Spark – Part 1

Software Concurrent

Lluís Garrido – lluis.garrido@ub.edu

Grau d'Enginyeria Informàtica

Spark

Què és ?

- Spark és un motor de computació unificat per a clústers d'ordinadors.
- A l'actualitat inclou una sèrie de llibreries i eines per a fer un munt de tasques paral·leles.

Per a què és útil?

- Spark és útil per a realitzar consultes sobre les dades, per aplicar la mateixa operació sobre tots els elements d'aquesta.

Per a què no és util?

- No és gaire útil per a aplicacions que impliquin "fine-grained operations", com per exemple un rastrejador de web incremental.

Spark: el contexte del Big Data

Per què ha sigut necessari desenvolupar aquests tipus de sistemes?

- Fins el 2005 els ordinadors es poden fer més ràpid augmentant la freqüència de rellotge. A causa de limitacions tecnològiques, van aparèixer els sistemes multiprocessador.
- Les tecnologies per emmagatzemar dades (els discos) i capturar-les (càmeres, sensors, etc.) es fan cada cop més econòmiques. És molt fàcil i econòmic capturar dades!
- Però per processar aquestes dades calen clústers de dades, computacions paral·leles. Fan falta nous models de programació. I és en aquest món on ha aparegut Spark.

Spark: una mica d'història...

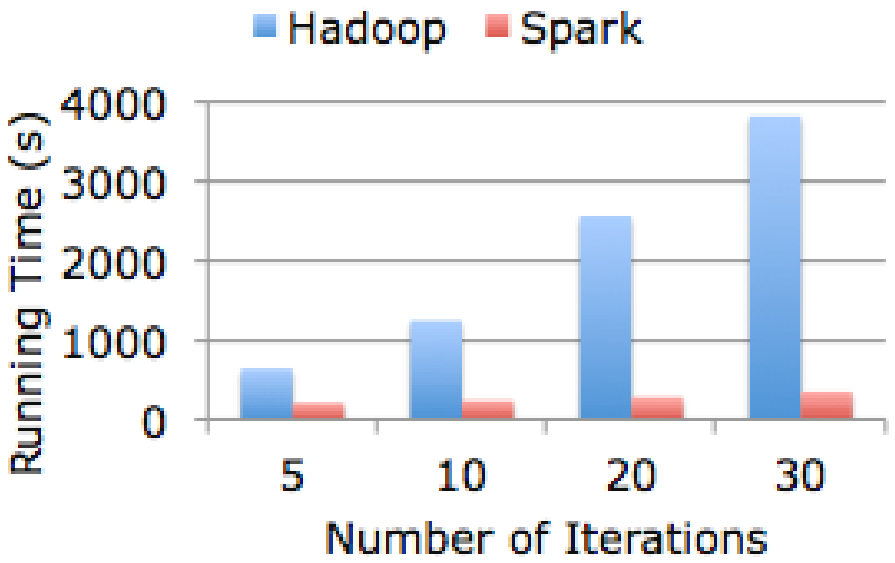
Una mica d'història del seu desenvolupament

- Apache Spark va començat a desenvolupar-se al 2009 al AMPLab de Berkerley com a projecte de recerca.
- El AMPLab havia desenvolupat abans el Hadoop Mapreduce, el motor de programació paral·lela dominant en aquell moment. El AMPLab va estudiar les avantatges i desavantatges d'aquest i desenvolupar un model més general.
- Dues coses els van quedar clar: la computació en clúster tenia un gran potencial ja que moltes organitzacions començaven a utilitzar Hadoop. Per altra banda, MapReduce feia difícil i ineficient desenvolupar grans aplicacions ja que feia falta accedir molt a disc (per carregar i desar dades).

Spark: una mica d'història...

Per tal de resoldre aquests problemes

- AMPLab va començar a desenvolupar una API basada en llenguatge funcional (Scala) que permetés expressar, de forma breu i concisa, aplicacions que tinguessin múltiples passos.
- El motor va ser dissenyat per tal que els múltiples passos d'una aplicació es fessin a memòria (en comptes d'emmagatzemar-les a disc contínuament). Això va fer Spark molt més ràpid!



Spark: una mica d'història...

- Les primeres versions de Spark només suportaven aplicacions tipus "batch", un sistema interactiu capaç d'executar consultes en clústers de centenars d'ordinadors. AMPLab va desenvolupar també Shark, capaç de suportar consultes SQL. Va ser publicat al 2011...
- El projecte va continuar desenvolupant-se i va quedar clar que feien falta llibreries per aprofitar tot el potencial de Spark: MLib (machine learning), Spark Streaming i GraphX (per processar grafs).
- Al 2013, el projecte havia crescut tant que hi havia més de 30 organitzacions externes, fora de Berkeley, que hi contribuïen.

Spark: una mica d'història...

- AMPLab va llençar Databricks (`https://databricks.com/`), una companyia per donar més difusió al projecte.
- Spark 1.0 va ser llençat el 2014, i Spark 2.0 el 2016.... i continua desenvolupant-se.
- Tot i que Spark ha sigut desenvolupat en Scala (i s'executa sobre la Java Virtual Machine), existeixen interfícies que permeten executar-lo des de Python, Java, Scala, R o SQL.

Spark: com executar-lo?

Com podem executar Spark?

- Fent servir Databricks (la versió gratuïta és força limitada ja que no s'executa en un clúster).
- Des del nostre ordinador, baixant una versió local de l'Spark: `http://spark.apache.org/downloads.html`. Seleccionar "Pre-built for Hadoop 2.7 and later". Descomprimiu-lo i s'instal·larà...
- Cal tenir instal·lada la Java Virtual Machine i, si es vol fer servir la interfície Python, també cal tenir-lo instal·lat.
- Per executar Spark de forma interactiva fent servir Python, executar "`pyspark`" que es troba dins del paquet que us heu baixat!

Spark: com executar-lo?

En aquest curs...

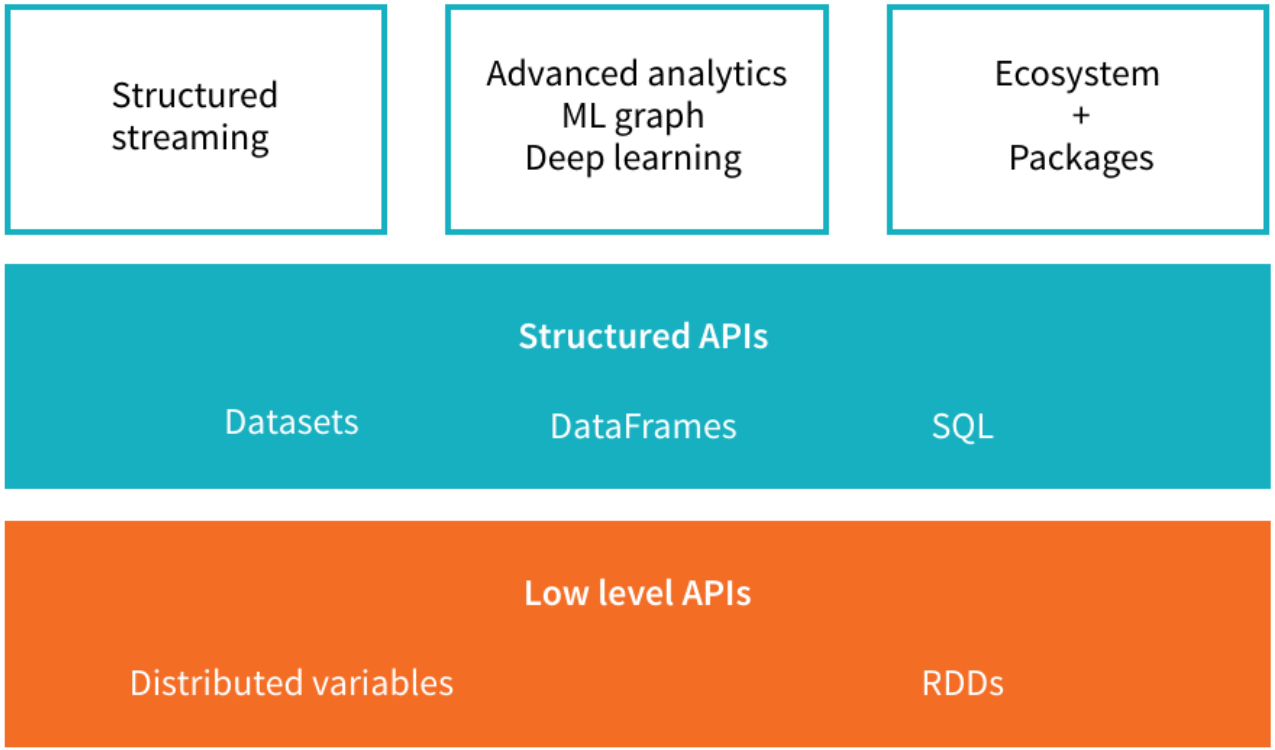
- Farem servir la versió local de Spark: Spark s'executa en un sol ordinador i s'aprofitaran els múltiples processadors que l'ordinador té. La programació no canvia si es fa servir en un clúster.

Un sistema en un clúster ha d'oferir

- Robustesa: el sistema ha de ser robust en front a fallades parcials. Si una unitat (ordinador) falla, la tasca ha de ser assumida per un altre ordinador. Mai s'han de perdre dades a causa d'una fallada parcial.
- Escalabilitat: si s'afegeixen noves unitats al sistema, aquest les ha d'incorporar al clúster sense necessitat de reiniciar-lo i ha de repercutir en un augment de la capacitat de càrrega del sistema.

Spark: les components i llibreries

Aquest és el kit de components i eines que ofereix Spark.



Spark: les components i llibreries

Breument,

- La "màgia" al darrera de l'Spark és el RDD (Resilient Distributed Datasets). És l'abstracció d'un sistema segur a fallades que permet realitzar operacions a un clúster d'ordinadors.
- Les operacions que permet fer són de "grà gruixut" (sobre totes les dades). Les operacions es distribueixen de forma intel·ligent i automàtica entre tots els ordinadors del clúster.
- El RDD és unes 20 vegades més ràpid que el Hadoop perquè, entre altres coses, el sistema utilitza un esquema (segur a fallades) en què els resultats de les operacions intermèdies s'emmagatzemen a memòria RAM i no s'han d'emmagatzemar a disc.
- El RDD va ser implementat a Berkeley en Scala, i originalment eren unes 14.000 línies de codi (al 2012).

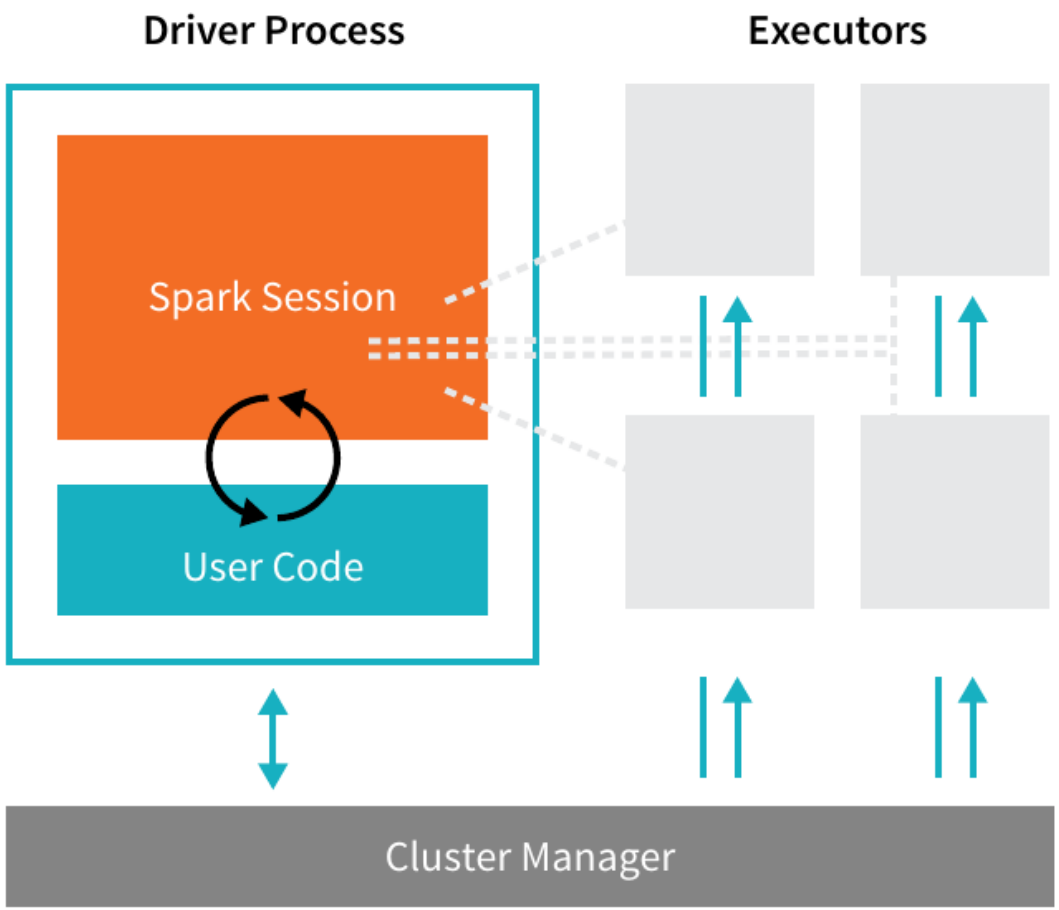
Spark: les components i llibreries

Breument,

- El RDD és l'API de baix nivell. No es recomana fer-la servir llevat que les APIs d'alt nivell no permetin solucionar el problema que teniu.
- L'API de nivell alt són els DataFrames, SQL (i Datasets) permeten realitzar operacions sobre les dades. El mòdul SQL permet fer consultes SQL sobre les dades, i el DataFrame es pot interpretar intuïtivament com una fulla de càlcul (distribuïda entre el clúster d'ordinadors).
- Per sobre de aquesta API hi han mòduls que permeten rebre i processar dades en "streaming", aplicacions d'aprenentatge automàtic, anàlisi de grafs, etc.

Spark: l'arquitectura

Aquesta és l'arquitectura de Spark



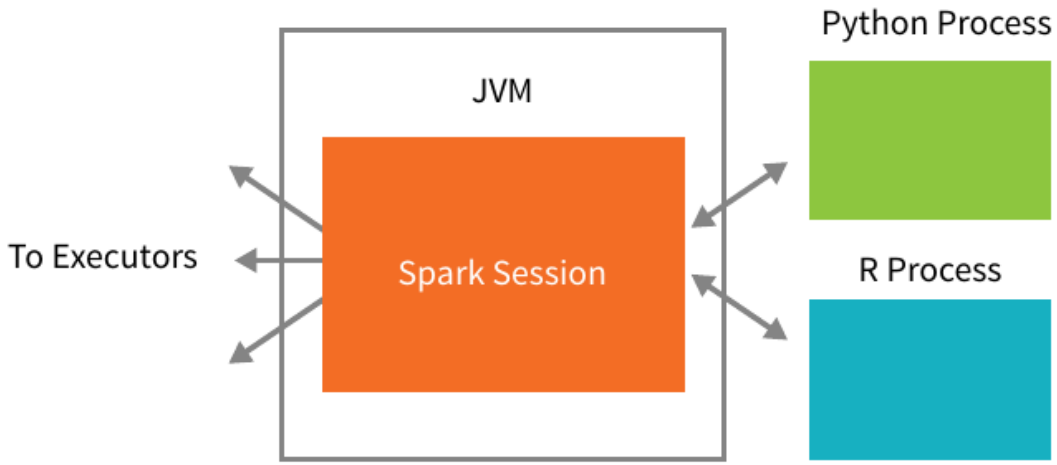
Spark: l'arquitectura

L'arquitectura de Spark

- Spark *driver*: és el procés que controla l'aplicació Spark. S'executa en un node del clúster i és responsable de distribuir i planificar el treball en els *executors*.
- Spark *executors*: són els processos que executen la feina que el driver els assigna. Es comuniquen amb el driver per informar-lo de l'estat de la computació que els ha estat assignat.
- El *cluster manager* és responsable de mantenir el clúster de màquines que executaran Spark així com els fitxers que s'hi emmagatzemen. Spark pot executar sobre diversos cluster managers: una versió local, Hadoop YARN o Mesos.

Spark: la SparkSession

El procés de Spark driver està associat, en executar Spark, a un objecte anomenat SparkSession. Podem executar codi Spark fent servir Scala, Java, Python, SQL o R.

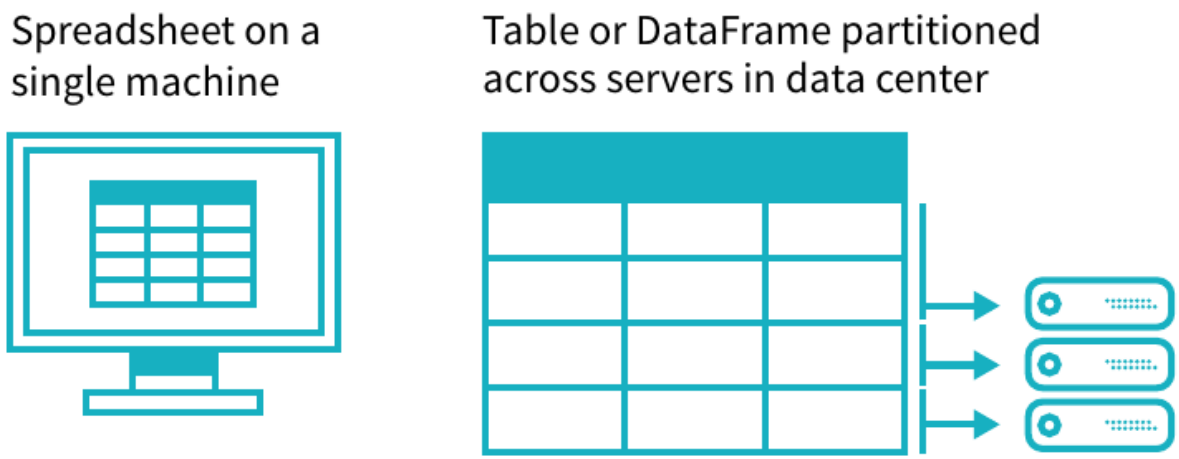


A Python, la SparkSession està associada a la variable objecte spark. Executeu "`pyspark`" i executeu "`spark`" per veure el seu valor.

Spark: els DataFrames

Principalment treballarem amb DataFrames

- Intuïtivament és una fulla de càlcul, amb les seves columnes i files. Estem acostumats que una fulla de càlcul estigui emmagatzemada només en un ordinador.
- A Spark, en canvi, la fulla de càlcul està distribuïda en particions. Cada partició és un conjunt de registres de la fulla de càlcul que està emmagatzemada en un ordinador diferent del clúster.



Spark: els DataFrames

Les particions són importants!

- Spark divideix (automàticament) les dades en particions, de forma que cada executor d'un clúster pot processar la partició que emmagatzemada al seu node. Cada partició està emmagatzemada en un node diferent.
- Inclús en mode local, dividir les dades en particions és beneficiós: cada processador d'un ordinador pot processar una partició diferent.
- Com a usuaris, no processarem les particions de forma individual: nosaltres especifiquem l'operació a realitzar i Spark s'encarrega de distribuir la feina entre els executors dels nodes.

Spark: les particions

Uns detalls sobre les particions

- El clúster manager és qui s'encarrega d'emmagatzemar les particions dels DataFrames als nodes del clúster. Hi ha, per exemple, el Hadoop Distributed Filesystem (HDFS). Mireu el fitxer `HDFS-comic.pdf`, que explica com funciona aquest sistema de fitxers.
- Spark, per sobre del clúster manager, és qui ho gestiona tot: el nombre de particions, on és cada partició, etc.

Spark: les transformacions

Com a usuaris volem fer consultes sobre el DataFrame. Per exemple,

- Ordenar els elements d'un DataFrame (composat de milions de registres) pel valor d'alguna columna.
- Filtrar (eliminar) del DataFrame tots els elements que no compleixin una determinada condició.
- Agrupar les registres d'un DataFrame que tenen a una determinada columna el mateix valor.

Tot això (i molt més) són les manipulacions que podem fer a un DataFrame. Tècnicament, s'anomenen transformacions.

Spark: les transformacions

Les estructures de dades de Spark (com els DataFrames) són inmutables, no es poden modificar els seus valors. Mitjançant les transformacions indiquem a Spark com volem manipular les dades que hi ha al DataFrame. Vegem un exemple[1]:

```
1 # Creem un DataFrame per llegir les dades
2
3 flightData = spark \
4     .read \
5     .option("inferSchema", "true") \
6     .option("header", "true") \
7     .csv("flight-data/2015-summary.csv")
8
9 # Fem una transformacio: ordenem les dades
10 # per la columna "count" del DataFrame
11
12 flightDataOrdered = flightData.sort("count")
13
```

[1] Els exemples estan en Python3. Cal indicar a la variable d'entorn `PYSPARK_PYTHON` on és l'executable de Python3.

Spark: les transformacions

Atenció! A l'exemple anterior demanem llegir el fitxer i ordenar les dades per una determinada columna.

- Observar que el resultat de l'ordenació s'assigna a un DataFrame diferent de l'original (ja que l'original és inmutable).
- Tant la lectura de dades com l'ordenació no és realitza en en moment d'executar l'operació. Tècnicament és el que s'anomena "*Lazy evaluation*". De moment només estem indicant les operacions que volem realitzar sobre les dades originals. Estem organitzant un pla de transformacions.
- En realitzar una acció Spark compilarà i executarà les transformacions que li hem demanat, de forma eficient, per realitzar l'acció demanada.

Spark: les accions

Hi ha tres tipus diferents tipus d'accions:

- Accions per visualizar informació a consola
- Accions per escriure dades a disc
- Accions per "transformar" les dades a objectes del llenguatge (Java, Python, ...) que s'estigui fent servir.

Per a l'exemple anterior

```
1 # Creem un DataFrame per llegir les dades
2
3 flightData = spark \
4     .read \
5     .option("inferSchema", "true") \
6     .option("header", "true") \
7     .csv("flight-data/2015-summary.csv")
8
9 # Fem una transformacio: ordenem les dades
10 # per la columna "count" del DataFrame
11
12 flightDataOrdered = flightData.sort("count")
13
14 # Fem una accio sobre el DataFrame ordenat.
15 # Mostrem els 5 primers elements
16
17 flightDataOrdered.show(5)
```

Spark: les accions

Recordem:

- En especificar una acció, s'utilitza el graf de transformacions especificades per tal de realitzar l'acció demanada de la forma més eficient possible.
- A l'exemple anterior, l'acció demana només els 5 primers elements. Es necessari llegir tot el fitxer però no fa falta ordenar tots els elements ja que només volem els 5 primers.

Spark: les accions

Vegem un altre exemple

```
1 # Creem un DataFrame per llegir les dades
2
3 flightData = spark \
4     .read \
5     .option("inferSchema", "true") \
6     .option("header", "true") \
7     .csv("flight-data/2015-summary.csv")
8
9 # Fem una transformacio: nomes volem els elements
10 # en que count es divisible per 10.
11
12 flightDataFiltered = flightData.where("count % 10 = 0")
13
14 # Fem una accio sobre el DataFrame filtrat.
15 # Mostrem els 5 primers elements
16
17 flightDataFiltered.show(5)
18
```

Spark: les accions

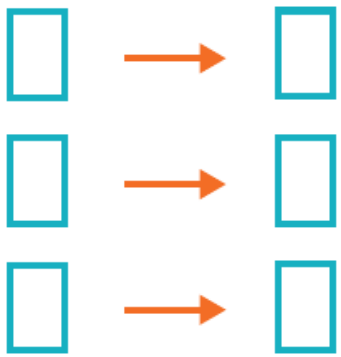
A l'exemple anterior, en executar l'acció show es compilen les transformacions (lectura i filtratge) i Spark decidirà que...

- No fa fa falta llegir tot el fitxer. En llegir-lo, es poden descartar directament tots els elements en què count no sigui divisible per 10.
- En tenir 5 elements, ja no cal continuar l'execució de l'acció!

Spark: tipus de transformacions

Hi ha dos tipus diferents de transformacions: *narrow dependency* i *wide dependency*.

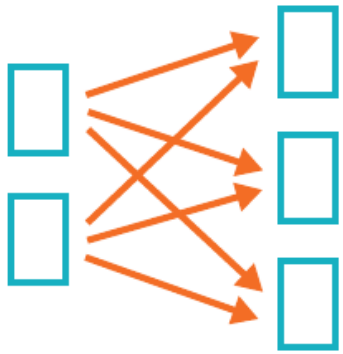
En una transformació de *narrow dependency*, una partició donarà com a resultat una altra partició. No cal moure dades entre els nodes del clúster.



Exemples: sumar dues columnes, afegir una nova columna, eliminar una columna, filtrar per un criteri d'una columna, etc.

Spark: tipus de transformacions

En una transformació de *wide dependency*, una partició pot contribuir a diverses particions a la sortida. S'acostuma a utilitzar el terme *shuffle*, ja que Spark haurà d'intercanviar informació de les particions als nodes del clúster.



Exemples: ordenar els elements del DataFrame per una determinada columna, agrupar els elements del DataFrame per una determinada columna, etc.

Spark: una gran idea!

Una de les grans novetats a l'Spark és el seu funcionament

1. L'usuari escriu el codi mitjançant transformacions i accions (DataFrames/SQL)
2. Si el codi és vàlid, Spark ho converteix a un pla lògic (*logical plan*). El pla lògic només és una abstracció de les operacions a realitzar.
3. El codi passa per l'anomenat *Catalyst Optimizer*, que ho transforma en un pla físic (*physical planning*), també anomenat Spark plan, un conjunt d'operacions optimitzat per a a les RDD que s'executen al clúster.

Spark: una gran idea!

Spark realitza, sempre que pot, totes les operacions que pot a memòria RAM... i això el fa ràpid!

- A la competició Daytona Graysort l'objectiu és ordenar, tan ràpid com sigui possible, 100TB de dades (1 trilió registres).
- El més ràpid (fins al 2014) era de Hadoop MapReduce: ho va aconseguir en 72 minuts amb 2100 màquines.
- El 2014 ho va guanyar Spark, amb 23 minuts i 206 màquines AWS EC2.

Spark: una gran idea!

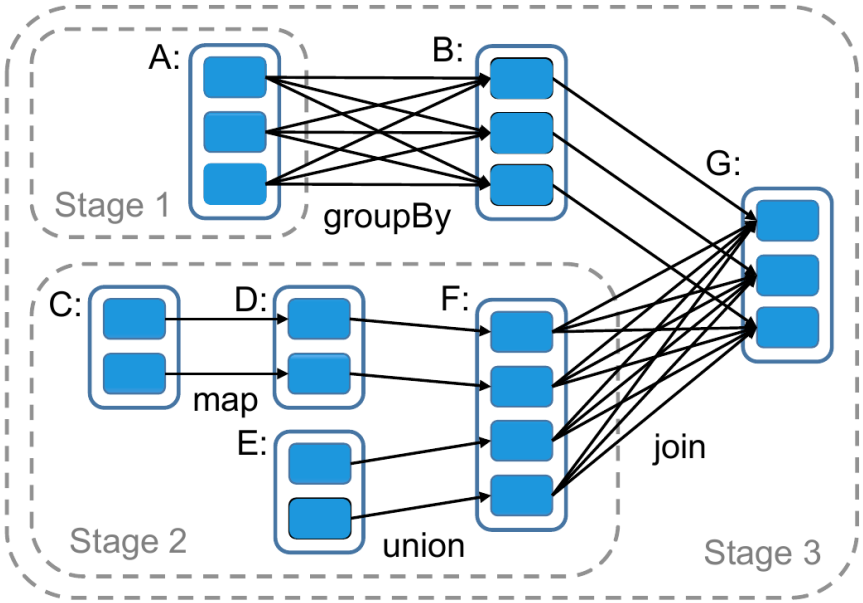
Spark introdueix també un mètode nou per assegurar la seguretat

- El sistema HDFS de Hadoop proporciona seguretat a nivell de fitxers en cas de fallada.
- Al sistema de processament de dades Hadoop les instruccions s'executen en el moment d'especificar-les. Els resultats dels passos intermedis s'emmagatzemen a disc per assegurar no perdre dades.
- Què és fa a Spark si, per exemple, una màquina del clúster falla?

Spark: una gran idea!

Spark introdueix també un mètode nou per assegurar la seguretat

- Spark emmagatzema les operacions a realitzar amb els RDD. Els resultats intermedis s'emmagatzemen a memòria RAM.
- Si un node falla, Spark pot "reconstruir" les particions necessàries a partir de les dades originals ja que sap on estan emmagatzemades al clúster de fitxers[2].



[2]Per a més informació, mireu l'article "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing"