BIG DATA

INTRODUCTION TO APACHE SPARK

1. CONEPTES CLAU

Dades a escala: aquests últims anys s'està produint un fenomen anomenat Datafication. Es tracta de que cada acció que realitzem, tant màquines com humans, es capta d'alguna manera o una altra. Per aquesta raó, es parla de 175-181 zettabytes de dades pel 2025. Quan parlem de processament de dades a escala, no només es parla de volum, sinó també de varietat.

No ens podem preocupar només per on guardem les dades, sinó que també ens hem de preocupar per la seva estructura. Les dades que obtenim de diferents fonts poden tenir estructures diferents estructures, o inclús no tenir-ne. Com que tenim tantes dades, es necessiten sistemes distribuïts per processar-les.

Facilitat d'ús: té a veure amb els diferents rols que pot haver en un equip. Al treballar amb dades, potser un framework és fàcil per un rol en concret, però no tant fàcil per altres, ja que cada rol té diferents backgrounds i cada un d'ells treballa en diferents tipus de problemes i amb llenguatges de programació diferents. El treball és multidisciplinari, és a dir, treballa diferents tipus de disciplines, no tots estan encarant el problema de la mateixa manera.

Quan es treballa amb dades, és comú seguir un patró determinat, el qual primer llegeixes les dades de les fonts, les neteges, les transformes, s'agreguen (relacions), i es guarden les dades d'una manera més agregada. El problema és que seguir aquest patró des de zero no és efectiu, i tenim diferents operadors de processament de dades per tal de no haver de començar des de zero en cada un dels processos. Necessitem treballar amb plataformes que tinguin abtraccions d'alt nivell.

Velocitat: Apache Spark és més ràpid de Hadoop MapReduce, però quan parlem de més ràpid, només importa el temps? La rapidesa és molt important perquè és bàsicament el time-to-market, és important que els pipelines de processament de dades s'executin de la manera més ràpida possible, per no perdre oportunitats de negoci.

Flexibilitat: quan processem les dades, no és un tipus de processament específic. Tenim diferents tipus de workload (per exemple pot ser batch, o real time streaming). Que una plataforma o un framework sigui flexible significa que puc utilitzar-lo en diferents workloads. Utilitzar una tècnica diferent per cada workload no és eficient ni barat.

La flexibilitat està associada als costos; l'empresa té el TCO (Total Cost of Ownership), que pot estar relacionat en tenir diversos temes, o pagar diferents tipus de llicències o de recursos de còmput.

2. APACHE SPARK

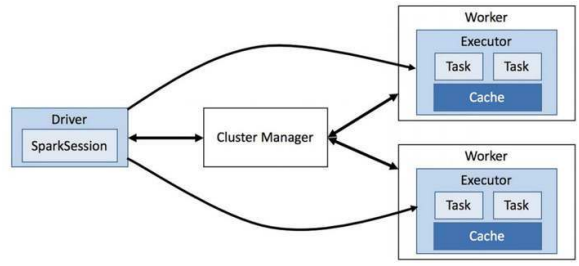
És el resultat d'una tesis de Matei Zaharia's. Al 2009 va presentar la seva tesis i volia trobar algo millor al Hadopp MapReduce, i va sortir Apache Spark. Al 2010 va passar Spark a Apache i el 2013 ja era un dels projectes top. Van fundar l'empresa Databricks al 2013, que és un dels principals contruibuidors, i al 2015 va entrar IMB a Apache Spark. Van desenvolupar Spark amb Scala perquè volien un llenguatge de programació que fos concís i que tingui un tipus de dades estàtic.

Spark és un sistema distribuït (i generalista) per processar grans quantitats de dades de manera eficient i ràpida. El clúster més gran de Spark té més de 8000 màquines. Quan parlem de clústers, hem de gestionar els recursos de les màquines, i per gestionar-los tenim un sistema de gestió de recursos, on tenim el master, que és l'entitat que dirigeix la execució, i els workers, que són els que realitzant l'execució.

Exemples de sistemes de gestió de recursos: Apache YARN, Apache Mesos, Kubernetes.

Per treballar amb Apache Spark hi ha uns requeriments que ens obliguen a processar les dades. Aquests requeriments es transformen en una lògica expressada amb l'API de Spark. Amb l'API puc crear una aplicació, que per executar-la es necessiten molts conceptes a nivell de runtime.

Cada aplicació de Spark és gestionada per un Driver, que és l'encarregat de dirigir l'execució de l'aplicació. L'única cosa que fa és gestionar la distribució de la computació de cada una de les accions. Aquestes accions les executen els Executors en forma de Task, que és qui realment fa l'execució del codi font (existeix un Driver per cada app, però poden existir N Executors, i s'assigna un CPU core per Task). Si comparem l'arquitectura dels clústers (master i worker) amb l'arquitectura de Spark (Driver i Executor), podríem dir que el procés del Driver està al master, i els Executors estan als workers.



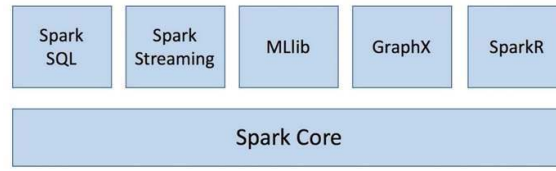
Hi ha dos nivells de paral·lelisme. Un primer amb els Executors, i després amb les Tasks.

RDD (Resilient Distributed Datasets)

És una estructura de dades que es pot operar de manera paral·lela i tolera errors, perquè els resultats que es van processant es van guardant els passos entremitjos. Treballem amb registres, i al moment de crear un RDD aquest és només de lectura, l'estructura no es pot modificar. Si la vull modificar, ho faig a través de transformacions (ex. map) i accions (ex. count). Quines aplicacions no poden ser representades amb RDD?

3. ARQUITECTURA

Spark es basa en Spark CORE, i sobre de Spark CORE existeixen una sèrie de llibreries les quals estan associades a un tipus de workload específic (flexibilitat).



Spark CORE és qui gestiona l'arquitectura de Driver-Executor i la distribució de treballs. On tenim definida la part de la programació i com es fa la monitorització de tots els processos, i com es fa per carregar i guardar dades en diferents sistemes.

La llibreria Spark SQL és la que ofereix una API d'alt nivell anomenat Dataframe, i ens permet treballar de manera més suelta que RDD. El catalyst optimizer s'encarrega de passar el codi SQL de manera eficient. Et permet llegir fitxers JSON, CSV, etc.

La llibreria MLlib es basa en els Dataframes i ens ofereix una sèrie d'algoritmes perquè puguem fer un recomenador.