFS).[7]

4.2. *Sistemes d'emmagatzemament clau-valor*

El problema dels sistemes de fitxers és que només permeten accés seqüencial (és a dir, obrir un determinat fitxer i llegir-lo des de l'inici fins al final). Evidentment, això és molt limitat i ineficient en molts casos d'ús en què es requereix accés aleatori (típicament implementat mitjançant algun mecanisme d'indexació).

La manera més bàsica d'implementar accés aleatori és mitjançant parelles clau-valor. És a dir, a cada valor li associem una clau en emmagatzemar-lo, de manera que, quan necessitem recuperar-lo, podem obtenir-lo directament només utilitzant la mateixa clau (aquesta clau fa el paper d'identificador dels diferents valors que anem emmagatzemant).

De nou Google va ser el primer d'implementar això a gran escala al núvol (BigTable), i de nou també el projecte Apache va crear un sistema obert basant-se en les seves especificacions, amb el nom HBase.[8]

6. Un bloc de disc és la unitat de direccionament del maquinari on s'emmagatzemen físicament les dades.

7. *https://hadoop.apache.org*.

8. *https://hbase.apache.org*.

4.3. *Gestors de documents*

La principal limitació de l'emmagatzemament clau-valor és que el valor es tracta com una caixa negra sense cap tipus d'estructura. Conseqüentment, no permet cap tipus d'indexació secundària i estem limitats a l'accés aleatori només a través de la clau.

Òbviament, és fàcil trobar casos d'ús en què requerim tenir índexs secundaris per aconseguir un temps de resposta adequat. No obstant això, per poder fer-ho, el primer que necessitem és dotar el valor d'algun tipus d'estructura que ho permeti.

Tanmateix, una estructura tabular rígida com la del model relacional no és adequada avui dia en què predomina el desenvolupament àgil de programari i l'esquema de dades està en contínua evolució. Així doncs, el terme *dades semiestructurades* va sorgir per descriure dades que tenen alguna estructura però que no és regular ni coneguda *a priori* pel sistema. Aquestes dades semiestructurades es defineixen en termes de documents (en format JSON o XML), cadascun dels quals conté la descripció del propi contingut (és a dir, cada document té el seu propi esquema independent).

Els sistemes que gestionen aquest tipus de dades es coneixen com a *gestors documentals* i actualment el més popular és MongoDB.[9]

4.4. *Entorns de processament distribuït a gran escala*

Seguint la filosofia NOSQL de creació de components independents i molt especialitzats, els tres sistemes descrits anteriorment s'especialitzen només en l'emmagatzemament distribuït depenent de l'estructura de les dades. Si volem ara ser capaços de paral·lelitzar-ne el processament, independentment d'aquesta estructura, necessitem un nou tipus de sistema que treballi sobre algun dels anteriors. Aquest tipus de sistema ha de permetre executar tasques en paral·lel. Com s'ha dit anteriorment, això no és cap novetat. Les arquitectures paral·leles ja es van introduir als anys noranta. Tanmateix, es van posar especialment de moda gràcies a Google, que de nou va fer públic el seu entorn de desenvolupament d'aplicacions MapReduce, implementat immediatament en codi obert dins del projecte d'Apache. No obstant això, com passa amb molts sistemes NOSQL, era massa específic i li mancava generalitat per realment ser útil a tothom. Així, la Universitat de Califòrnia va iniciar un projecte per millorar-lo i va desenvolupar-ne una generalització com a part de la tesi doctoral de Matei Zaharia, que es va convertir també en codi obert el 2010 i que es va traslladar a l'Apache Software Foundation el 2013 amb el nom de Spark. Només dos anys després, Spark ja era un dels projectes Apache més actius segons les estadístiques de la mateixa pàgina web.[10]

L'objectiu d'un entorn de desenvolupament d'aplicacions com Spark és alleugerir els desenvolupadors de la complexitat de distribuir i paral·lelitzar el seu codi per tal que s'executi de manera eficient sobre conjunts de dades massives en el núvol. La idea és oferir una abstracció que permeti especificar les transformacions de dades utilitzant programació funcional. És a dir, els paràmetres d'una funció poden ser també funcions. En el cas de MapReduce, les funcions que es podien parametritzar es deien *map* i *reduce*, d'aquí el nom d'aquest entorn de desenvolupament. L'origen d'aquestes funcions el trobem a LISP. La funció *map* pren com a argument una funció unària i un conjunt de valors, de manera que la funció s'aplica a cadascun dels valors. Per exemple, «(map 2 × (3, 4 ,5 ,6))» multiplica per dos cadascun dels valors del conjunt proporcionat i dona com a resultat el conjunt «(6, 8, 10, 12)». Alhora, la funció *reduce* pren com a argument una funció binària i un conjunt de valors, que es combinen per parelles utilitzant aquesta funció. Per exemple, «(reduce + (6, 8, 10, 12))» té com a entrada la sortida anterior i suma tots els seus elements, i dona com a resultat «36». De manera similar, Google utilitzava aquestes dues funcions per, primer, calcular les estadístiques i els hipervincles d'una pàgina web concreta, i, seguidament, agregar aquests càlculs i obtenir el rànquing de tot Internet.

En aquest punt, és fonamental adonar-se que l'operador *map* es pot executar en paral·lel sobre cadascun dels elements del conjunt de dades (o les pàgines web en el cas de Google), ja que no hi ha cap dependència entre ells. Potser no és tan evident, però l'operador *reduce* també és fàcilment paral·lelitzable.

L'entorn de desenvolupament de Spark és simplement una evolució i generalització de MapReduce, que ofereix moltes més operacions i una gestió de la memòria molt més eficient. Igual que MapReduce, Spark segueix una arquitectura amb un únic procés coordinador i molts altres que fan el rol de treballadors i són els realment encarregats d'executar les transformacions de dades en paral·lel. Aquests treballadors requereixen recursos del núvol (disc, memòria, processadors, etc.), que poden ser gestionats per gestors de recursos com ara YARN, Mesos o Kubernetes, entre d'altres.

L'execució eficient d'un treball Spark es basa en la seva representació en forma de graf dirigit, on cada node correspon a una transformació de dades i les arestes són les dependències entre elles. El planificador de Spark examina el graf generat per l'usuari i crea una nova estructura de dades amb diferents etapes. Aquestes etapes són, en realitat, subgrafs del graf de l'usuari amb l'objectiu de maximitzar el nombre de tasques aplicables en paral·lel a fragments de dades independents, i reduir les barreres de sincronisme que requereixen coordinació entre treballadors, en forma de moviment de dades en la xarxa.

5. Arquitectura

Un cop vistes diverses alternatives per als components del nostre sistema, el que hem de veure ara és com hem de

9. *https://www.mongodb.com*.
10. *https://projects.apache.org/statistics.html*.

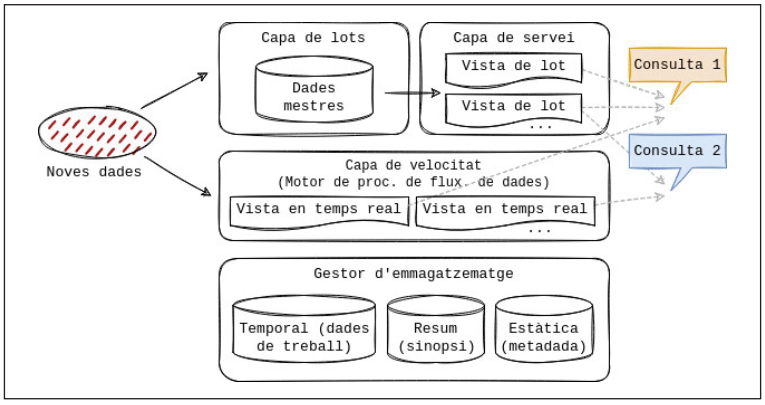

Figura 2. Arquitectura-γ.
Font: Elaboració pròpia.

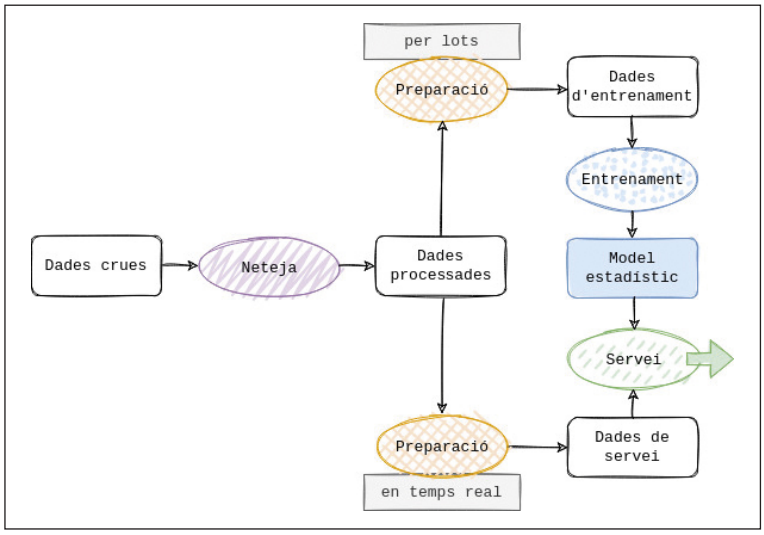

Figura 3. Fluxos de transformació de dades.
Font: Elaboració pròpia.

connectar els que hàgim triat (per exemple, HDFS, HBase, MongoDB, Spark). Per això, el primer que hem de fer és adonar-nos que podem trobar requisits contradictoris dins del mateix sistema d'anàlisi de dades massives. D'una banda, l'anàlisi predictiva té com a objectiu preveure com una determinada entitat (per exemple, un client, una persona usuària) es comportarà en un futur proper. Evidentment, ja que la finalitat d'una predicció és reaccionar o almenys estar preparat per prendre alguna acció, disposar de les dades totalment actualitzades i un bon temps de resposta davant de les consultes és crucial en aquest cas. Per contra, l'anàlisi descriptiva estudia com funciona el negoci a diferents nivells de granularitat (per exemple, regions, ciutats o districtes) i com evoluciona al llarg del temps. El fet de no incloure en l'anàlisi les dades més recents no sol ser un problema per a les tendències a llarg termini, i incórrer en dies o fins i tot setmanes d'endarreriment en el processament és acceptable en aquests casos.

En conseqüència, com que els requisits temporals dels diferents tipus d'anàlisi són contradictoris, hem de distingir ambdós fluxos de processament, la qual cosa dona lloc al que es coneix com a *arquitectura-*γ.[11] Aquesta, esquematitzada a la figura 2, consta de dues branques d'execució alimentades de les mateixes fonts: una se centra en el processament i en diferit per lots, i l'altra, en el processament en temps real. És important, però, adonar-se que el manteniment d'aquests fluxos potencialment redundants genera alguns riscos de gestió.

De la mateixa manera que està plenament justificat que l'anàlisi descriptiva no disposi de les dades totalment actualitzades, tampoc no ho requereix la creació del model predictiu i, consegüentment, es pot executar per lots. Per contra, l'aplicació d'aquest model sí que tindrà uns requeriments temporals molt més restrictius i s'haurà d'executar en temps real. En aquest punt, és important adonar-se que, perquè la predicció tingui sentit, hem de garantir que les dades utilitzades per crear el model han de tenir exactament les mateixes transformacions que les dades sobre les quals volem fer després la predicció amb aquest model (en cas contrari, la validesa de la predicció es veuria compromesa). Això ho podem veure esquematitzat a la figura 3, en què la transformació per lots i la transformació en temps real haurien de coincidir en tot moment per obtenir resultats coherents.

Així, l'arquitectura-γ va evolucionar cap a una arquitectura-κ (Kreps, 2014), com una simplificació amb un únic motor d'execució (per tant, una implementació única de

11. El nom prové de la semblança de les dues branques d'execució amb la lletra grega.

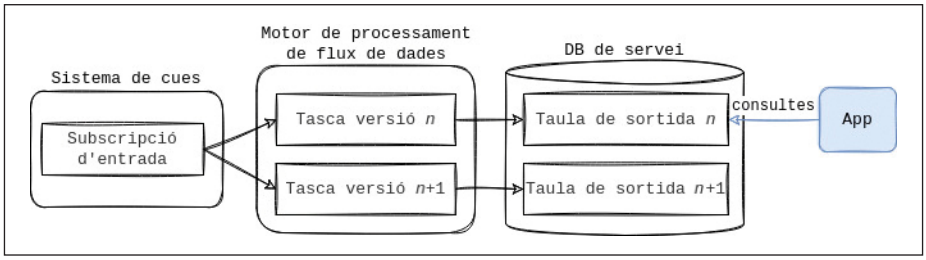

FIGURA 4. Arquitectura-κ.
FONT: Elaboració pròpia.

les transformacions de les dades), tal com podem veure esquematitzat a la figura 4. La manera, doncs, d'executar els processos per lots és simplement reproduir les dades a través del sistema com si es tractés d'un flux que arriba de forma contínua i que es processa en temps real. Si, per qualsevol motiu, necessitem diferents versions de les transformacions per a diferents models predictius, podem mantenir-les totes en el mateix sistema i triar la més adequada en cada moment, de manera independent de si és per a desenvolupament o producció. Podem trobar una versió més avançada d'aquestes arquitectures i diversos casos d'ús a Nadal *et al.*, 2017.

6. Conclusions

Hi ha la creença que la part més complexa d'un projecte de dades massives és l'aprenentatge automàtic i la creació de models estadístics. No obstant això, malgrat que aquesta part requereix coneixements matemàtics i algorísmics molt especialitzats, no és normalment la que requereix més esforços en un projecte d'aquest tipus. El fet que no existeixi una solució genèrica per a qualsevol cas d'ús fa que la gestió de les dades s'emporti habitualment molt més temps que l'aprenentatge automàtic pròpiament, i la manca de coneixements específics sobre el tema pot dur el projecte al fracàs.

En el present article hem repassat les diferents eines de gestió de dades massives utilitzades actualment, des dels sistemes de fitxers fins als sistemes de processament distribuïts, passant pels sistemes d'emmagatzemament clau-valor i documentals, així com les arquitectures γ i κ, que indiquen com organitzar la seva utilització.

Finançament

Aquest treball està parcialment finançat per la Comissió Europea sota l'acord 011093164, corresponent al projecte ExtremeXP.

Bibliografia

ATHANASSOULIS, M. [*et al.*] (2016). «Designing access methods: The RUM conjecture». A: *Proceedings of the 19th International Conference on Extending Database Technology* (*EDBT*). Bordeus: OpenProceedings.org, p. 461-466.

CARNELLEY, P.; SCHWENK, H. (2016). «Big data: Turning promise into reality». *IDC White Paper* [en línia]. <https://www.scribd.com/document/343663115/Dell-Emc-Big-Data-Turning-Promise-Reality> [Consulta: 15 setembre 2022].

DAVENPORT, T.; BEAN, R. (2022). *Data and AI leadership executive survey 2022* [en línia]. NewVantage Partners. <wavestone-2022-Data-and-AI-Leadership-Executive-Survey-Report-1.pdf> [Consulta: 24 octubre 2022].

FRIENDLY, M.; DENIS, D. J. (2001). *Milestones in the history of thematic cartography, statistical graphics, and data visualization*. Toronto: York University.

KREPS, J. (2014). «Questioning the Lambda architecture». *O'Reilly* [en línia]. <https://www.oreilly.com/radar/questioning-the-lambda-architecture> [Consulta: 24 octubre 2022].

MARR, B. (2018) «How much data do we create everyday? The Mind-Blowing stats everyone should read». *Forbes* [en línia]. <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read> [Consulta: 24 octubre 2022].

NADAL, S. [*et al.*] (2017). «A software reference architecture for semantic-aware big data systems». *Information and Software Technology*, 90, p. 75-92.

SAMBASIVAN, N. [*et al.*] (2021). «"Everyone wants to do the model work, not the data work": Data cascades in high-stakes AI». A: *CHI'21: Proceedings of the 2021 Conference on Human Factors in Computing Systems*. Nova York, EUA: Association for Computing Machinery, article 39, p. 1-15. <https://doi.org/10.1145/3411764.3445518>.

SCULLEY, D. [*et al.*] (2015). «Hidden technical debt in machine learning systems». A: *Advances in neural information processing systems 28: 29th Annual Conference on Neural Information Processing Systems 2015*. Vol. 2, p. 2503-2511.