Introduction à Apache Spark

1. Qu'est-ce que Apache Spark ?

Spark ou **Apache Spark** est un framework open source de calcul distribué. Créé en 2009 en Californie, Spark est aujourd'hui un projet de la fondation Apache. Il s'agit d'un des concurrents principaux de Hadoop.

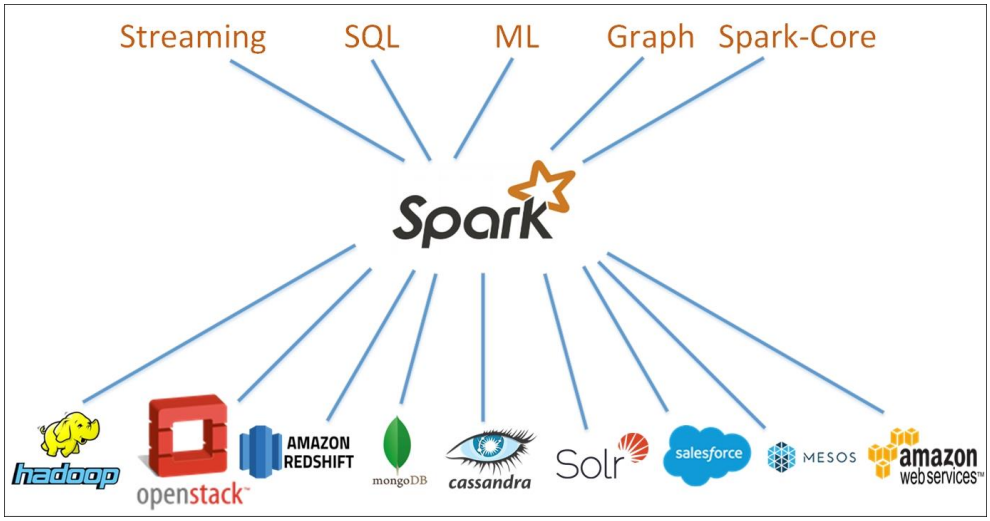
À l'origine son développement est une solution pour accélérer le traitement des systèmes **Hadoop**. Les développeurs mettent notamment en avant la rapidité du produit en termes d'exécution des tâches par rapport à **MapReduce**. Ainsi Spark finit par dépasser Hadoop en vitesse d'exécution en étant 10 à 100 fois plus rapide. Cela est dû au fait que Spark ne va pas chercher à écrire ses données sur un disque mais dans la RAM. Il s'agit d'un choix économique basé sur le fait que la RAM coûte de moins en moins cher.

Apache Spark propose une suite d'outils tel que :

- **Spark** pour les traitements "en batch"
- **Spark Streaming** pour le traitement en continu de flux de données
- **MLlib** pour le "machine learning"
- **GraphX** pour les calculs de graphes (encore en version alpha)
- **Spark SQL**, une implémentation SQL-like d'interrogation de données.

Par ailleurs, il s'intègre parfaitement avec l'écosystème Hadoop notamment HDFS.

Enfin un des grands intérêts d'Apache Spark est qu'il donne la possibilité d'interagir avec lui aussi bien en **Java** qu'en **Scala**, en **Python** ou en **R**.



2. Fonctionnement

https://www.youtube.com/watch?v=Y6bkFcIoB-U

Spark réalise une lecture des données au niveau du cluster (grappe de serveurs sur un réseau), effectue toutes les opérations d'analyse nécessaires, puis écrit les résultats à ce même niveau. Malgré le fait que les tâches s'écrivent avec les langages Scala, Java et Python, il utilise au mieux ses capacités avec son langage natif, Scala.

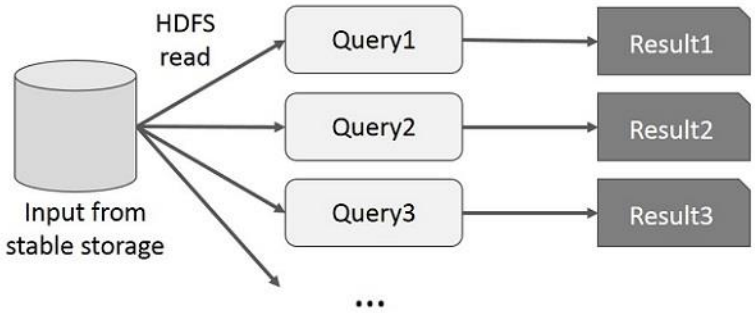
De ce fait, là où le MapReduce de Hadoop travaille par étape, Spark peut travailler sur la totalité des données en même temps. Il est donc jusqu'à dix fois plus rapide pour le traitement en lots et jusqu'à cent fois plus rapide pour effectuer l'analyse en mémoire.

Spark exécute la totalité des opérations d'analyse de données en mémoire et en temps réel, en utilisant la RAM. Il s'appuie sur des disques seulement lorsque sa mémoire n'est plus suffisante. À l'inverse, de Hadoop qui écrit les données sur le disque après chacune des opérations.

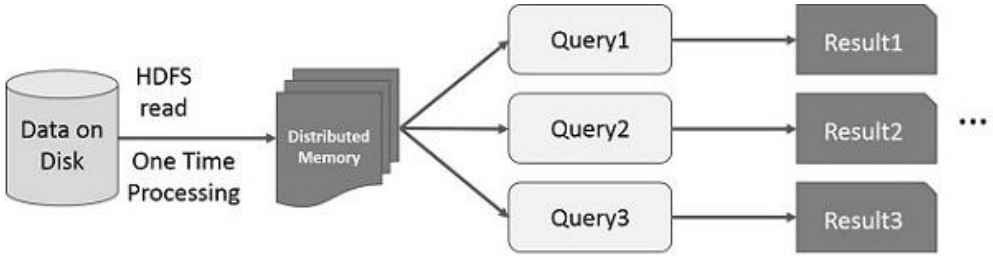
Cependant, Spark ne dispose pas de système de gestion de fichier qui lui est propre. Il est nécessaire de lui en fournir un, par exemple Hadoop Distributed File System ou Cassandra. Il est conseillé de l'utiliser avec Hadoop.

En cas de panne ou de défaillance du système : les objets de données sont stockés dans ce que l'on appelle des **ensembles de données distribués résilients** (**RDD : *resilient distributed datasets***) répartis sur le cluster de données permettant la récupération complète de données.

Un RDD est une collection de données calculée à partir d'une source et conservée en mémoire vive (tant que la capacité le permet). L'un des avantages apportés par RDD se trouve dans sa capacité à conserver suffisamment d'informations sur la manière dont une partition RDD a été produite. En cas de perte d'une partition il est donc en mesure de la recalculer.



Opération interactive via MapReduce



Opération interactive via Spark RDD

On remarque que c'est l'utilisation de la RAM qui permet à Spark de limiter les accès au disque et de pouvoir garder certains RDD en mémoire de manière persistente afin d'accroître son temps d'exécution.

3. Installation

L'installation d'**Apache Spark** va se faire beaucoup plus facilement que celle d'**Hadoop**.
Pour commencer, mettez en place (ou reprenez) une machine virtuelle Linux : Ubuntu, Debian, Centos, etc.
Désinstallez **Hadoop** s'il est déjà présent sur votre machine virtuel.

Vérifiez que vous avez bien la **version 8** de **Java** d'installée sur votre machine :

```
> sudo apt install openjdk-8-jre-headless
```

Il sera peut-être nécessaire de désinstaller d'autres versions de Java.

```
> sudo apt-get purge openjdk-\*
```

Ainsi que la **version 3** de **Python**.

Installez la dernière version de **Scala** (si *apt install* ne fonctionne pas : https://www.scala-lang.org/download/).

Installez ensuite la **version 2.4.5** de **Spark** avec pour package le type « **pre-built for Apache Hadoop 2.7** » : https://spark.apache.org/downloads.html

Ajoutez ensuite les lignes suivantes à votre fichier **.bashrc** (ou équivalent) :

export PATH=$PATH:/path/to/spark/bin
export PYSPARK_PYTHON=python3

Puis exécutez la commande suivante :

> source .bashrc

Allez dans le répertoire bin de Spark et lancez la commande ***spark-shell***. Si tout se passe bien, vous devriez avoir l'écran suivante :

4. Test et exemples

Lancez le spark-shell python **pyspark** et exécutez la commande suivante :

> sc.parallelize(range(1000)).count()

Cela devrait vous retourner 1000.

Maintenant lisez la documentation suivante afin de créer votre première application utilisant l'API de Spark :
https://spark.apache.org/docs/latest/quick-start.html

Deux petites questions :
- que fait l'option « **--master »** ?
- qu'implique le « **local[4] »** ?

5. L'API Spark

Lien utile :
- https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark

L'API Spark va permettre d'utiliser deux grands types de fonctions :
- les transformations RDD. Ce sont des fonctions permettant d'avancer dans le traitement des données, d'indiquer la manière dont on doit les traiter.
- les actions . Ce sont des fonctions permettant d'obtenir un résultat à partir d'un ensemble de données.

a) Les transformations

Voici un exemple de quelques fonctions de transformations :

- **map(func)** : retourne un nouvel ensemble de données en passant chaque élément de l'ensemble initiale à travers la fonction ***func***.
- **filter(func)** :retourne un nouvel ensemble de données en sélectionnant ceux pour lesquels la fonction ***func*** renvoie true.
- **union(otherDataset)** : retourne un nouvel ensemble de données fusionnant l'ensemble de données source et celui passé en paramètre.
- **intersection(otherDataset)** : retourne un nouvel ensemble de données contenant les données similaires entre l'ensemble source et celui passé en paramètre.
- join(otherDataset) : retourne un ensemble de données contenant tous les éléments ayant des clés similaires entre l'ensemble source et celui passé en paramètre.

b) Les actions

Voici un exemple de quelques fonctions d'actions :

- **reduce(func) :** Regroupe les éléments d'un ensemble de données en utilisant la fonction ***func*** (qui doit avoir une valeur de retour).
- **collect()** : retourne tous les éléments d'un ensemble de données en tant que tableau.

- **count()** : retourne le nombre d'éléments dans un ensemble de données.
- **first()** : retourne le premier élément d'un ensemble de données (similaire à take(1)).
- **take(n)** : retourne un tableau comportant les ***n*** premiers éléments d'un jeu de données.
- **takeOrdered(n, key=None)** : retourne les ***n*** premiers éléments d'un RDD en utilisant l'ordre ascendant ou une clé de comparaison optionnelle.
- **saveAsTextFile(path)** : écrit les éléments d'un ensemble de données dans le fichier passé en paramètre.
- **countByKey()** : compte le nombre d'éléments par rapport aux clés et renvoie un dictionnaire.
- **foreach(func)** : applique la fonction ***func*** sur tous éléments d'un jeu de données.

5. Exercices

a) Créez un programme qui compte le nombre de mots dans un fichier « input.txt » en utilisant l'API Spark.

b) Toujours en utilisant Spark, créez un programme qui compte le nombre de mots similaires dans un fichier. Vous passerez le mot recherché à votre programme et il affichera le nombre de fois qu'il a trouvé ce mot dans le fichier.

c) Avec Spark, créez un programme qui lit un fichier, le divise en phrase et affiche le nombre de mots présents dans chaque phrase. Le résultat sera retourné dans un fichier.

6. Mini-projet

Créez un programme qui indique le nombre de commerces parisien faisant des livraisons à domicile malgré le coronavirus et stocker les informations suivantes :
- le nom du commerce
- le type de commerce
- l'adresse
- le numéro de téléphone
- le site internet

Vous stockerez les informations de ces commerces dans un fichier « commerces-coronavirus.txt ».

Afin de traiter ces informations, vous effectuerez un requêtage sur l'API opendata de Paris et stockerez le résultat dans un fichier input-paris.txt. Vous utiliserez bien entendu l'API Spark pour effectuer le traitement.

Lien vers l'API :
https://opendata.paris.fr/explore/dataset/coronavirus-commercants-parisiens-livraison-a-domicile/information/?disjunctive.code_postal&disjunctive.type_de_commerce