Speed Layer

Le traitement de données massives (Big Data) implique souvent l'utilisation d'infrastructures distribuées comme l'écosystème Hadoop. Toutefois, lorsque l'on souhaite traiter des données en temps quasi-réel (speed layer), Hadoop *MapReduce* montre ses limites. C'est ici qu'intervient **Apache Spark**, plus adapté aux applications interactives et itératives grâce à un stockage en mémoire.

Pourquoi Hadoop n'est-il pas suffisant ?

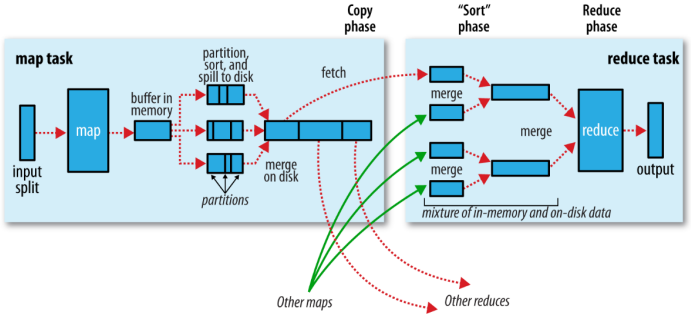
Avantages de Hadoop MapReduce

- Adapté aux traitements par lots (*batch*) de très larges volumes de données.
- Tolérance aux pannes intégrée.
- Utilise des machines standards (commodity hardware).

Limites de Hadoop MapReduce

- Stockage et lecture sur disque à chaque étape (Map, Shuffle, Reduce), ce qui ralentit considérablement les traitements itératifs.
- Difficulté à gérer les cas d'usage interactifs ou en flux quasi-temps réel.
- Shuffle/Sort coûteux en termes de communication réseau.

✘ For each job, each task (map, reduce, shuffle and sort) requires multiple read/write access to the workers hard drives



Apache Spark

Caractéristiques de Spark

- **Performance**: stockage et exécution en mémoire (jusqu'à 100x plus rapide que Hadoop dans certains cas).
- **API souples**: compatibilité avec Python (PySpark), Scala, Java, et R.
- **Stack complet**: Spark SQL, Streaming, MLlib (Machine Learning), GraphX (graphes).
- **Évolutivité** facile (scale-up et scale-out sur de grands clusters).

Spark Stack

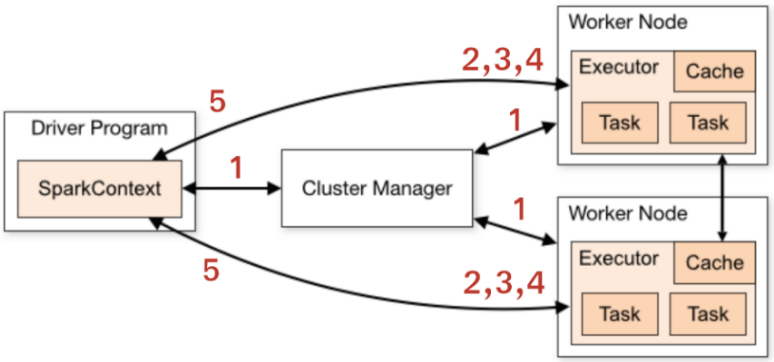
- **Spark Core**: gestion des ressources, scheduling et API de base.
- **Spark SQL**: manipulation de données structurées via des requêtes SQL.
- **Spark Streaming**: traitement de flux de données en temps quasi-réel.
- **MLlib**: librairie de machine learning distribuée.
- **GraphX**: librairie pour le traitement distribué de graphes.

Architecture d'une Application Spark

- **Driver**: gère le `SparkContext`, planifie les tâches et collecte les résultats.
- **Cluster Manager**: alloue les ressources (nœuds, mémoire, CPU).
- **Executors (Workers)**: effectuent les tâches de calcul réparties.

1. se connecte à un gestionnaire de cluster qui alloue des ressources aux applications (données, etc.).
2. récupère des exécuteurs sur les nœuds de travail.
3. envoie le code de l'application aux exécuteurs.
4. envoie les tâches aux exécuteurs pour exécution.
5. supervise l'exécution des tâches.

Hugo Tondenier



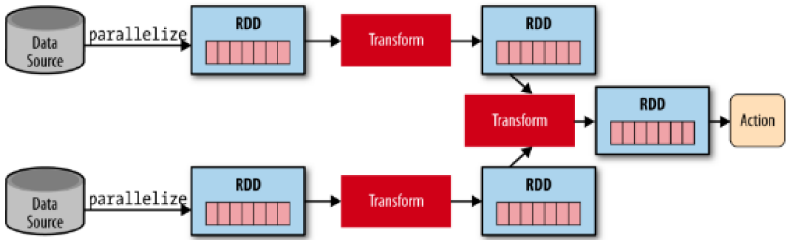
RDD : Resilient Distributed Datasets

Définition

- Structure de données distribuée, immuable et tolérante aux pannes.
- Permet l'exécution de transformations (*map*, *filter*, *reduceByKey*, etc.) et d'actions (*collect*, *count*, etc.).
- **Évaluation paresseuse** (*lazy evaluation*): l'exécution ne démarre qu'à l'appel d'une action.

Transformations et Actions

- **Transformations**: retournent un `RDD` (ex: `map()`, `filter()`, `reduceByKey()`, `flatMap()`...).
- **Actions**: déclenchent réellement le calcul et renvoient une valeur (ex: `collect()`, `count()`, `saveAsTextFile()`).



Tolérance aux pannes

- Spark peut reconstruire les RDDs perdus en rejouant les transformations grâce au DAG (*Directed Acyclic Graph*) des opérations.

Programmation en PySpark

Qu'est-ce que PySpark ?

- API Python pour Spark, facilitant l'écriture de jobs distribués.
- Interactive et adaptée pour le prototypage rapide (shell `pyspark`).

SparkContext

- Objet principal (`sc`) permettant la connexion à un cluster Spark ou à un mode local (ex: `local[*]`).

Création de RDD en Python

- `sc.parallelize([...])` depuis une collection en mémoire.
- `sc.textFile("chemin_vers_fichier")` depuis un fichier local ou HDFS.

Fonctions lambda

- Fonctions anonymes (ex: `lambda x: x+1`).
- Souvent utilisées dans `map()`, `filter()`, etc. pour des transformations rapides.