Tipologies i arquitectures d'un sistema *big data*

PID_00250685

Jordi Nin

Temps mínim de dedicació recomanat: 4 hores

Universitat Oberta de Catalunya

Índex

Introducció

És un fet natural que cada dia generem més i més dades, i que capturar-les, emmagatzemar-les i processar-les són peces fonamentals en una gran varietat de situacions, bé siguin d'àmbit empresarial o bé tinguin la finalitat de dur a terme algun tipus de recerca científica.

Per assolir aquests objectius, cal habilitar un conjunt de tecnologies que permetin dur a terme totes les tasques necessàries en el procés d'anàlisi de grans volums d'informació o dades massives (*big data*). Aquestes tecnologies tenen impacte en gairebé totes les àrees de les tecnologies de la informació i de les comunicacions, també conegudes com a TIC: des del desenvolupament de nous sistemes d'emmagatzematge de dades —com serien les memòries d'estat sòlid o *solid state disk* (SSD), que permeten accedir de manera eficient a grans conjunts de dades— o el desenvolupament de xarxes d'ordinadors més ràpides i eficients —basades, per exemple, en fibra òptica, que permeten compartir gran quantitat de dades entre múltiples servidors—, fins a noves metodologies de programació que permeten als desenvolupadors i investigadors fer ús d'aquests nous components de maquinari d'una manera relativament senzilla.

En general, tota arquitectura de dades massives requereix una gran quantitat de servidors, generalment ordinadors de propòsit general com els que tenim a casa. Cadascun d'aquests servidors disposa de diverses unitats centrals de procés (CPU), d'una gran quantitat de memòria principal d'accés aleatòria (RAM) i d'un conjunt de discos durs per a l'emmagatzematge estable d'informació, tant de dades capturades del món real com de resultats ja processats.

El principal objectiu d'una arquitectura de dades massives és que tots aquests elements —CPU, memòria i disc— siguin accessibles de manera distribuïda, per tal que el seu ús sigui transparent per a l'usuari i hi hagi una falsa sensació de centralitat, és a dir, que per a l'usuari de la infraestructura solament hi hagi un únic conjunt de CPU, una única memòria central i un sistema d'emmagatzematge central. Per permetre aquesta abstracció cal que les dades es trobin distribuïdes i copiades diverses vegades en diferents servidors i que l'entorn de programació permeti distribuir els càlculs que s'han de realitzar de manera senzilla alhora que eficient.

En primer lloc, hem de ser capaços de capturar i emmagatzemar les dades disponibles. Actualment, els sistemes de gestió d'arxius distribuïts més habituals que trobem en arquitectures de dades massives són bases de dades relacionals, com Oracle, PostgreSQL o IBM-bd2; bases de dades NoSQL com Apatxe Cassandra, Redis o mongoDB, i sistemes de fitxers de text com HDFS (Hadoop Distributed File System), que permeten emmagatzemar grans volums de

dades. Aquests sistemes són els encarregats d'emmagatzemar les dades i permetre'n l'accés d'una manera eficient. En paral·lel, aquests sistemes permeten guardar els resultats parcials generats durant el processament de les dades i també els resultats finals, en el cas que els parcials ocupin molt d'espai.

En segon lloc, trobem tres entorns de processament de dades predominants al mercat actual:

- **Hadoop MapReduce**. Dissenyat inicialment per Google amb codi propietari i alliberat posteriorment per Yahoo! com a codi obert o *open source*, basa la seva manera de processar les dades en dues operacions senzilles: l'operació **Map**, que distribueix el còmput juntament amb les seves dades corresponents als diferents servidors, i l'operació **Reduce**, que combina tots els resultats parcials obtinguts en les diferents operacions Map. La principal limitació d'aquest model de processament és l'ús intensiu que fa del disc dur, fet que fa que sigui ineficient en molts casos, però extremadament útil en d'altres.

- **Apache Spark**. Desenvolupat per Matei Zaharia, que, tot i que es basa en la mateixa idea de «divideix i venceràs» de Hadoop MapReduce, utilitza la memòria principal per a distribuir i processar les dades. Això li permet ser molt més eficient quan ha de realitzar càlculs iteratius (que es repeteixen contínuament). Aquesta capacitat el converteix en el substitut perfecte de Hadoop MapReduce quan aquest últim no és vàlid.

- **GPU Computing**. Aquest tipus de processament pot definir-se com l'ús d'una unitat de processament gràfic (GPU) disponible en qualsevol ordinador en combinació amb una CPU per tal d'accelerar aplicacions d'anàlisi de dades i enginyeria. Aquests sistemes solen emprar-se en situacions que no requereixen una gran quantitat de càlculs.

Tots aquests entorns s'enfronten a dos grans reptes:

1) Desenvolupar algorismes que siguin capaços de treballar únicament amb una part de les dades i que els seus resultats siguin associatius i fàcilment combinables.

2) Distribuir les dades de manera eficient entre els servidors per no saturar la xarxa durant el processament.

En aquest mòdul ampliarem totes aquestes idees i veurem com és possible afrontar aquests reptes.

Objectius

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

1. Comprendre els diferents components de maquinari d'una arquitectura de dades massives.
2. Conèixer l'*stack* de programari típic de gestió d'una arquitectura de dades massives.
3. Entendre com s'emmagatzemen i distribueixen les dades massives en un sistema d'arxius distribuït.
4. Entendre les diferents jerarquies de memòria per poder processar dades massives de manera eficient.
5. Ser capaç de diferenciar els diferents tipus de processament distribuït: el model *batch* (per lots) enfront del model *streaming* (seqüencial).

1. Què entenem per dades massives o *big data*?

Big data o **dades massives** és el concepte amb el qual fem referència a la identificació, la captura, l'emmagatzematge i el processament de grans quantitats de dades i, alhora, als procediments emprats per a extreure coneixement vàlid de les dades.

Generalment, als documents cientificotècnics s'usa directament el terme en anglès *big data*, tal com apareix en l'article seminal de Viktor Schönberger «Big data: revolució de les dades massives».

Bibliografia complementària

Viktor Mayer-Schönberger; Kenneth Cukier (2013). *Big Data: A Revolution That Will Transform How We Live, Work and Think*. Boston, Nova York: John Murray Publishers Ltd.

> ***Dades massives*** és un terme que fa referència a un volum de dades tan gran que supera la capacitat habitual tant del maquinari com del programari per capturar, administrar i processar les dades en un temps raonable. El volum a partir del qual les dades comencen a considerar-se massives creix constantment. A l'article de Viktor Schönberger s'estimava la seva grandària d'entre una dotzena de terabytes fins a diversos petabytes de dades en un únic conjunt de dades. Avui dia aquest volum es queda petit, ja que podem emmagatzemar diversos terabytes d'informació en els ordinadors convencionals.

Un dels principals problemes de les dades massives és que, a diferència dels sistemes gestors de bases de dades tradicionals, no es limiten a fonts de dades amb una estructura determinada i senzilla d'identificar i processar. Generalment diferenciem tres tipus de fonts de dades massives. Noteu que aquesta classificació s'aplica tant a dades massives com a no massives:

- **Dades estructurades (*structured data*)**. Són dades que tenen ben definides la seva longitud i el seu format, com les dates, els nombres o les cadenes de caràcters. S'emmagatzemen en format tabular. Exemples d'aquest tipus de dades són les bases de dades relacionals i els fulls de càlcul.

- **Dades no estructurades (*unstructured data*)**. Són dades que en el seu format original no disposen d'un format específic. No es poden emmagatzemar en un format tabular perquè la seva informació no es pot desgranar en un conjunt de tipus bàsics de dades (nombres, dates o cadenes de text). Exemple d'aquest tipus de dades són els documents PDF o Word, documents multimèdia (imatges, àudio o vídeo), correus electrònics, etc.

- **Dades semiestructurades (*semistructured data*)**. Són dades que no es limiten a un conjunt de camps definits com en el cas de les dades estructurades, sinó que contenen marcadors per separar els seus diferents elements. És una informació poc regular com per ser gestionada d'una manera estàndard (taules). Aquest tipus de dades posseeix les seves pròpies metadades —dades que defineixen com són les dades— semiestructurades que descriuen els objectes i les seves relacions, i que en alguns casos estan acceptades per convenció, com per exemple els formats HTML, XML o JSON.

Dades semiestructurades

Trobem diferents tipus de codificacions per a les dades semiestructurades, com són:
1) HTML. L'HyperText Markup Language és un llenguatge de programació que s'utilitza per al desenvolupament de pàgines d'internet.
2) XML. L'Extended Markup Language no és un llenguatge en si mateix, sinó un sistema que permet definir llenguatges d'acord amb les necessitats.
3) JSON. El Javascript Object Notation és un format lleuger d'intercanvi de dades pensat perquè als ordinadors els resulti simple interpretar-lo i generar-lo.

Moltes vegades trobem el concepte de dades massives relacionat amb diferents conceptes diferents coneguts com les *v* del *big data*:

- **V de volum**. Aquest és el primer aspecte que ens ve al cap quan pensem en les dades massives. Ens diu que les dades tenen un volum massa gran per ser gestionades d'una manera tradicional en un temps raonable.

- **V de velocitat**. Tot i que les dades no sofreixen variacions gaire freqüents, normalment es poden analitzar amb tècniques tradicionals sense problemes, tot i que això pot tardar hores i fins i tot dies. No obstant això, en l'àmbit de les dades massives la quantitat d'informació creix tan de pressa que el temps de processament de la informació es converteix en un factor fonamental perquè aquest tractament aporti avantatges que marquin la diferència.

- **V de varietat**. Com hem descrit anteriorment, les dades massives no es processen únicament com a dades estructurades. Tècnicament, no és senzill incorporar grans volums d'informació a un sistema d'emmagatzematge quan el seu format no està perfectament definit. En aquest escenari ens trobem amb infinitat de tipus de dades que s'aglutinen per a ser tractades, i és per això que enfront d'aquesta varietat augmenta el grau de complexitat tant en l'emmagatzematge com en el seu processament.

- **V de veracitat**. Quan disposem d'un alt volum d'informació que creix a gran velocitat i que pot tenir una estructura molt variada, és inevitable dubtar del grau de veracitat que tenen aquestes dades. Així doncs, cal fer neteja de dades i verificar-les per assegurar que generem coneixement sobre dades veraces.

Les *v* del *big data*

Les tres primeres *v* apareixen a la definició original de les dades massives, i més endavant s'hi va incorporar la quarta. Encara a dia d'avui aquesta quarta *v* no està acceptada per tothom.

1.1. Utilitat: on trobem dades massives?

La resposta a aquesta pregunta és senzilla; en trobem en tots els àmbits del coneixement, com per exemple:

- **Xarxes socials**. El seu ús, que cada vegada està més estès, fa que els usuaris hi incorporin cada vegada més una gran part de la seva activitat i la dels seus coneguts. Les empreses utilitzen tota aquesta informació amb moltes

finalitats. Per exemple, per fer estudis de màrqueting, per avaluar la seva reputació o fins i tot per creuar les dades dels candidats a un lloc de treball determinat.

- **Consum**. Amazon és capdavanter en vendes creuades. Gran part del seu èxit es basa en l'anàlisi massiva de dades de patrons de compra d'un usuari creuades amb les dades de compra d'uns altres, creant així anuncis personalitzats i butlletins electrònics que inclouen justament tot allò que l'usuari vol en aquell instant.

- **Salut i medicina**. L'any 2009, el món va experimentar una pandèmia de grip A, també coneguda com a grip porcina o H1N1. El web Google Flu Trends* va ser capaç de predir-la gràcies als resultats de les cerques de paraules clau al seu cercador. Flu Trends va fer servir les dades de les cerques dels usuaris que contenen *influenza-like illness symptoms* —que es pot traduir com a 'símptomes semblats a la malaltia de la grip'— i les va agregar segons ubicació i data, de manera que va ser capaç de predir l'activitat de la grip fins i tot amb dues setmanes més d'antelació respecte als sistemes tradicionals.

* http://bit.ly/2CXuECR

- **Política**. Barack Obama va ser el primer candidat a la presidència dels Estats Units que va basar tota la seva campanya electoral en les anàlisis realitzades pel seu equip de dades massives.** Aquesta anàlisi va ajudar Barack Obama a obtenir la victòria enfront de l'altre candidat republicà, Mitt Romney, amb el 51,06% dels vots, una de les eleccions presidencials més disputades.

** http://bit.ly/2yTn9K1

- **Telefonia**. Les companyies de telefonia utilitzen la informació generada pels telèfons mòbils —posició GPS i els CDR— per a estudis demogràfics, planificació urbana, etc.

Call detail record

Call detail record (CDR) és un registre de dades generat en la comunicació entre dos telèfons fixes o mòbils que documenta els detalls de la comunicació (per exemple, trucada telefònica, missatges de text, etc.). El registre conté diversos atributs com l'hora, la duració, l'estat de finalització, el número de la font o el número de destí.

- **Finances**. Els grans bancs disposen de sistemes de *trading* algorítmic que analitzen una gran quantitat de dades de tot tipus per decidir quines operacions en borsa són les més rendibles en cada moment.

***Trading* algorítmic**

El *trading* algorítmic és una modalitat d'operació en els mercats financers (*trading*) que es caracteritza per l'ús d'algorismes, regles i procediments automatitzats en diferents graus, per tal d'executar operacions de compra o venda d'instruments financers.

1.2. Dades, informació i coneixement

L'última de les quatre *v* de les dades massives ens adverteix que no totes les dades que capturem tenen valor. Ja fa molt de temps que Albert Einstein va dir que «la informació no és coneixement». Quanta raó tenia! Les dades necessiten ser processades i analitzades perquè se'n pugui extreure el valor que contenen.

En general, diem que les **dades** són la mínima unitat semàntica i que es corresponen amb elements primaris d'informació que per si sols són irrellevants com a suport a la presa de decisions. El saldo d'un compte corrent o el nom-

bre de fills d'una persona, per exemple, són dades que, sense un propòsit, una utilitat o un context, no serveixen com a base per a recolzar la presa d'una decisió.

Per contra, parlem d'**informació** quan obtenim un conjunt de dades processades que tenen un significat (rellevància, propòsit i context) i que, per tant, són d'utilitat per a qui ha de prendre decisions, ja que disminueixen la seva incertesa.

Finalment, definirem **coneixement** com una barreja d'experiència, valors i informació que serveix com a marc per a la incorporació de noves experiències i informació i és útil per a la presa de decisions.

És fonamental aconseguir la tecnologia, tant de maquinari com de programari, per transformar les dades en informació, a més de l'habilitat analítica humana per transformar la informació en coneixement, de manera que per mitjà d'aquest coneixement puguem optimitzar els processos de negoci.

1.3. Com processem tota aquesta informació?

Com hem introduït anteriorment, una de les principals característiques de les dades massives és la capacitat de processar una gran quantitat de dades en un temps raonable. Això és possible gràcies a la **computació distribuïda**.

La computació distribuïda és un model que serveix per resoldre problemes de computació massiva per mitjà d'un gran nombre d'ordinadors organitzats en clústers incrustats en una infraestructura de telecomunicacions que s'ocupa tant de distribuir les dades com els resultats obtinguts durant el còmput.

Les característiques principals d'aquest model són les següents:

- La manera de treballar dels usuaris en la infraestructura de dades massives ha de ser similar a la que tindrien en un sistema centralitzat.
- La seguretat interna en el sistema distribuït i la gestió dels seus recursos és responsabilitat del sistema operatiu i dels seus sistemes de gestió i administració.
- S'executa en múltiples servidors alhora.

Múltiples servidors

En general, quan ens referim a un conjunt de servidors parlem d'un clúster.

- Ha de proveir un entorn de treball còmode per als programadors d'aplicacions.

- Disposa d'un sistema de xarxa que interconnecta els diferents servidors de manera transparent amb l'usuari.

- Ha de proveir transparència en l'ús de múltiples processadors i en l'accés remot.

- El seu disseny de programari ha de ser compatible amb diversos usuaris i sistemes que interactuen al mateix temps.

Tot i que l'ús de la computació distribuïda facilita molt la feina, els mètodes tradicionals de processament de dades no són vàlids per a aquests sistemes de càlcul. Per aconseguir l'objectiu de processar grans conjunts de dades, Google va desenvolupar l'any 2004 la metodologia de processament de dades MapReduce,* un motor que actualment està darrere dels processaments de dades de Google. No obstant això, va ser el desenvolupament de Hadoop MapReduce per part de Yahoo! el que va propiciar un ecosistema d'eines de codi obert (*open source*) de processament de grans volums de dades.

La innovació clau de MapReduce és la capacitat d'executar un programa, dividint-lo i executant-lo en paral·lel alhora, mitjançant múltiples servidors sobre un conjunt de dades immens que també està distribuït.

Tot i que l'aparició de MapReduce va canviar completament la manera de treballar i va facilitar l'aparició de les dades massives, també té una gran limitació: únicament és capaç de distribuir el processament als servidors copiant les dades que s'han de processar per mitjà del seu disc dur. Aquesta limitació fa que aquest paradigma de processament sigui poc eficient quan és necessari realitzar càlculs iteratius.

Posteriorment, l'any 2014, Matei Zaharia va crear Apache Spark,** que solucionava les limitacions de MapReduce permetent distribuir les dades als servidors amb la seva memòria principal o RAM. Aquesta innovació ha permès que molts algorismes de processament de dades es puguin aplicar eficientment a grans volums de dades de manera distribuïda.

En paral·lel a aquestes millores, des de l'any 2007 empreses com NVIDIA han començat a introduir l'ús de unitats de processament gràfic (GPU) com a alternativa al càlcul tradicional en unitats centrals de processament (CPU). Les GPU disposen de processadors molt més simples que les CPU. Això facilita que en una sola targeta gràfica instal·lada en un servidor puguin integrar-se més processadors i que, per tant, es pugui fer un gran nombre de càlculs numèrics en paral·lel.

* http://bit.ly/15V7Pyh

Open source

Open source (en català, 'codi obert') és el terme amb el qual es coneix el programari distribuït i desenvolupat lliurement. El codi obert té un punt de vista més orientat als beneficis pràctics de compartir el codi que a les qüestions ètiques i morals, que destaquen a l'anomenat programari lliure.

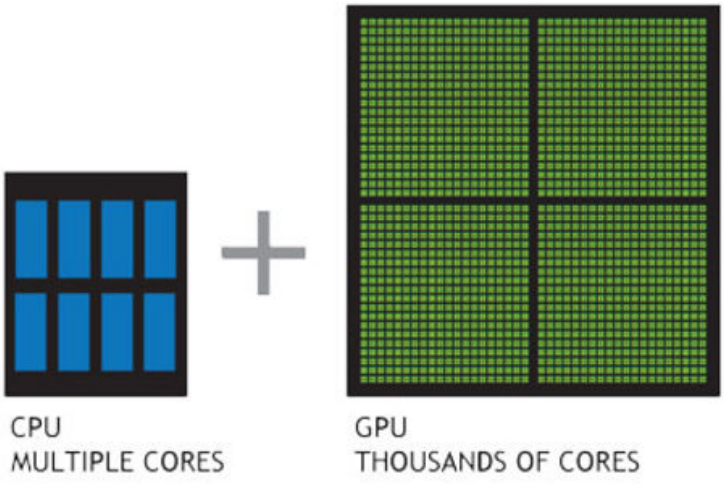
Exemple de càlcul iteratiu

Un exemple de càlcul iteratiu és el càlcul dels pesos d'una recta de regressió o, en general, qualsevol mètode d'estimació de paràmetres basat en el descens del gradient.

** http://spark.apache.org

Una manera senzilla d'entendre la diferència entre la GPU i la CPU és comparar la manera que tenen de processar les tasques. Una CPU està formada per diversos nuclis optimitzats per al processament en sèrie, mentre que una GPU consta de milers de nuclis més petits i eficients dissenyats per a gestionar múltiples tasques simultàniament.

Figura 1. Diferències entre una arquitectura basada en CPU i en GPU



Font: http://www.nvidia.com/object/what-is-gpu-computing.html

En general, una CPU i una GPU són el mateix: circuits integrats amb una gran quantitat de transistors que realitzen càlculs matemàtics llegint nombres en binari. La diferència és que la CPU és un processador de propòsit general, amb el qual podem fer qualsevol tipus de càlcul, mentre que la GPU és un processador de propòsit específic: està optimitzada per treballar amb grans quantitats de dades i realitzar les mateixes operacions una vegada i una altra.

Per això, encara que ambdues tecnologies es dediquin a realitzar càlculs, tenen un disseny substancialment diferent. La CPU està dissenyada per al processament en sèrie: es compon de pocs nuclis molt complexos que poden executar pocs programes al mateix temps. En canvi, la GPU té centenars o milers de nuclis senzills que poden executar centenars o milers de programes específics alhora. Les tasques de les quals s'encarrega la GPU requereixen un alt grau de paral·lelisme: tradicionalment, una instrucció i múltiples dades.

1.4. Computació científica

La computació científica és el camp d'estudi relacionat amb la construcció de models matemàtics, algorismes i tècniques numèriques que s'ocupa de resoldre problemes científics, d'anàlisi de dades i problemes d'enginyeria. Típi-

cament, es basa en l'aplicació de diferents formes de càlcul de problemes de diverses disciplines científiques.

Aquest tipus de computació requereix una gran quantitat de càlculs (usualment de punt flotant) i normalment s'executen en superordinadors o plataformes de computació distribuïda com les descrites anteriorment, sigui utilitzant processadors d'ús general o CPU, o bé utilitzant processadors gràfics (GPU) adaptats al còmput numèric, com hem descrit al subapartat anterior.

Les aplicacions principals de la computació científica són:

- **Simulacions numèriques.** Els treballs d'enginyeria que es basen en la simulació busquen millorar la qualitat dels productes —per exemple, millorar la resistència al vent d'un nou model de cotxe— i reduir els temps i costos de desenvolupament, ja que tots els càlculs es realitzen en un ordinador i no és necessari fabricar res. Moltes vegades, això implica l'ús de simulacions amb eines d'enginyeria assistida per ordinador (CAE) per a operacions d'anàlisi de mecànica estructural / elements finits, dinàmica de fluids computacional (CFD) o electromagnetisme (CEM). En general, aquestes simulacions requereixen una gran quantitat de càlculs numèrics a causa de la dificultat (o impossibilitat) de resoldre les equacions de manera analítica.

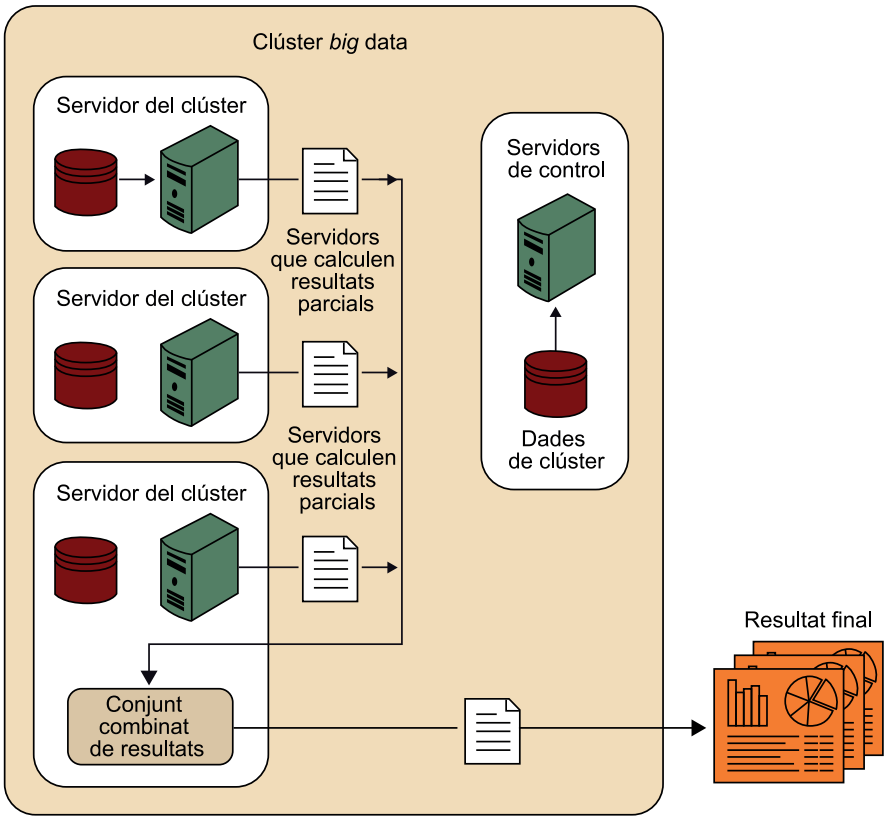
- **Anàlisi de dades.** Com hem vist, l'anàlisi de dades massives ajuda en gran manera a la presa de decisions de negoci. Un dels problemes dels mètodes d'aprenentatge automàtic o *machine learning* és que els mètodes més complexos, com les xarxes neuronals —també conegudes com a *deep learning*—, requereixen molt de temps de còmput per a ser entrenats. Això afecta la seva utilitat en escenaris en què el *time to market* és extremadament baix i, per tant, no es disposa d'una gran quantitat de temps per a entrenar les xarxes neuronals. Per sort, aquest tipus de mètodes es basen a entrenar una gran quantitat de neurones/perceptrons en paral·lel. En aquest punt les GPU ajuden a realitzar aquest entrenament molt més ràpidament gràcies a la seva arquitectura.

- **Optimització.** Una tasca d'optimització implica determinar els valors per a una sèrie de paràmetres de tal manera que, sota certes restriccions, se satisfaci alguna condició —per exemple, buscar els paràmetres d'una xarxa neuronal que minimitzi l'error de classificació, trobar els preus de renovació d'un conjunt d'assegurances que maximitzi el benefici en la renovació de les pòlisses, etc. Hi ha moltes tipus d'optimització, tant lineals com no lineals. El factor comú de tots aquests tipus és que requereixen utilitzar un conjunt d'algorismes, com el descens del gradient que exigeix un gran volum de càlculs repetitius que han de realitzar-se sobre un conjunt determinat de dades. Un cop més, ens trobem amb l'escenari de càlculs senzills repetits en paral·lel sobre un conjunt, possiblement molt gran, de dades.

2. Estructura general d'un sistema de dades massives

Encara que totes les infraestructures de dades massives tinguin característiques especifiques que adapten el sistema als problemes que han de resoldre, totes comparteixen alguns components comuns, tal com es descriu a la figura 2.

Figura 2. Exemple d'estructura general d'un possible sistema de dades massives



La primera característica comuna que cal destacar és que cada servidor del clúster posseeix el seu propi disc, memòria RAM i CPU. Això permet crear un sistema de còmput distribuït amb ordinadors heterogenis i de propòsit general, no dissenyats específicament amb la finalitat de crear clústers. Això redueix molt els costos d'aquests sistemes de computació, tant de creació com de manteniment. Tots aquests servidors es connecten mitjançant una xarxa local. Aquesta xarxa s'utilitza per comunicar els resultats que cada servidor calcula amb les dades que emmagatzema localment al seu disc dur. La xarxa de comunicacions pot ser de diferents tipus, des d'Ethernet a fibra òptica, depenent de com d'intensiu sigui l'intercanvi de dades entre els servidors.

Xarxa Ethernet

Ethernet és un estàndard de xarxes d'àrea local per a ordinadors amb accés al medi per detecció de l'ona portadora i amb detecció de col·lisions (CSMA/CD). El seu nom prové del concepte físic de l'èter.

A la figura 2 observem tres tipus de servidors:

- Els servidors que calculen resultats parcials. Aquests servidors s'ocupen de fer els càlculs necessaris per obtenir el resultat desitjat en les dades que emmagatzemen al seu disc dur.
- Els servidors que combinen els diferents resultats parcials per obtenir el resultat final desitjat. Aquests servidors són els encarregats d'emmagatzemar durant un temps els resultats finals.
- Els servidors de control o gestors de recursos, que asseguren que l'ús del clúster sigui correcte i que cap tasca no saturi els servidors. També s'encarreguen d'inspeccionar que els servidors funcionin sense errors. En cas que detectin un mal funcionament, forcen el reinici del servidor i avisen l'administrador de la presència de problemes en certs nodes.

L'ús combinat d'aquest conjunt de servidors és possible, com veurem en aquest mòdul, gràcies a l'ús d'un sistema de fitxers distribuït i al fet que els càlculs que s'han de realitzar s'implementen en un entorn de programació distribuïda, com per exemple Hadoop o Spark.

2.1. Estructura d'un servidor amb GPU

En un servidor equipat amb GPU la gestió de la memòria és una mica més complexa que en un servidor amb CPU, ja que per poder obtenir el màxim rendiment de tots els processadors d'una GPU és necessari garantir que tots aquests processadors reben un flux de dades constant durant l'execució de les tasques del clúster.

A la figura 3 descrivim les diferents memòries de què disposa un servidor amb GPU i els seus fluxos d'informació. Concretament, el processament en aquests equips implica l'intercanvi de dades entre HDD (disc dur), DRAM, CPU i GPU.

DRAM

DRAM correspon a les sigles de l'anglès *dynamic random access memory*, que significa 'memòria dinàmica d'accés aleatori' (o RAM dinàmica), i s'utilitza per denominar un tipus de tecnologia de memòria RAM basada en condensadors, els quals perden la seva càrrega progressivament i necessiten un circuit dinàmic de refresc que, cada cert període, revisa aquesta càrrega i la reposa en un cicle de refresc. En oposició amb aquest concepte sorgeix el de memòria SRAM (RAM estàtica), amb la qual es denomina el tipus de tecnologia RAM basada en semiconductors que, mentre segueixi alimentada, no necessita refrescar-se. DRAM és la tecnologia estàndard per a memòries RAM d'alta velocitat.

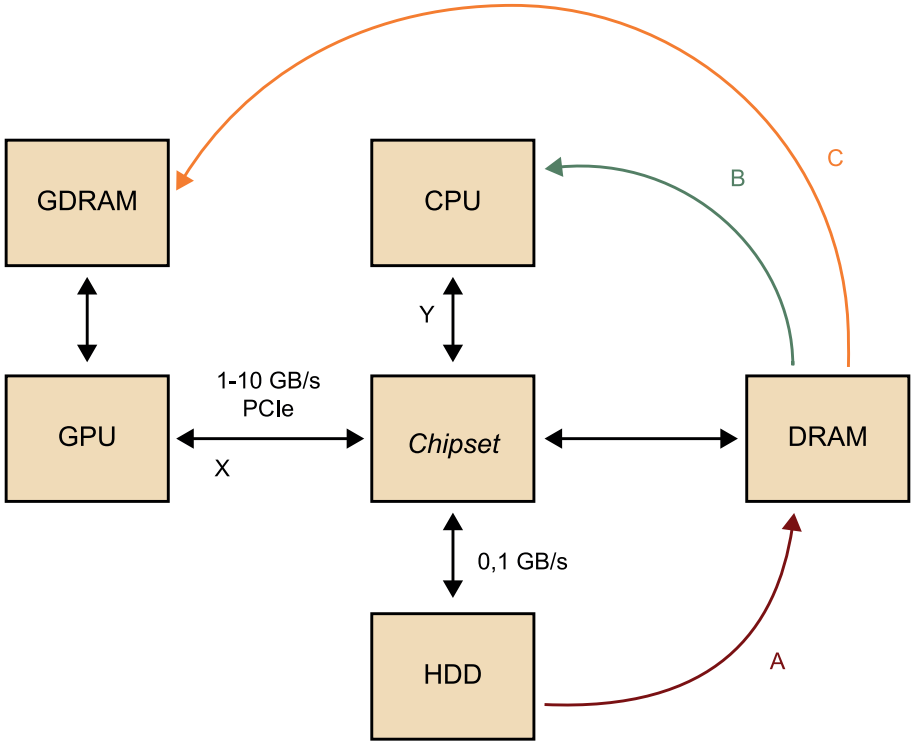
La figura 3 mostra com es transfereixen les dades quan un servidor realitza càlculs amb una CPU i una GPU. Si observem els diferents fluxos d'informació, veiem:

- **Fletxa A**. Transferència de dades d'un disc dur a la memòria principal (pas inicial comú tant per a la computació en CPU com en GPU).

- **Fletxa B**. Processament de dades amb una CPU (transferència de dades: DRAM —→ *chipset* —→ CPU).

- **Fletxa C**. Processament de dades amb una GPU (transferència de dades: DRAM —→ *chipset* —→ CPU —→ *chipset* —→ GPU —→ GDRAM —→ GPU).

Figura 3. Jerarquia de la memòria en un servidor equipat amb GPU



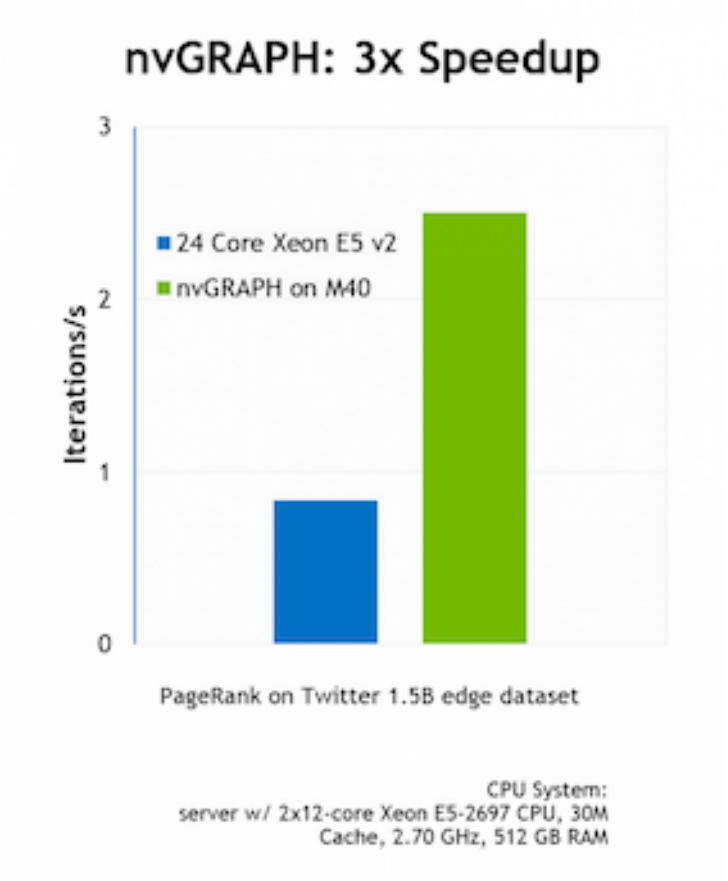
Com a resultat d'això, la quantitat total de temps que necessitem per completar qualsevol tasca inclou:

- La quantitat de temps requerit perquè una CPU o una GPU duguin a terme els seus càlculs.

- La quantitat de temps dedicat a la transferència de dades entre tots els components. Aquest punt és crucial en el cas de les GPU, tant per la quantitat de dades que han de transferir-se en paral·lel com per l'augment dels components intermedis necessaris.

És fàcil trobar a internet comparacions sobre els temps d'execució d'una tasca en GPU i en CPU. Per exemple, a la figura 4 podem observar els temps d'execució de l'algorisme de *PageRank* sobre una xarxa social. Atès que aquestes comparacions varien cada poc temps, aquí ens centrarem a descriure quin és l'impacte que té la transferència de dades en una GPU i així poder valorar si l'ús de GPU és viable o adequat.

Figura 4. Jerarquia de la memòria en un servidor equipat amb GPU



Font: NVIDIA

Tot i que és molt probable que quan fem servir un superordinador estigui optimitzat per treballar amb GPU, un servidor estàndard pot ser molt més lent quan intercanvia dades d'un tipus d'emmagatzematge a un altre. Mentre que la velocitat de transferència de dades entre una CPU estàndard i el *chipset* de còmput és de 10-20 GBps* (punt *Y* a la figura 3), una GPU intercanvia dades amb DRAM a una velocitat d'1-10 GBps (vegeu el punt *X* de la mateixa figura). Tot i que alguns sistemes poden aconseguir fins a uns 10 GBps (PCIe v3),** en la majoria de les configuracions estàndard els fluxos de dades entre una GPU (GDRAM) i la DRAM del servidor tenen una velocitat aproximada d'1 GBps.

*** GBps és el símbol de gigabytes per segon.**

**** PCIe v3 és l'abreviatura de *peripheral component interconnect express* de tercera generació.**

Per tant, encara que una GPU proporcioni una computació més ràpida, el principal coll d'ampolla és la velocitat de transferència de dades entre la memòria de la GPU i la memòria de la CPU (punt *X*). Per aquest motiu, per a cada projecte en particular cal mesurar el temps dedicat a la transferència de dades des d'una GPU o cap a una GPU amb el temps estalviat a causa de l'acceleració de la GPU. Així doncs, és millor avaluar el rendiment real en un petit conjunt de dades per després poder estimar com es comportarà el sistema en una escala major.

A partir del punt anterior podem concloure que, atès que la velocitat de transferència de dades pot ser bastant lenta, el cas d'ús ideal és quan la quantitat de dades d'entrada/sortida per cada GPU és relativament petita en compara-

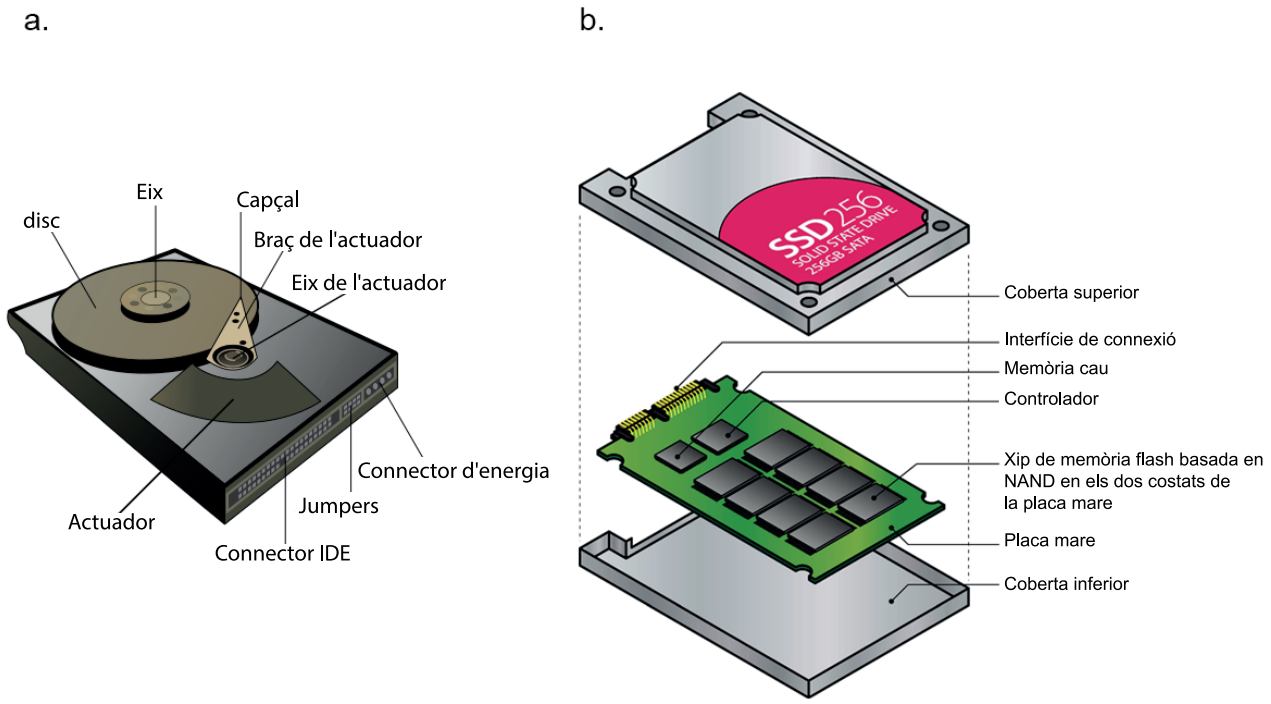
ció amb la quantitat de càlculs que s'han de realitzar. És important tenir en compte que, primer, el tipus de tasca ha de coincidir amb les capacitats de la GPU; i, segon, la tasca es pot dividir en subprocesos independents paral·lels com amb MapReduce. Aquest segon punt afavoreix la idea de poder realitzar una gran quantitat de càlculs en paral·lel com hem descrit a l'apartat anterior.

3. Sistema d'arxius

Una cop ja hem vist l'arquitectura general d'un sistema de dades massives i hem parlat de la diferència entre còmput en CPU i en GPU i de quines característiques han de tenir les diferents jerarquies de memòria en un servidor, és el moment de centrar-nos en com s'han d'emmagatzemar les dades per poder aplicar-hi càlculs.

El **sistema d'arxius** o **sistema de fitxers** és el component encarregat d'administrar i facilitar l'ús del sistema d'emmagatzematge, sigui basat en discos durs magnètics —en anglès, *hard disk drive* (HDD)— o en memòries d'estat sòlid —en anglès, *solid state disk* (SDD). En general, és estrany en sistemes de dades massives fer servir un tipus d'emmagatzematge terciari com DVD o CD-ROM, ja que aquests no permeten l'accés a la informació de manera distribuïda. A la figura 5 podem observar una comparació entre ambdues tecnologies.

Figura 5. Comparació entre un disc dur magnètic i una memòria d'estat sòlid



Breument, podem dir que un disc dur tradicional (figura 5a) es compon d'un o més plats o discos rígids units per un mateix eix que gira a gran velocitat dins d'una caixa metàl·lica segellada. Sobre cada plat, i en cadascuna de les seves cares, se situa un capçal de lectura/escriptura que sura sobre una prima làmina d'aire generada per la rotació dels discos. Per llegir o escriure infor-

mació primer s'ha de buscar la ubicació on realitzar l'operació de lectura o escriptura a la taula de particions situada al principi del disc. Després s'ha de posicionar el capçal al lloc adequat i, finalment, llegir o escriure la informació seqüencialment. Com que aquests dispositius d'emmagatzematge no disposen d'accés aleatori, se solen considerar un sistema d'emmagatzematge lent. En canvi, les memòries d'estat sòlid (figura 5b) sí que disposen d'accés aleatori a la informació emmagatzemada i, en general, són bastant més ràpides, tot i que tenen una vida útil relativament curta i que es redueix dràsticament si realitzem moltes operacions d'escriptura.

Tornant als sistemes d'arxius, diem que les seves principals funcions són l'assignació d'espai als arxius, l'administració de l'espai lliure i de l'accés a les dades emmagatzemades. Els sistemes d'arxius estructuren la informació emmagatzemada en un dispositiu d'emmagatzematge de dades. Aquesta informació després serà representada textualment o gràficament mitjançant un gestor d'arxius.

L'estructura lògica dels arxius sol representar-se de forma jeràrquica o en «arbre», una metàfora basada en la idea de carpetes i subcarpetes per organitzar els arxius amb algun tipus d'ordre. Per accedir a un arxiu s'ha de proporcionar la seva **ruta** (ordre jeràrquic de carpetes i subcarpetes) i el **nom** de l'arxiu seguit d'una **extensió** (per exemple, *.txt*) que indica el contingut de l'arxiu.

Exemple de ruta

home/user/mydata/data.csv és un exemple de la ruta completa amb el seu nom d'arxiu i l'extensió d'un fitxer de dades.

Quan treballem amb grans volums d'informació, un únic dispositiu d'emmagatzematge no és suficient i hem d'utilitzar sistemes d'arxius que permetin gestionar múltiples dispositius. Aquesta característica, tal com l'acabem de descriure, no és exactament el que necessitem; de fet, qualsevol sistema d'arxius modern permet emmagatzemar informació en diversos dispositius d'una manera més o menys transparent per a l'usuari. Realment el que necessitem és un sistema d'arxius que ens permeti gestionar múltiples dispositius distribuïts en diferents nodes (ordinadors) connectats entre ells utilitzant un sistema de xarxa.

Quan ens cal aquest nivell de distribució, no tots els sistemes d'arxius són una opció vàlida. Un **sistema d'arxius distribuït** o **sistema d'arxius de xarxa** és un sistema d'arxius d'ordinadors que serveix per compartir arxius, impressores i altres recursos com un emmagatzematge persistent en una xarxa d'ordinadors. El sistema NFS (de l'anglès *network file system*) va ser desenvolupat per Sun Microsystems l'any 1985 i és un sistema estàndard i multiplataforma que permet accedir i compartir arxius en una xarxa heterogènia com si estiguessin en un sol disc. Però, realment això és el que necessitem? La resposta és no. Aquest sistema ens dona la possibilitat d'accedir a una gran quantitat de dades d'una manera distribuïda, però sofreix un gran problema: les dades no estan emmagatzemades en el mateix lloc on s'han de realitzar els càlculs, la qual cosa provoca que cada vegada que hem d'executar un càlcul, les dades han de ser copiades per la xarxa d'un node a un altre. Aquest procés és lent i costós.

Per solucionar aquest problema es va crear el Hadoop Distributed File System (HDFS), que és un sistema d'arxius distribuït, escalable i portàtil per l'entorn de treball (*framework*) de càlcul distribuït Hadoop, tot i que actualment s'utilitza en gairebé tots els sistemes de dades massives (per exemple, Hadoop, Spark, Flink, Kafka, Flume, etc). Al subapartat següent introduirem el seu funcionament i els seus components principals.

3.1. Hadoop Distributed File System (HDFS)

HDFS és un sistema de fitxers **distribuït**, **escalable** i **portàtil** escrit en Java i creat especialment per treballar amb fitxers de gran grandària. Una de les seves característiques principals és una **grandària de bloc molt superior a l'habitual** per no perdre temps en els accessos de lectura. Els fitxers que normalment seran emmagatzemats o situats en aquest tipus de sistema de fitxers segueixen el patró ***Write once read many*** (que es pot traduir per 'escriu-ho una vegada, llegeix-ho moltes'). Per tant, està especialment indicat per a processos per lots de grans fitxers, els quals solament seran escrits una vegada i, per contra, seran llegits gran quantitat de vegades per poder analitzar-ne el contingut profundament.

Volum d'informació en HDFS

El volum mínim d'informació que es pot llegir en HDFS és de 64 MB, mentre que en els sistemes d'arxiu no distribuït aquest volum no sol superar els centenars de kB.

Procés per lots (*batch*)

Es coneix com a sistema per lots (*batch*) l'execució d'un programa sense el control o la supervisió directa de l'usuari. Aquest tipus de programes es caracteritza perquè la seva execució no requereix cap tipus d'interacció amb l'usuari.

Per tant, en HDFS tenim un sistema d'arxius distribuït i especialment optimitzat per a emmagatzemar grans quantitats de dades. D'aquesta manera, els fitxers seran dividits en blocs d'una mateixa grandària i distribuïts entre els nodes que formen el clúster de dades —els blocs d'un mateix fitxer se situaran en nodes diferents—; això ens facilitarà el còmput en paral·lel i ens evitarà desplaçar grans volums de dades entre diferents nodes d'una mateixa infraestructura de dades massives.

Logotip del Hadoop Distributed File System (HDFS)

Les arquitectures HDFS tenen dos tipus de nodes, diferenciats completament segons la funció que exerciran a l'hora de ser utilitzats. Els dos tipus de nodes HDFS són els següents:

- ***Namenode*** **(JobTracker)**. Aquest tipus de node, del qual només hi ha un per clúster, és el més important, ja que és responsable de la topologia de tots els altres nodes i, per tant, de gestionar l'espai de noms. L'espai de noms (*namespace*, en anglès) indica la ubicació (ruta) on es troben les dades. Concretament, indica el nom del *rack* o bastidor (i, més específicament, del *switch* o commutador) on està el node amb les dades.

- ***Datanodes*** **(TaskTracker)**. Aquest tipus de nodes, dels quals normalment hi haurà diversos, són els que realitzen l'accés a les dades pròpiament dit. En aquest cas, emmagatzemen els blocs d'informació i els recuperen sota demanda.

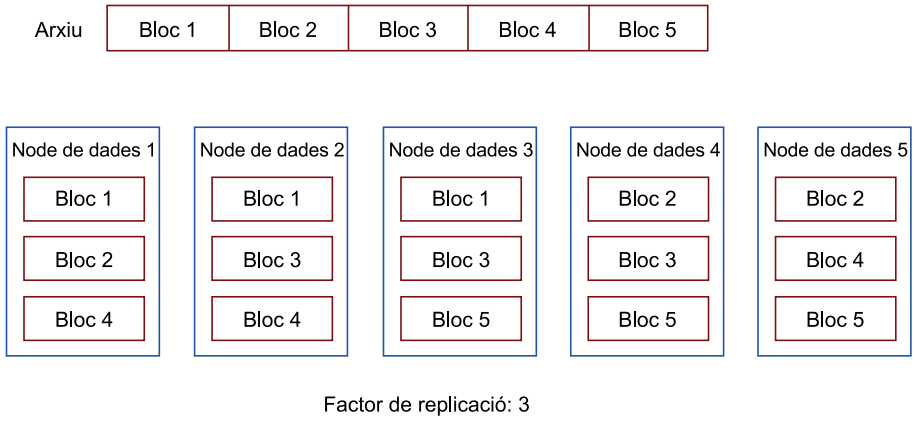
Simplificant, es pot considerar el JobTracker com el node principal o director d'orquestra, mitjançant el qual es distribuirà el tractament i processament dels

fitxers als TaskTracker, o nodes *worker*, que realitzaran la feina. Una tasca molt important del sistema de fitxers HDFS és definir correctament el nombre de rèpliques de cadascun dels arxius de dades. Aquest valor indica quantes còpies hi ha al clúster de cada fitxer: com més còpies hi hagi, menys necessitat de desplaçar dades entre els *datanodes*, però menys espai per a emmagatzemar dades. És un valor que ha de definir-se correctament.

Exemple d'emmagatzematge en HDFS

A la figura 6, s'ha definit el valor del nombre de rèpliques a 3. Aquest nombre permet que cada *datanode* posseeixi més del 50% de la informació, per tant, no ha de sol·licitar una gran quantitat d'informació als altres *datanodes*. D'aquesta manera permet que puguem acabar els processos en curs, fins i tot si fallen dos nodes dels cinc que disposa el clúster.

Figura 6. Exemple d'emmagatzematge en HDFS



3.2. Bases de dades NoSQL

Fins fa uns anys, les bases de dades relacionals han estat l'única alternativa als sistemes de fitxers per emmagatzemar grans volums d'informació. Aquest tipus de bases de dades utilitzen llenguatge de consulta estructurat (SQL) com a llenguatge de referència. Aquest tipus de bases de dades segueixen les regles ACID.* Aquestes propietats ACID permeten garantir que les dades són emmagatzemades de manera fiable i complint amb un conjunt de regles d'integritat definides sobre una estructura basada en taules que contenen files i columnes.

* http://bit.ly/2DPRlsv

Llenguatge SQL

SQL és l'acrònim en anglès d'*structured query language*. El SQL és un llenguatge declaratiu d'accés a bases de dades relacionals que permet especificar-hi diversos tipus d'operacions.

Regles ACID

En el context de bases de dades, ACID (acrònim anglès d'*atomicity, consistency, isolation, durability*) són una sèrie de propietats que ha de complir tot sistema de gestió de bases de dades per garantir que les transaccions siguin fiables.

No obstant això, amb els requeriments del món de les dades massives ens trobem que les bases de dades relacionals no poden gestionar la grandària, la complexitat dels formats o la velocitat de lliurament de les dades que requereixen moltes aplicacions. Un exemple d'aplicació seria Twitter, on milions d'usuaris accedeixen al servei concurrentment tant per consultar com per generar noves dades.

Aquestes noves aplicacions han propiciat l'aparició de nous sistemes de bases de dades, anomenats NoSQL, que permeten donar una solució als reptes d'escalabilitat i rendiment que representen les dades massives.

El concepte NoSQL agrupa diferents solucions per a emmagatzemar diferents tipus de dades, des de taules a grafs, passant per documents, imatges o qualsevol altre format. Qualsevol base de dades NoSQL és distribuïda i escalable per definició. Hi ha nombrosos productes disponibles, molts d'ells de codi obert, com Cassandra, que basa el seu sistema de funcionament a emmagatzemar la informació en columnes, en lloc de files, i generen un conjunt d'índexs associatius que li permeten recuperar grans blocs d'informació en un temps molt reduït.

Les bases de dades NoSQL no pretenen substituir les bases de dades relacionals, sinó que simplement aporten solucions alternatives que milloren el rendiment dels sistemes gestors de bases de dades per a determinats problemes i aplicacions. Per aquest motiu, NoSQL també s'associa al concepte ***not only SQL***. NoSQL no prohibeix el llenguatge estructurat de consultes. Si bé és cert que alguns sistemes NoSQL són totalment no relacionals, uns altres simplement eviten funcionalitats relacionals concretes com esquemes de taules fixes o certes operacions de l'àlgebra relacional. Per exemple, en lloc d'utilitzar taules, una base de dades NoSQL podria organitzar les dades en objectes, parells clau-valor o fins i tot tuples seqüencials.

El principi que segueix aquest tipus de bases de dades és el següent: com que en determinats escenaris no és possible utilitzar bases de dades relacionals, no queda cap altre remei que relaxar alguna de les limitacions inherents d'aquests tipus de sistemes d'emmagatzematge. Per exemple, podem pensar en col·leccions de documents amb camps no definits estrictament, que fins i tot poden anar canviant amb el temps, en lloc de taules amb files i columnes amb un format prefixat. En certa manera, fins i tot podríem arribar a pensar que un sistema d'aquest tipus no és ni tan sols una base de dades entesa com a tal, sinó un sistema d'emmagatzematge distribuït per a gestionar dades dotades d'una certa estructura que pot ser extremadament flexible.

HashMap

Un HashMap és una col·lecció d'objectes, com un vector o *arrays*, però sense ordre. Cada objecte s'identifica mitjançant algun identificador apropiat. El nom *hash* fa referència a una tècnica d'organització d'arxius anomenada *hashing* o dispersió en el qual s'emmagatzemen els registres en una adreça que és generada per una funció que s'aplica sobre la clau del registre.

En general, hi ha quatre tipus de bases de dades NoSQL, depenent de com emmagatzemen la informació:

- **Clau-valor.** Aquest format és el més típic. Podem entendre'l com un HashMap en què cada element està identificat per una clau única, la qual cosa permet la recuperació de la informació de manera molt ràpida. Normalment el valor s'emmagatzema com un objecte binari i el seu contingut no és important per al clúster.

- **Basada en documents**. Aquest tipus de base de dades emmagatzema la informació com un document —generalment amb una estructura simple com JSON o XML— i amb una clau única. És similar a les bases de dades clau-valor, però amb la diferència que el valor és un fitxer que pot ser entès pel clúster i pot realitzar operacions sobre els documents.

- **Orientades a grafs**. Hi ha altres bases de dades que emmagatzemen la informació com a grafs, en què les relacions entre els nodes són el més important. Són molt útils per representar informació de xarxes socials.

- **Orientades a columnes**. Guarden els valors en columnes en lloc de files. Amb aquest canvi guanyem molta velocitat en lectures, ja que si es requereix consultar un nombre reduït de columnes, és molt ràpid fer-ho. La principal contrapartida és que no és eficient per realitzar escriptures.

4. Sistema de càlcul distribuït

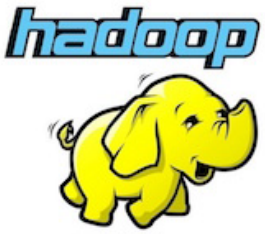
Els sistemes de càlcul distribuït permeten la integració dels recursos de diferents màquines en xarxa i converteixen la ubicació d'un recurs en alguna cosa transparent a l'usuari. L'usuari accedeix als recursos del sistema distribuït per mitjà d'un gestor de recursos i es despreocupa d'on es troba aquest recurs i de quan el podrà fer servir. En aquest mòdul ens centrarem a descriure dos sistemes de càlcul distribuït diferents i molt estesos en l'ecosistema de les dades massives: MapReduce i Spark. A més, també introduirem com es poden aplicar les idees del càlcul distribuït sobre arquitectures basades en GPU.

4.1. Paradigma MapReduce

MapReduce és un model de programació introduït per Google l'any 1995 per donar suport a la computació paral·lela sobre grans volums de dades, amb dues característiques principals:

1) la utilització de clústers d'ordinadors

2) la utilització de maquinari no especialitzat



Logotip d'Apache Hadoop

El nom d'aquest sistema està inspirat en els noms dels seus dos mètodes o funcions de programació principals: *Map* i *Reduce*, que veurem a continuació. MapReduce ha estat adoptat mundialment, gràcies al fet que existeix una implementació de codi obert denominada Hadoop, desenvolupada per Yahoo, que permet fer servir aquest paradigma utilitzant el llenguatge de programació Java.

El primer que hem de tenir en compte quan parlem d'aquest model de càlcul és que no totes les anàlisis es poden calcular amb aquest paradigma. Concretament, només són aptes aquelles que poden calcular-se com a combinacions de les operacions de `Map()` i de `Reduce()`. Les funcions *Map* i *Reduce* estan definides ambdues pel que fa a dades estructurades en tuples del tipus <*clau*, *valor*>. En general, aquest sistema funciona molt bé per calcular agregacions, filtres, processos de manipulació de dades, estadístiques, etc., operacions fàcils de paralel·litzar que no requereixen un processament iteratiu i en les quals no és necessari compartir les dades entre tots els nodes del clúster.

En l'arquitectura MapReduce tots els nodes es consideren *workers* (en català, 'treballadors'), excepte un, que pren el rol de *master* (en català, 'mestre'). El

mestre s'encarrega de recopilar treballadors en repòs, és a dir, sense tasca assignada, i els assigna una tasca específica de `Map()` o de `Reduce()`. Un *worker* només pot tenir tres estats: repòs, treballant i complet. El rol de mestre s'assigna de manera aleatòria en cada execució.

Ara vegem amb una mica més de detall com funcionen les dues operacions bàsiques de MapReduce:

- La funció `Map()` és una funció associativa que s'aplica en paral·lel a tots els elements del conjunt de dades d'entrada. En aquest punt és molt important el concepte d'associativitat, ja que no podem establir un ordre concret en la finalització de les diferents funcions `Map()`.

  Com a resultat retorna una llista de parells <*clau*, *valor*> per a cada crida. Una vegada s'ha calculat aquesta associació clau-valor, el clúster agrupa els parells amb la mateixa clau de totes les llistes i crea un grup per a cadascuna de les diferents claus generades. Des del punt de vista de l'arquitectura, el node mestre pren les dades d'entrada, les divideix en petites peces, o problemes de menys complexitat, i els distribueix als nodes *worker*. Al seu torn, un node *worker* pot tornar a subdividir les dades que li han arribat, fet que dona lloc a una estructura en forma d'arbre. Una vegada s'ha acabat la divisió de les dades, els nodes *worker* processen els subproblemes i passen les respostes al node mestre.

- La funció `Reduce()` produeix una crida buida, un valor o fins i tot una llista de valors en cada crida. La tornada de totes aquestes crides, independentment de si han produït o no un resultat, es recull com la llista de resultat final desitjat.

En conseqüència, una execució MapReduce transforma una llista de parells <*clau*, *valor*> en una llista de valors. La funció `Map()` s'executa en paral·lel de manera distribuïda en cadascun dels nodes *worker* del clúster. Com hem vist a l'apartat 3, les dades d'entrada que es troben emmagatzemades en HDFS, es divideixen en un conjunt d'M particions d'entrada. Aquestes particions són processades en diversos nodes. Com es descriu a la figura 7, en una crida de MapReduce solen ocórrer les operacions següents:

- Es divideixen les dades d'entrada amb relació al nombre de *workers* disponibles en aquest moment.

- Es decideix quin dels nodes serà el node mestre, la resta de nodes seran considerats *workers*. El node mestre s'encarrega de buscar els nodes *worker* en repòs (sense tasca assignada) i li assignarà una tasca específica de `Map()` o de `Reduce()`.

- Un *worker* que rebi una tasca de `Map()` usarà com a entrada la partició que li correspongui i analitzarà els parells <*clau*, *valor*> per crear una nova

parella de sortida, tal com estigui definit dins de la funció `Map`. Els parells *clau* i *valor* produïts s'emmagatzemen a la memòria temporal (*buffer*).

- Cada cert temps, els parells clau-valor emmagatzemats a les memòries temporals o *buffers* dels *workers* s'escriuen al disc local, distribuïts en *R* regions. Aquestes regions són enviades al node mestre, que s'encarrega de reexpedir les noves dades als nodes *worker* que tinguin assignades tasques de `Reduce()`.

- Quan el node mestre notifica a un node *worker* de tipus `Reduce` la localització d'una partició, aquest fa servir una sèrie de crides remotes per a fer lectures de la informació emmagatzemada als discos durs dels *workers* de tipus `Map()`. Quan el *worker* de tipus `Reduce()` llegeix totes les dades, les agrupa fent servir la informació continguda a les claus de manera que s'agrupin les dades que posseeixen la mateixa clau. El pas és necessari perquè moltes claus de funcions `Map()` diverses poden anar a una mateixa funció `Reduce()`.

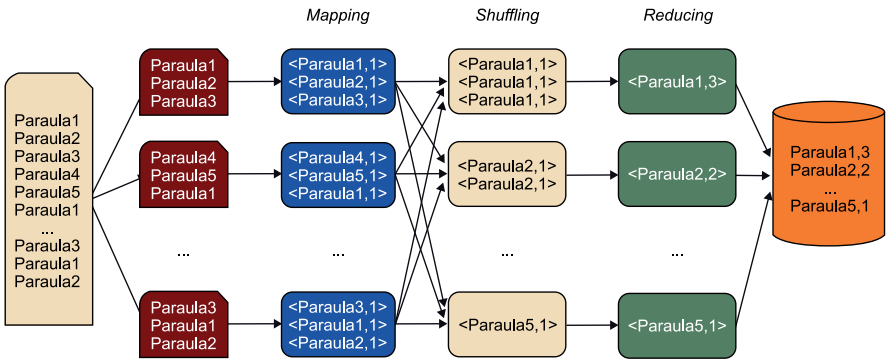
- Els nodes *worker* de tipus `Reduce()` iteren sobre el conjunt de valors ordenats intermedis per a cadascuna de les claus úniques oposades. Per poder dur a terme aquest pas, és necessari que la funció `Reduce()` conegui el conjunt de valors associats a cada clau. La sortida d'aquesta funció s'afegeix al fitxer de sortida de l'execució.

- Una vegada totes les tasques `Map()` i `Reduce()` s'han completat, el node mestre finalitza l'execució i retorna el control a l'usuari.

Exemple: recompte de paraules

Observem la figura 7. En aquest cas la funció `Map()` s'ocupa de convertir cada paraula en una estructura <*clau*, *valor*>, on la clau serà la pròpia paraula i el valor serà igual a 1. Posteriorment, s'agrupen aquestes estructures clau-valor que comparteixen la mateixa clau en un mateix node. Finalment, la funció `Reduce()` s'ocupa de sumar tots els valors —en aquest cas tots seran igual a 1— i de retornar una nova estructura clau-valor on s'emmagatzema la paraula i el seu nombre d'aparicions. Això pot serveix per executar un nou càlcul més complex sobre les paraules o, com a l'exemple, per retornar un llistat amb el resultat a l'usuari.

Figura 7. Exemple de procés en MapReduce



Un dels aspectes més importants d'aquesta manera de realitzar càlculs és que és **tolerant a errors**. Quan en un dels nodes *workers* es produeix un error, el node mestre se n'adona, ja que periòdicament fa una sol·licitud d'estatus a cadascun dels nodes *worker*. Si la comprovació de l'estatus no és correcta, es cancel·la el treball assignat a aquest node *worker* i es reassigna a un altre node perquè el realitzi.

4.2. Apache Spark: processament distribuït en memòria principal

Com hem descrit al subapartat 4.1., MapReduce únicament és capaç de distribuir el processament als servidors copiant les dades que s'han de processar per mitjà del seu disc dur. Aquesta limitació redueix el rendiment dels càlculs iteratius, on les mateixes dades han d'usar-se una vegada i una altra. Aquest tipus de processament és bàsic en la majoria d'algorismes d'aprenentatge automàtic.

Logotip d'Apache Spark

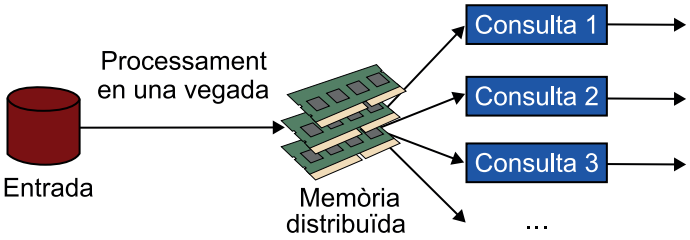
El projecte de Spark es va centrar des del començament a aportar una solució factible per a aquests defectes de Hadoop, millorant el comportament de les aplicacions que fan ús de MapReduce i augmentant el seu rendiment considerablement.

Spark és un sistema de càlcul distribuït per al processament de grans volums de dades i que gràcies a la seva anomenada «interactivitat» fa que el paradigma MapReduce ja no es limiti a les fases `Map()` i `Reduce()` i puguem realitzar més operacions com *mappers*, *reducers*, *joins*, *groups by*, filtres, etc. Spark proporciona API per a Java, Scala i Python, tot i que és preferible que es programi en Scala, ja que és el seu llenguatge natiu.

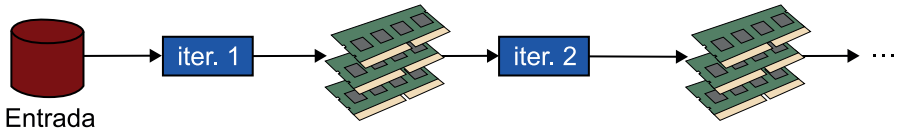
API

Una interfície de programació d'aplicacions (API) és el conjunt de rutines, funcions i procediments (o mètodes, en la programació orientada a objectes) que ofereix una biblioteca de programació perquè pugui ser utilitzada per un altre programari com una capa d'abstracció.

Figura 8. Comparació d'una execució de Spark i Hadoop



a. Baixa latència computacional mitjançant l'anàlisi de les dades en memòria



b. Algoritmes iteratius

El principal avantatge de Spark respecte a Hadoop és que guarda en memòria totes les operacions sobre les dades. Aquesta és la clau del seu bon rendiment. La figura 8 mostra algunes de les seves característiques principals:

- Baixa latència computacional mitjançant l'anàlisi de les dades en memòria (figura 8a).
- Els algorismes iteratius s'executen de manera eficient gràcies al fet que les operacions successives comparteixen les dades en memòria (figura 8b).

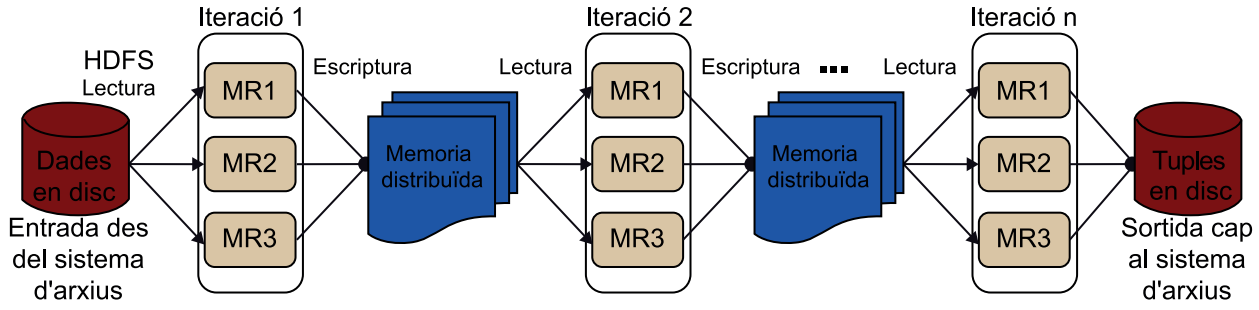
A la figura 9 s'il·lustra com es realitza l'execució d'un programa en Spark. Una execució típica de Spark s'organitza de la manera següent:

1) A partir d'una variable d'entorn anomenada *spark context* que defineix com està organitzat el clúster, es crea un objecte RDD —que emmagatzema un conjunt de dades en memòria principal de manera distribuïda— llegint dades del sistema de fitxers (que podria ser perfectament HDFS), una base de dades o qualsevol altra font d'informació en forma de llista.

2) Una vegada creat l'RDD inicial es realitzen transformacions per crear més objectes RDD a partir del primer. Aquestes transformacions s'expressen en termes de programació funcional i no eliminen l'RDD original, sinó que creen un nou RDD en cada operació.

3) Després de realitzar les accions i transformacions necessàries sobre les dades, els objectes RDD han de convergir per crear l'RDD final. Aquest RDD s'acaba emmagatzemant al sistema d'arxius del clúster o en qualsevol altre lloc que ofereixi persistència.

Figura 9. Fluxe d'execució de Spark



4.2.1. Resilient Distributed Dataset (RDD)

A Spark, a diferència de Hadoop, no utilitzarem una col·lecció de dades distribuïdes al sistema d'arxius, sinó que farem servir els Resilient Distributed Datasets (RDD).

Enllaç d'interès

Per a tenir informació més detallada sobre RDD consulteu l'adreça web següent: http://bit.ly/1ajlZop.

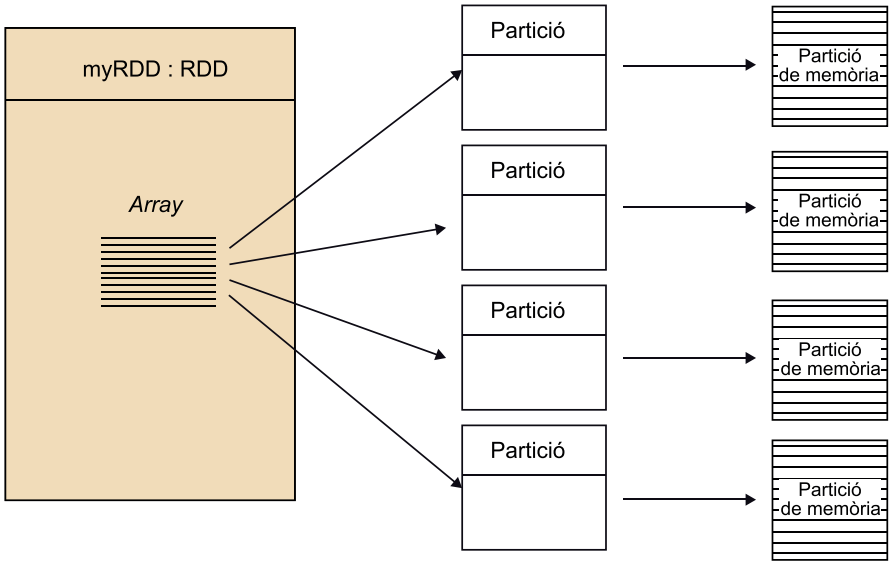
Els RDD són col·leccions lògiques, immutables i particionades de registres de dades distribuïdes a la memòria principal dels nodes del clúster i que poden ser

reconstruïdes si alguna partició es perd.* Es creen mitjançant la transformació de les dades utilitzant transformacions (*filters*, *joins*, *group by*, etc). D'altra banda, permet analitzar, és a dir, guardar, les dades mitjançant accions com *reduce*, *collect*, *take*, *cache*, *persist*, etc.

*** No necessiten ser materialitzades però sí reconstruïdes per mantenir l'emmagatzematge estable.**

Els RDD, descrits a la figura 10, són tolerants a errors, per la qual cosa guarden un registre de les transformacions realitzades, anomenat el llinatge (*lineage*, en anglès) dels RDD. Aquest llinatge permet que els RDD es reconstrueixin en cas que una porció de dades es perdi per una fallada en un node del clúster.

Figura 10. Representació en memòria principal d'un RDD de Spark



Gràcies a això, els RDD ens proporcionen els beneficis següents:

- La consistència es torna més senzilla gràcies a la propietat d'immutabilitat.
- Obtenim la tolerància a errors amb un baix cost, gràcies a la idea del llinatge i al fet de poder generar punts de control (*checkpoints*) gràcies a generar accions —`cache()` i `persist()`, concretament— sobre els RDD que permeten al programador guardar els resultats a la memòria RAM o en el disc dur, respectivament.
- Malgrat ser un model restringit a una sèrie de casos d'ús per defecte, gràcies a la flexibilitat dels RDD es pot utilitzar Spark per a una quantitat d'aplicacions molt variades en què MapReduce obté un rendiment molt baix.

4.3. Hadoop i Spark: un mateix origen

Tant Hadoop com Spark estan escrits en Java. Aquest factor comú no és una simple casualitat, sinó que té un explicació molt senzilla: tots dos ecosistemes

usen els *remote method invocation* (RMI) de Java per poder comunicar-se de manera eficient entre els diferents nodes del clúster.

RMI és un mecanisme ofert per Java que serveix per invocar un mètode de manera remota. Forma part de l'entorn estàndard d'execució de Java i proporciona un mecanisme simple per a la comunicació de servidors en aplicacions distribuïdes basades exclusivament en Java.

RMI es caracteritza per la facilitat del seu ús a la programació, ja que està específicament dissenyat per a Java; proporciona pas d'objectes per referència, recol·lecció d'escombraries distribuïdes (*garbage collector* distribuït) i pas de tipus arbitraris. Mitjançant RMI, un programa Java pot exportar un objecte, de manera que aquest objecte serà accessible per mitjà de la xarxa i el programa romandrà a l'espera de peticions en un port TCP. A partir d'aquest moment, un client pot connectar-se i invocar els mètodes proporcionats per l'objecte. Això és justament el que fa el node mestre a MapReduce per enviar funcions `Map()` o `Reduce()` als nodes *worker*. Spark utilitza el mateix sistema per enviar les transformacions i accions que s'han d'aplicar als RDD.

4.4. GPU: simplicitat i alta paralel·lització

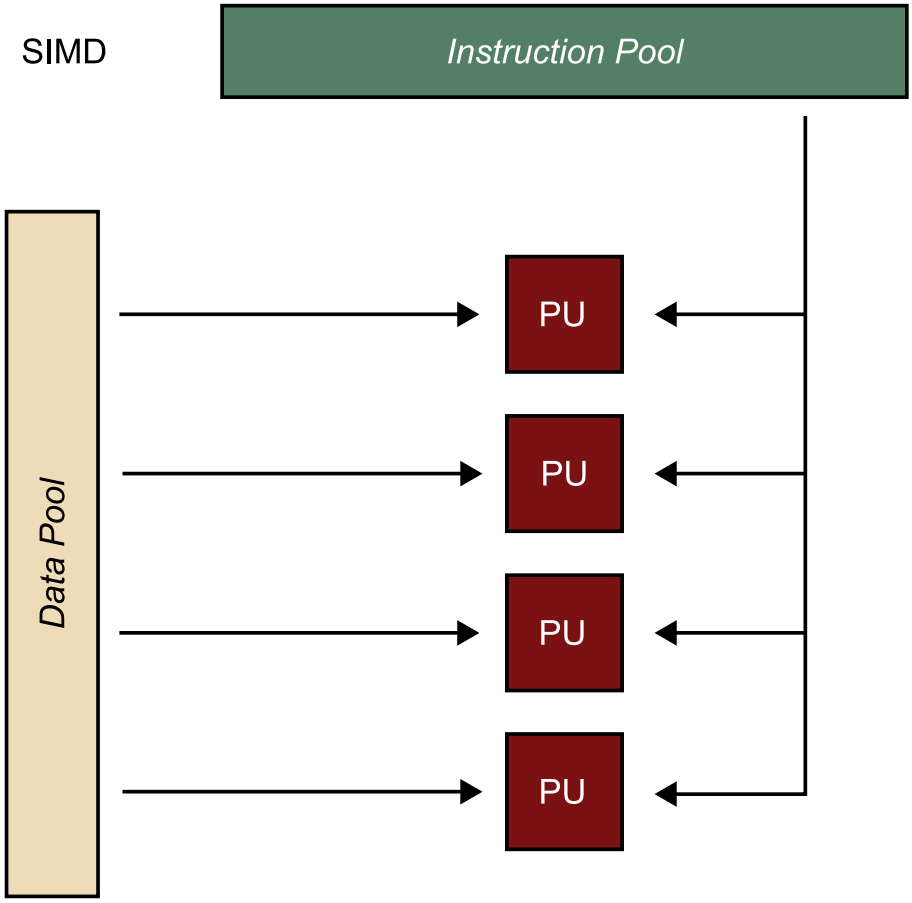
Com hem comentat en apartats anteriors, els servidors amb GPU es basen en arquitectures ***many-core***. Aquestes arquitectures disposen d'una quantitat ingent de nuclis (processadors) que realitzen una mateixa operació sobre múltiples dades.

En computació això es coneix com SIMD (*single instruction, multiple data*). SIMD és una tècnica utilitzada per aconseguir un gran nivell de paral·lelisme a nivell de dades. Per aquest motiu en subapartats anteriors ens hem ocupat de descriure com s'organitzen els fluxos de dades dins d'un servidor amb GPU, ja que l'alta disposició de dades en els registres dels múltiples processadors és un element essencial per poder obtenir millores en el rendiment d'aplicacions basades en GPU.

Els repertoris SIMD consisteixen en instruccions que apliquen una mateixa operació sobre un conjunt més o menys gran de dades. És una organització en què una única unitat de control comú despatxa les instruccions a diferents unitats de processament. Totes aquestes reben la mateixa instrucció, però operen sobre diferents conjunts de dades. És a dir, la mateixa instrucció és executada de manera sincronitzada per totes les unitats de processament. La figura 11 descriu de manera gràfica aquesta idea.

L'ús d'aquestes arquitectures fa possible definir la idea de **GPGPU** o computació de propòsit general sobre processadors gràfics. Això ens permet realitzar el còmput d'aplicacions que tradicionalment s'executaven sobre CPU en GPU d'una manera molt més ràpida.

Figura 11. Estructura funcional del concepte «Una instrucció, múltiples dades»



Font: https://es.wikipedia.org/wiki/SIMD

Llenguatges de programació com CUDA, desenvolupat per NVDIA, permeten utilitzar una plataforma de computació paral·lela amb GPU per dur a terme computació general a una gran velocitat. Amb llenguatges com CUDA, enginyers i desenvolupadors poden accelerar les aplicacions informàtiques mitjançant l'aprofitament de la potència de les GPU d'una manera més o menys transparent, ja que encara que CUDA es basi en el llenguatge de programació C, disposa d'API i llibreries per a una gran quantitat de llenguatges, com per exemple, Python o Java.

Logotip d'NVIDIA

5. Gestor de recursos

Un gestor de recursos distribuïts és un sistema de gestió de cues de tasques. Permet que diversos usuaris, grups i projectes puguin treballar junts fent servir una infraestructura compartida, com, per exemple, un clúster de computació.

Cues de tasques

Una cua no es més que un sistema per executar les tasques en un cert ordre aplicant-hi una sèrie de polítiques de priorització.

5.1. Apache Mesos

Apache Mesos* és un gestor de recursos que simplifica la complexitat de l'execució d'aplicacions en un conjunt compartit de servidors. Originalment, Mesos va ser construït com un sistema global de gestió de recursos i era completament agnòstic sobre els programes i serveis que s'executen al clúster.

* http://mesos.apache.org

Amb aquesta idea, Mesos ofereix una capa d'abstracció entre els servidors i els recursos. És a dir, bàsicament el que ens ofereix Mesos és un lloc on executar aplicacions sense preocupar-nos dels servidors que tenim per sota. Seguint la manera de funcionar de MapReduce, dins d'un clúster de Mesos tindrem un únic node mestre (del qual podem tenir rèpliques inactives) que s'ocuparà de gestionar totes les peticions de recursos que rebi el clúster. La resta de nodes seran nodes esclaus *slaves*, que són els encarregats d'executar les tasques dels entorns d'execució (per exemple, Spark, Hadoop...). Aquests nodes reporten el seu estat directament al node mestre actiu. És a dir, el node mestre també s'encarrega del seguiment i control de les tasques en execució.

5.2. YARN (Yet Another Resource Negotiator)

YARN (Yet Another Resource Negotiator)** neix per donar solució a una idea fonamental: dividir les dues funcions principals del JobTracker. És a dir, tenir en serveis o dimonis totalment separats i independents la gestió de recursos d'una banda i, per una altra, la planificació i el monitoratge de les tasques o execucions.

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

***Deamon* o servei**

Un *daemon*, 'dimoni', (nomenclatura utilitzada en sistemes UNIX i UNIX-like) o servei (nomenclatura usada en Windows) és un tipus especial de procés informàtic no interactiu que s'executa en segon pla en comptes de ser controlat directament per l'usuari.

Un algorisme de MapReduce per si sol no és suficient per a la majoria d'anàlisis que Hadoop pot arribar a resoldre. Amb YARN, Hadoop disposa d'un entorn de gestió de recursos i aplicacions distribuïdes on es poden implementar múltiples aplicacions de processament de dades totalment personalitzades i específiques per realitzar una gran quantitat d'anàlisis concurrentment.

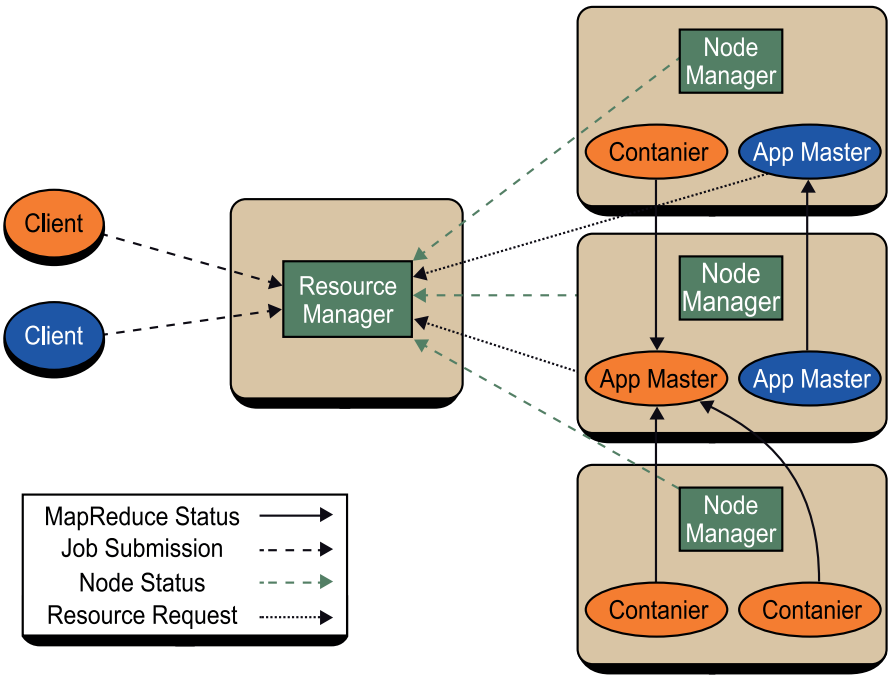
D'aquesta separació sorgeixen dos elements:

- **ResourceManager (RM)**. Aquest element és global i s'encarrega de tota la gestió dels recursos.

- **ApplicationMaster (AM)**. Aquest element és específic de cada aplicació i s'encarrega de la planificació i del monitoratge de les tasques.

Això deriva en el fet que ara una aplicació és simplement un *Job* (en el sentit tradicional de MapReduce). A la figura 12 s'il·lustra l'arquitectura de YARN per poder entendre-la d'una manera més clara.

Figura 12. Exemple d'arquitectura a YARN



Font: https://hadoop.apache.org/docs/

D'aquesta manera, el ResourceManager i el NodeManager (NM) esclau de cada node formen l'entorn de treball, en què el ResourceManager s'encarrega de repartir i gestionar els recursos entre totes les aplicacions del sistema, mentre que l'ApplicationMaster s'encarrega de la negociació de recursos amb el ResourceManager i els NodeManager per poder executar i controlar les tasques, és a dir, els sol·licita recursos per poder treballar.

6. Escenaris de processament distribuït

Ara descriurem els tres escenaris de processament de dades distribuïdes més comuns. El processament de grans volums d'informació en lots (*batch*), el processament de dades en flux (*stream*) i el processament de dades amb targetes gràfiques (GPU). Encara que tots aquests escenaris s'incloguin en els entorns de dades massives, tenen diferències importants que provoquen que no puguin resoldre's de la mateixa manera.

6.1. Processament en lots o *batch*

Un sistema per lots (en anglès, *batch processing*) es refereix a l'execució d'un programa sense el control o la supervisió directa de l'usuari. Aquests tipus de programes es caracteritzen perquè la seva execució no requereix cap tipus d'interacció amb l'usuari.

Per norma general, aquests tipus d'execucions s'utilitzen en tasques repetitives sobre grans conjunts d'informació. Un exemple seria el processament de logs d'un servidor web. En aquest cas, cada nit es processarien els fitxers de logs generats pels servidors web durant el dia. Aquest processament en lots es pot realitzar de forma senzilla utilitzant MapReduce, ja que el coneixement que ens interessa obtenir dels logs són valors acumulats, estadístiques i/o càlculs que es combinen amb els resultats obtinguts en els dies anteriors. En aquest cas no hi ha cap tipus de processament iteratiu de les dades.

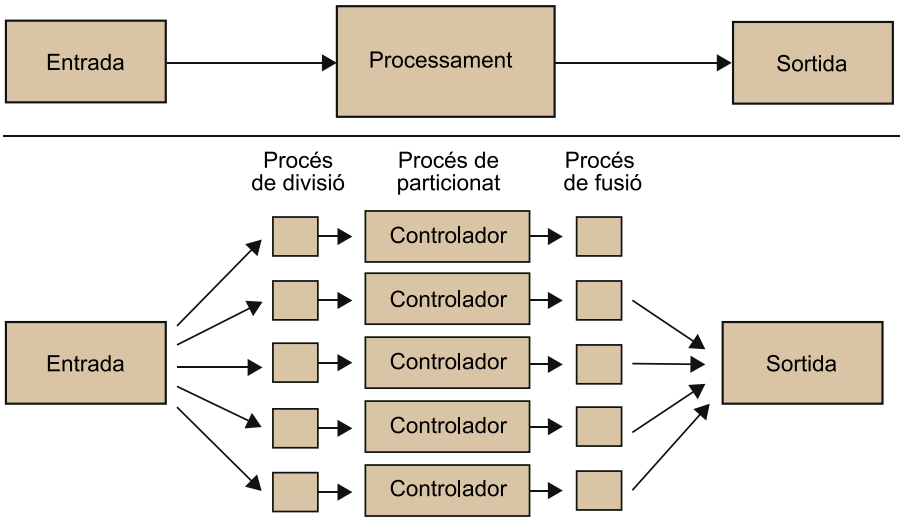
Fitxer de logs

Un fitxer de logs, o 'bitàcora' en català, és un fitxer seqüencial que emmagatzema tots els esdeveniments que ha processat un servidor. Normalment es guarden en un format estàndard per poder ser processats de manera senzilla.

El processament en lots requereix l'ús de diferents tecnologies per a l'entrada de les dades, el seu processament i la seva sortida. En el processament en lots es pot dir que Hadoop ha estat l'eina estrella que ha permès emmagatzemar quantitats enormes de dades i escalar-les horitzontalment, afegint més nodes de processament en el clúster.

A la figura 13 observem l'estructura típica del processament per lots. En la part superior de la figura observem com es realitzaria el processament sense un sistema de càlcul distribuït, mentre que en la part inferior podem observar com es realitzaria fent servir el paradigma de programació basat en MapReduce.

Figura 13. Exemple d'aplicació d'un procés en lots



6.2. Processament de dades en flux o *stream*

El processament de dades en flux és un tipus de tècnica de processament i l'anàlisi de dades que es basa en la implementació d'un model de flux de dades en el qual les dades associades a sèries de temps (fets) flueixen contínuament per mitjà d'una xarxa d'entitats de transformació que componen el sistema. En general, i tret que necessitem fer el processament i l'anàlisi de dades en temps real, s'assumeix que no hi ha limitacions de temps obligatòries en el processament de dades en flux.

Exemple

Intentar determinar quins clients dels que estan accedint a una botiga en línia en un moment precís tenen més probabilitat de comprar. Una vegada s'ha determinat quins són, s'afegeix una marca en la seva sessió web i se'ls assigna una prioritat més alta perquè gaudeixin de més recursos del servidor web. Això provocarà que la seva sessió sigui més ràpida i, per tant, la seva experiència de compra en la tenda en línia sigui millor i augmenti fint i tot més la seva probabilitat de compra.

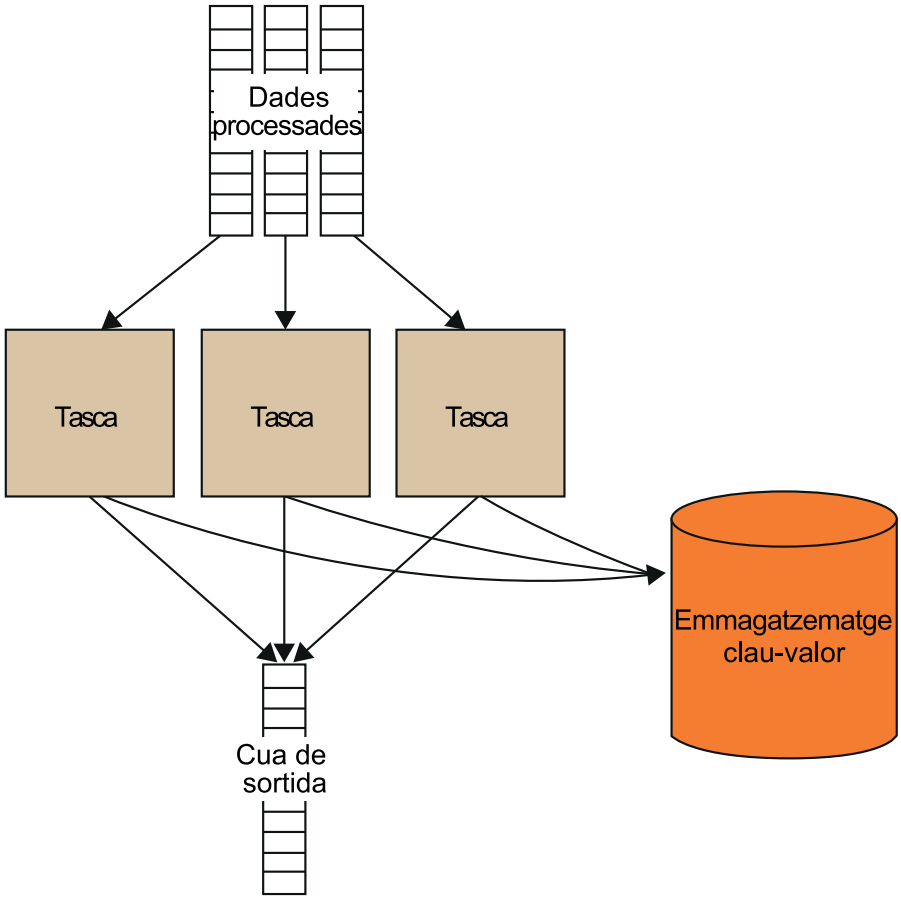
Generalment, les úniques limitacions d'aquests sistemes són les següents:

- S'ha de disposar de prou memòria per emmagatzemar les dades d'entrada en cua.

- La taxa de productivitat del sistema a llarg termini hauria de ser més ràpida, o almenys igual, a la taxa d'entrada de dades en aquest mateix període. Si això no fos així, els requisits d'emmagatzematge del sistema creixerien sense límit.

Aquest tipus de tècniques de processament i anàlisi de dades no està destinat a analitzar un conjunt complet de grans dades, sinó una part. A la figura 14 observem un esquema senzill de processament de dades en flux en què les dades processades es guarden tant en una cua de sortida com en un format clau-valor perquè puguin ser processades posteriorment en lots.

Figura 14. Exemple d'aplicació de procés de dades en flux



6.3. Processament en GPU

Un aspecte fonamental de les actuals GPU és l'ús de petits processadors (coneguts també com a *unified shaders*). Aquests processadors són molt senzills i executen un conjunt d'instruccions molt concret que realitza operacions aritmètiques bàsiques, però el seu creixement en la integració (nombre de processadors) en cada nova generació de GPU és significatiu. Actualment, a les targetes gràfiques modernes ens trobem amb un nombre d'entre mil i quatre mil processadors.

El problema d'aquesta integració és precisament que no totes les tasques han de ser més eficients en una GPU. Les GPU estan especialitzades en tasques altament paralel·lizables amb algorismes que poden subdividir-se i processar-se per separat per, després, unir els subresultats i obtenir el resultat final. Típicament són problemes científics, matemàtics o simulacions, tot i que un exemple molt més popular és la problemàtica de minar bitcoins.

Exemple

Els bitcoins s'han guanyat la confiança de molta gent perquè són diners relativament fàcils d'aconseguir. Per a un usuari final, la tasca que ha de realitzar per a obtenir un bitcoin és tan senzilla com executar un programa i esperar. Depenent de la capacitat del servidor trigarà més o menys i podrem obtenir-ne beneficis depenent del consum energètic. Al cap i a la fi, és un problema amb base econòmica, en què els beneficis finals dependran dels ingressos (els bitcoins bestiars) menys les despeses (energètiques i de temps invertit).

Per obtenir un bitcoin s'ha de resoldre un problema criptogràfic que fa ús d'un algorisme de *hashing*, concretament el SHA-256. Una persona que vulgui generar un bitcoin ha d'executar aquest algorisme sobre múltiples cadenes alfanumèriques repetidament fins que el resultat sobre una d'elles sigui vàlid, i llavors guanyarà una fracció de bitcoin.

Com que hem vist que minar bitcoins és una tasca altament paralel·litzable en la qual s'han d'executar un conjunt d'operacions matemàtiques senzilles sobre cadenes alfanumèriques aleatòries, els sistemes amb GPU són la millor opció, ja que permeten repetir una mateixa operació matemàtica en paral·lel de manera eficient.

Secure Hash algorithm

SHA versió 2 és un conjunt de funcions *hash* criptogràfiques dissenyades per l'Agència de Seguretat Nacional (NSA) que va ser publicat l'any 2001 per l'Institut Nacional d'Estàndards i Tecnologia (NIST) com un Estàndard Federal de Processament de la Informació (FIPS) als EUA. Una funció `hash` és un algoritme que transforma un conjunt arbitrari d'elements de dades, com pot ser una cadena alfanumèrica, en un únic valor de longitud fixa (el *hash*). El valor *hash* calculat pot ser utilitzat per a la verificació de la integritat de còpies d'una dada original sense la necessitat de proveir la dada original. Aquesta irreversibilitat significa que un valor *hash* pot ser lliurement distribuït o emmagatzemat, ja que només s'utilitza per a finalitats de comparació.

7. *Stacks* de programari per a sistemes de dades massives

Al mercat actual de les dades massives és possible trobar una gran quantitat de plataformes, programari, llibreries, projectes de codi obert, etc. que en combinar-se permeten generar tot l'*stack* necessari per a la captura, l'emmagatzematge i el processament de grans volums de dades.

Atès que l'ecosistema de dades massives encara es troba en evolució, és difícil descriure un *stack* de programari complet i universal que funcioni correctament en tots els projectes relacionats amb les dades massives. A la figura 15 es representen les quatre capes principals d'un sistema de dades massives, i a més s'hi inclouen alguns dels programaris o tecnologies més comunes que podem trobar per a resoldre els problemes d'aquesta capa.

Figura 15. Descripció de les capes principals d'un sistema de dades massives

Concretament, observem les capes següents (en anglès, *layers*):

- **Ingest Layer.** És la capa encarregada de capturar dades. Tenim eines com Apache Flume,* un sistema de gestió de fluxos de dades (*streams*) que permet transformar dades en cru abans de ser emmagatzemades al sistema.

* https://flume.apache.org

- **Store Layer.** A la capa d'emmagatzematge trobem tot tipus de sistemes per guardar i recuperar grans volums de dades de manera eficient. Aquí s'in-

clouen tot tipus de bases de dades, tant relacionals com NoSQL, sistemes de fitxers distribuïts, com HDFS, o altres tecnologies d'emmagatzematge al núvol, com Amazon S3.

- **Process/Analyze Layer.** Sens dubte la capa de processament és la més complexa i més heterogènia de totes. És on situem tecnologies com Spark o Hadoop, de les quals tant hem parlat en aquest mòdul.

- **Visualize Layer.** Finalment, en la capa de visualització, eines com Tableau* permeten una visualització de dades interactiva de forma relativament senzilla.

* https://www.tableau.com

Com ja hem comentat, aquest *stack* és molt variable. Depèn, en primer lloc, del projecte concret en el qual treballem. En aquest sentit, el tipus de dades que emprem, així com el tipus d'anàlisi requerida seran determinants per a configurar un *stack* determinat amb els elements imprescindibles o més adequats per poder implementar satisfactòriament el projecte en qüestió.

D'altra banda, és important destacar que el conjunt d'eines disponibles pot variar significativament en poc temps.

Per tot això, no és possible definir un *stack* únic i perdurable que pugui ser utilitzat en qualsevol projecte a curt o mig termini i serà necessari definir un *stack* concret en cada situació i en cada moment.

Resum

Tots els experts confirmen que les dades massives (*big data*) han aparegut a l'escena de les tecnologies de la informació per quedar-s'hi. Molts estudis apunten que la revolució que està generant aquesta nova cultura de les dades ha impactat i impactarà en tots i cadascun dels negocis actuals i del futur. Per adonar-se de com de certa és aquesta afirmació només cal veure la gran quantitat d'empreses emergents (*start-ups*), o nous negocis, que estan apareixent al voltant d'aquesta nova cultura de les dades massives o de *data science*.

En aquest mòdul hem introduït els principals components d'una arquitectura de dades massives. Hem començat descrivint l'estructura general d'un sistema de dades massives, després hem parlat sobre com emmagatzemar les dades en sistemes d'arxius distribuïts o en bases de dades NoSQL. Seguidament, hem passat a descriure dues tecnologies diferents per poder processar grans volums d'informació de forma eficient fent ús de la computació distribuïda i científica. A continuació, hem explicat com es gestionen tots els components d'un clúster fent servir un gestor de recursos, com, per exemple, YARN.

Finalment, hem esmentat els dos paradigmes o escenaris principals del processament distribuït, del processament en lots (*batch*), del processament en flux (*stream*) i del processament en GPU, hem explicat les seves diferències i les seves característiques principals.

Glossari

API *m* Vegeu **interfície de programació d'aplicacions**.

batch *m* Vegeu **procés per lots**.

bloc de dades *m* Unitat mínima en la qual un fitxer emmagatzema informació.

call detail record *m* Registre de dades generat en la comunicació entre dos telèfons fixos o mòbils que documenta els detalls de la comunicació (per exemple, trucada telefònica, missatges de text, etc.). El registre conté diversos atributs com l'hora, la durada, l'estat de finalització, el número de la font o número de destinació.
sigla **CDR**

CDR *m* Vegeu ***call detail record***.

chipset *m* Conjunt de circuits integrats dissenyats sobre la base de l'arquitectura d'un processador que permet que aquest funcioni en una placa base. Serveix de pont de comunicació del processador amb la resta de components de la placa, com són la memòria, les targetes d'expansió, els ports USB, el ratolí, el teclat, etc.

clúster *m* En general, quan ens referim a un conjunt de servidors parlarem de clúster.

codi obert Terme amb el qual es coneix el programari distribuït i desenvolupat lliurement. El codi obert té un punt de vista més orientat als beneficis pràctics de compartir el codi que a les qüestions ètiques i morals, les quals destaquen a l'anomenat programari lliure.
en open source

cua de tasques *f* Una cua no és més que un sistema per a executar les tasques en un cert ordre aplicant una sèrie de polítiques de una priorització.

dades estructurades *m pl* Dades que tenen ben definits la seva longitud i el seu format, com les dates, els nombres o les cadenes de caràcters.
en structured data

dades no estructurades *m pl* Dades que en el seu format original manquen d'un format específic.
en unstructured data

dades semiestructurades *m pl* Dades que no es limiten a un conjunt de camps definits, com en el cas de les dades estructurades, sinó que contenen marcadors per separar els seus diferents elements.
en semistructured data

deamon *f* Un *daemon* (nomenclatura utilitzada en sistemes UNIX i UNIX-like) o servei (nomenclatura utilitzada en Windows) és un tipus especial de procés informàtic no interactiu que s'executa en segon pla en comptes de ser controlat directament per l'usuari.

disc magnètic rotacional *m* Acrònim de Hard Disk Drive.
sigla **HDD**

DRAM *f* Vegeu **memòria dinàmica d'accés aleatori**.

fitxer de logs *m* Fitxer de logs, o bitàcola en català, és un fitxer seqüencial que emmagatzema tots els esdeveniments que ha processat un servidor. Normalment es guarden en un format estàndard per poder ser processats de forma senzilla.

HashMap *m* Un HashMap és una col·lecció d'objectes, com un vector o *arrays*, però sense ordre. Cada objecte s'identifica mitjançant algun identificador apropiat. El nom *hash* fa referència a una tècnica d'organització d'arxius anomenada *hashing* o dispersió en el qual s'emmagatzemen els registres en una adreça que és generada per una funció que s'aplica sobre la clau del registre.

HDD m Vegeu **disc magnètic rotacional**.

interfície de programació d'aplicacions *f* Conjunt de rutines, funcions i procediments (o mètodes, a la programació orientada a objectes) que ofereix una biblioteca de programació perquè pugui ser utilitzada per un altre programari com una capa d'abstracció.
sigla **API**

llenguatge SQL *m* Llenguatge declaratiu d'accés a bases de dades relacionals que permet especificar diversos tipus d'operacions.
en structured query language

MapReduce Tècnica de processament distribuït basada en el concepte de divideix i venceràs.

memòria dinàmica d'accés aleatori *f* També anomenada RAM dinàmica, és el tipus de memòria que es refereix a una tecnologia de memòria RAM basada en condensadors.
en dynamic random access memory (sigla DRAM)

múltiples servidors *m pl* Vegeu **clúster**.

Page Rank *m* Família d'algorismes utilitzats per a assignar de forma numèrica la rellevància dels documents (o pàgines web) indexats per un motor de cerca.

processament iteratiu *m* Tipus especial de processament que requereix accedir moltes vegades a les dades d'entrada.

procés per lots *m* Es coneix com a sistema per lots (*batch*) l'execució d'un programa sense el control o la supervisió directa de l'usuari. Aquest tipus de programes es caracteritzen perquè la seva execució no requereix cap tipus d'interacció amb l'usuari.
en batch

regles ACID *f pl* En el context de bases de dades, ACID (acrònim anglès de *atomicity, consistency, isolation, durability*) són una sèrie de propietats que ha de complir tot sistema de gestió de bases de dades per garantir que les transaccions siguin fiables. Per a una explicació més detallada es pot consultar l'entrada següent de la Wikipedia: http://es.wikipedia.org/wiki/ACID.

remote method invocation Tecnologia Java per a executar codi arbitrari de forma remota en un ordinador.
sigla **RMI**

RMI *m* Vegeu ***remote method invocation***.

shuffling Tècnica que permet agrupar un conjunt de dades que comparteixen certes característiques.

solid state disk Sistema d'emmagatzematge de dades permanent amb accés directe a les dades.
sigla **SSD**

SSD *m* Vegeu **solid state disk**.

tolerància a errors *f* Propietat que permet a un sistema computacional recuperar-se d'un error de maquinari.

unitat de processament gràfic *f* Coprocessador dedicat al processament de gràfics o operacions de coma flotant, fet per alleugerir la càrrega de treball del processador central en aplicacions com els videojocs, aplicacions 3D interactives o de càlcul científic.
en Graphics Processor Unit (sigla GPU)

Bibliografia

Barlas, Gerassimos (2014). *Multicore and GPU Programming. An Integrated Approach*. Boston: Elsevier.

Biery, Roger (2017). *Introduction to GPUs for Data Analytics. Advances and Applications for Accelerated Computing*. San Francisco: O'Reilly Media.

Kumar, Manish; Singh, Chanchal (2017). *Building Data Streaming Applications with Apatxe Kafka*. Birmingham: Packt Publishing.

Sarkar, Aurobindo (2017). *Learning Spark SQL*. Birmingham: Packt Publishing.

White, Tom (2015). *Hadoop: The Definitive Guide, 4th Edition Storage and Analysis at Internet Scale*. Boston: O'Reilly Media.

Zaharia, Matei; Chambers, Bill (2017). *Spark: The Definitive Guide, Big Data Processing Made Simple*. Boston: O'Reilly Media.