# Spark : RDD et Dataframe

Note: ref = `https://sparkbyexamples.com/pyspark-rdd` ref2 = `https://github.com/spark-examples/pyspark-examples`

---

## RDD et Dataframe

Spark permet de gerer et manipuler les données sous 2 formes :

- Le Resilient Distributed Dataset (RDD)
- Le Dataframe

---

## RDD et Dataframe

- Le **RDD** et le **Dataframe** sont des structures utilisées pour stocker et manipuler les informations en mémoire de manieres distribuées.
- Dans un RDD ou un Dataframe, les données sont segmentées en morceaux appelés **partition**.
- Les **partitions** d'un RDD ou d'un dataframe sont réparties sur les differentes machines du clusteur.

---

## RDD et Dataframe

Cette structure de données qui permet :

- Une parralelisation des données (traitement distribué)
- Une tolérance aux pannes (stokages distribués et redondance, selon les configurations)
- Traitement in-mémory rapide

---

## RDD et Dataframe

Ils sont tous les deux fabriquer à partir d'une source de données chargées en mémoires (pas d'alteration de la source de données).

Ils sont tous les deux immutables (ne peuvent pas être modifiés une fois créés).

Toutes les transformations sur les RDD ou les Dataframe créent un nouveau RDD ou un nouveau Dataframe.

---

## Repartition et Coalesce

Par default, spark sépare les données en un nombre de partitions selon les ressources du cluster.

En generale, égal au nombre de coeurs de calcul du cluster pour maximiser le parralelisme et donc l'efficacité du traitement.

---

## Repartition et Coalesce

Le nombre de partitions et leurs répartitions du cluster peuvent être modifiées à l'aide des fonctions :

- **repartition()**
- **coalesce()**

---

La methode **repartition()**

> Permet de redefinir le nombre de partitions d'un RDD ou d'un Dataframe.

La methode va recuperer toutes les partitions existantes, les fusionnées puis redécoupé les données en un nombre de partions demandé avant de les redistribué.

```
rdd2 = rdd1.repartition(10)
```

---

La methode **coalesce()**

> Permet de fusionner les partitions d'un RDD ou d'un Dataframe pour en réduire le nombre.

La fusion se fait sur les partitions existantes sans les redistribuer.

```
rdd2 = rdd1.coalesce(5)
```

---

## RDD

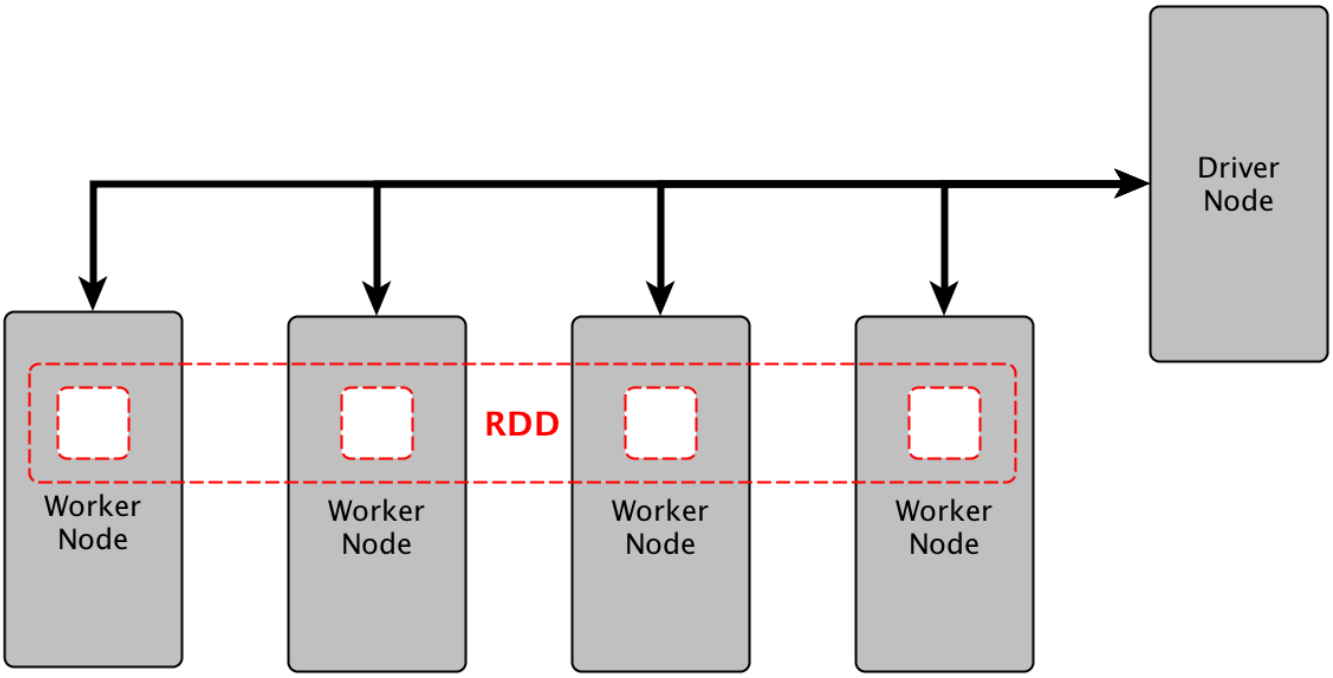
Le **Resilient Distributed Dataset (RDD)**

> C'est une collections d'objets immutables et distribués, qui peuvent être traités en parralele par spark.

Data => RDD = liste de partitions (sur plusieurs machines)

---

## RDD




---

## RDD

Les RDD sont des objets **immutables**, c'est à dire qu'ils ne peuvent pas être modifiés une fois créés !

A chaque manipulation (transformation), un **nouveau RDD** est créé comme résultat de la transformation.

---

## RDD

Spark travail suivant le concept de **lazy evaluation** :

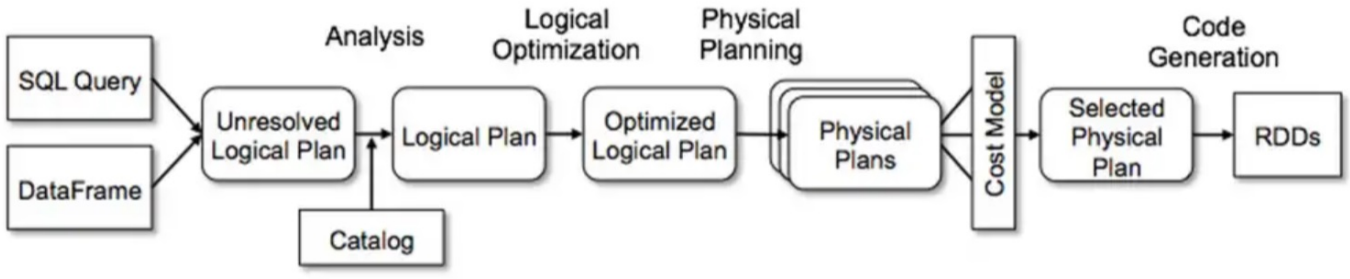
Les **transformations** sur les données ne sont pas effectuées immédiatement, mais sont enregistrées dans une séquences d'opérations à effectuer utltérieurement.

Spark creer un RDD logique ne contenant pas les données mais la logiques de leur transformation.

---

## RDD

Spark garde alors en mémoire la liste des transformations, appelé **plan d'execution** à effectuer sur les données sans les effectuées (ex : map, filter, etc...).

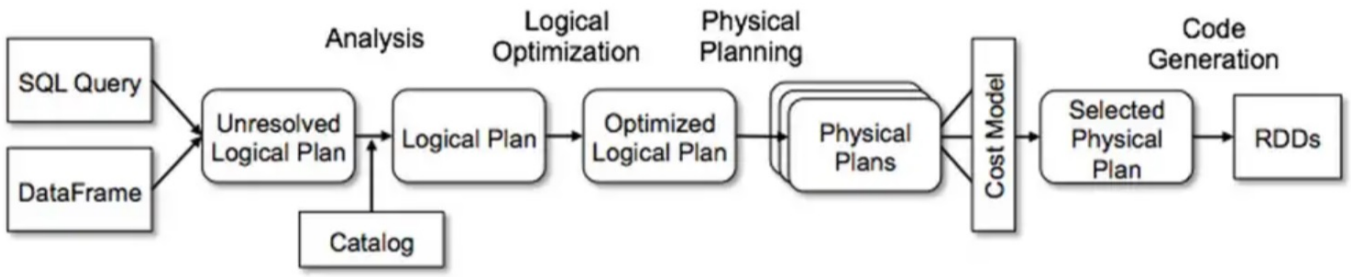



---

## RDD

Ce n'est qu'au moment d'opération d'**action** (ex : collect, count,save, etc...) que les données sont chargées en mémoire et que les transformations sont effectuées d'un bloc.

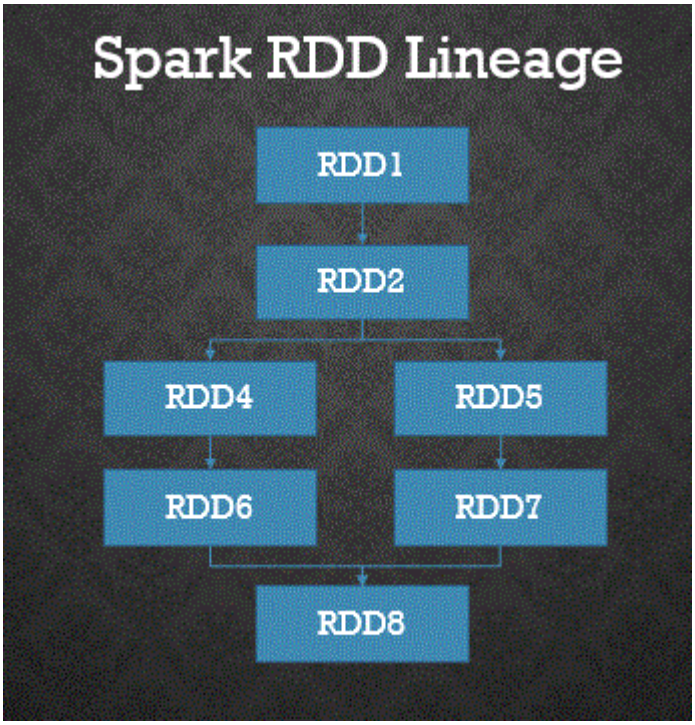
(etape parfois appelé la materialisation).




Note: Optimise le temps de calcule donc la consomation de mémoire pour éviter la redondance des etapes.

---

## RDD




---

## RDD

La creation d'un RDD peut se faire de deux manières :

- en parralelisant une collection existante
- en chargeant un fichier distribué (HDFS, S3, etc...)

---

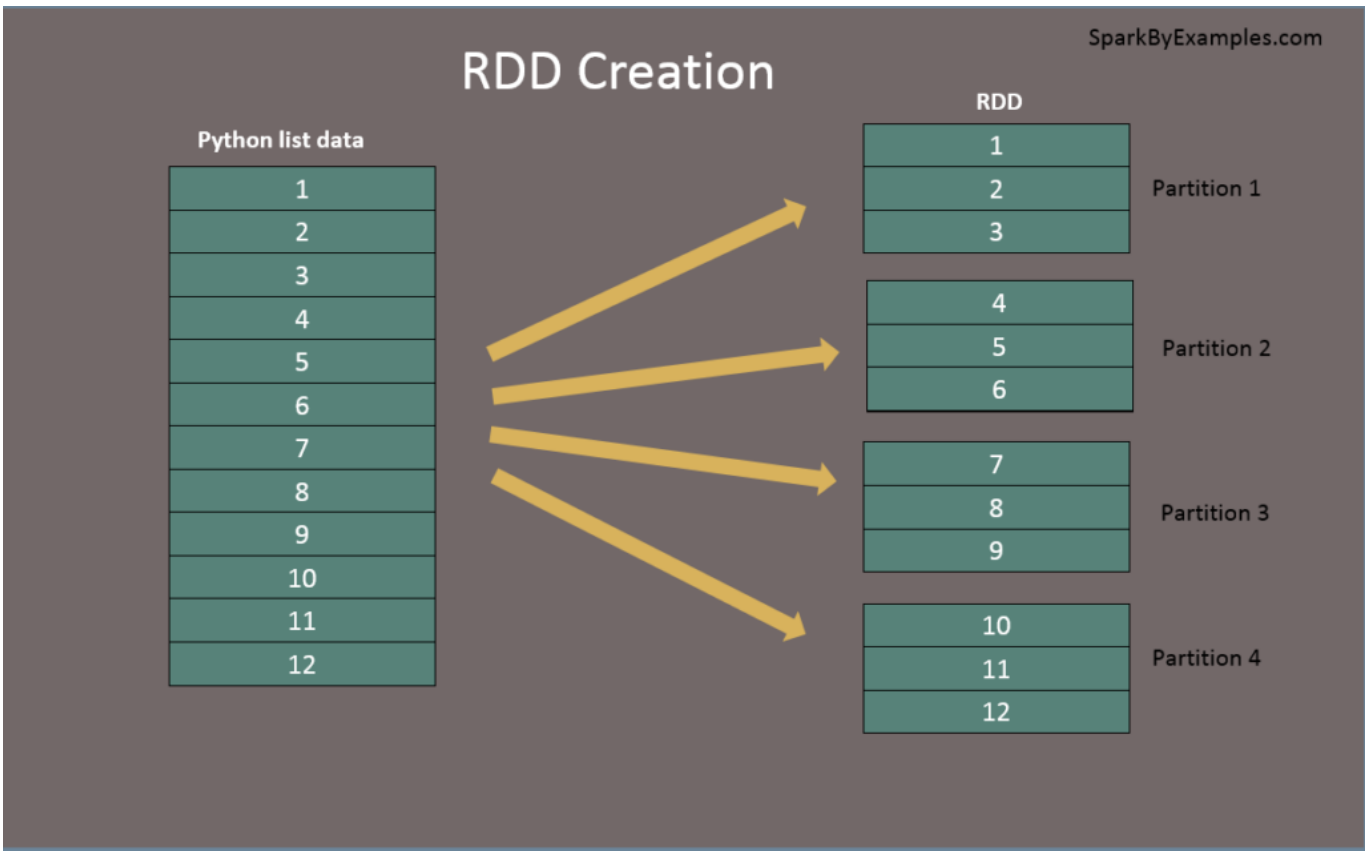
## Creation par parralelisation

> Charger des données en mémoire en les découpant en blocs appelés **partitions** (faisant partie d'un meme RDD).

Ces blocs seront ensuite répartis sur les differentes machines du cluster.

---

## Creation par parralelisation




---

## Creation par parralelisation

Pour creer un RDD par parralelisation, on utilise la methode **parallelize()** de la classe **SparkContext**.

```
# creation d'un RDD par parralelisation
data = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
mon_rdd = spark.sparkContext.parallelize(data, 2)
```

Parametres :

- **Iterable** correspondant au données
- **Nombre de replicats** (optionel) de chaque partitions.

---

## Creation par fichier distribué

Pour charger des données depuis une source externe, on utilise les methodes :

- **textFile()** de la classe **SparkContext**
- **wholeTextFiles()** de la classe **SparkContext**

Remarque : il est possible de travailler avec des fichiers locaux, distribués ou compressés.

---

## Creation par fichier distribué

Utilisation de la methode **textFile()** :

Permet de lire un fichier ou répertoires de fichiers et les chargés dans un RDD contenant la collections de lignes composant le fichier.

```
# creation d'un RDD par chargement de fichier
mon_rdd = spark.sparkContext.textFile("mon_fichier.txt")
# exemple un fichier distribué sur HDFS
mon_rdd = spark.sparkContext.textFile("hdfs://mon_hdfs/mon_fichier.txt")
# fixe le nombre de partitions
mon_rdd = spark.sparkContext.textFile("mon_fichier.txt", 2)
```

---

## Creation par fichier distribué

Utilisation de la methode **wholeTextFiles()** :

Permet de lire un répertoire de fichiers et le chargés dans un **PairRDD** contenant une collection **clé/valeur**.

- **clé** : chemin du fichier
- **valeur** : contenu du fichier

```
# creation d'un RDD par chargement de repertoire
mon_rdd = spark.sparkContext.wholeTextFiles("mon_repertoire")
# exemple un repertoire distribué sur HDFS
mon_rdd = spark.sparkContext.wholeTextFiles("hdfs://mon_hdfs/mon_repertoire")
# fixe le nombre de partitions
mon_rdd = spark.sparkContext.wholeTextFiles("mon_repertoire", 2)
```

---

## RDD : les transformations

Les transformations sont des opérations qui permettent de créer un nouveau RDD à partir d'un RDD existant sans charger les données en mémoire (lazy evaluation).

---

les plus courantes sont :

```
# methode qui applique une fonction à chaque element du rdd
# retourne un nouveau RDD contenant autant d'element que le RDD d'origine
rdd2 = rdd.map(func)
# methode qui applique une fonction à chaque element du rdd
# retourne un nouveau RDD pouvant contenir plus d'element que le RDD d'origine
```

```
# aggreger dans le meme RDD => applatis les resultats dans un meme RDD
rdd2 = rdd.flatMap(func)
# methode qui permet de selectionner des elements dans un RDD
rdd2 = rdd.filter(func)
# methode qui permet d'echantillonner un RDD
rdd2 = rdd.sample(withReplacement, fraction, seed)
# methode qui permet de recuperer uniquement les elements uniques d'un RDD
rdd2 = rdd.distinct([numTasks])
# methode qui permet de regrouper les valeurs d'un RDD posedant des clés identiques. RDD
(clé/valeur) => RDD (clé, [valeur])
rdd2 = rdd.groupByKey([numTasks])
# methode qui permet d'appliquer une fonction d'aggregation aux valeurs d'un RDD posedant des
clés identiques. RDD (clé/valeur) => RDD (clé, valeur)
rdd2 = rdd.reduceByKey(func, [numTasks])
# methode qui permet de recuperer dans un RDD les elements en communs
rdd2 = rdd.intersection(otherDataset)
# methode qui permet d'ajouter les elements de deux RDD dans un nouveau RDD
rdd2 = rdd.union(otherDataset)
...
```

Note: les transformations sont séparées en deux catégories :

- les **Narrow transformations** : les données sont conservées dans les partitions d'origine
- les **Wide transformations** : les données sont redistribuées dans des nouvelles partitions

---

## RDD : les actions

Les actions sont les opérations qui declanche le traitement des données et qui permettent de recuperer les resultats.

les plus courantes sont :

```
# methode qui permet de recuperer le nombre d'element d'un RDD
rdd.count()
# methode qui permet de recuperer les elements d'un RDD dans une liste python
rdd.collect()
# methode qui permet de recuperer le premier element d'un RDD
rdd.first()
# methode qui retourne l'element  qui a la plus grande valeur
rdd.max()
# reduit les element d'un RDD
rdd.reduce()
...
```

---

## Le dataframe

> C'est une structure bi-dimensionnelle, immutable et distribuée, de données structurées.

C'est une surcouche au Resilient Distributed Dataset (RDD) qui permet de **structurer** les données en lignes et colonnes.

---

## Le dataframe

Le dataframe est donc un RDD avec un schema de données organisé en lignes et colonnes.

Le format est tres proche des formats de données SQL, excel, csv, json, ...

```
+---------+----+----------+--------+-------+---------+-------------+-------+---------+----------+
|Direction|Year|      Date| Weekday|Country|Commodity|Transport_Mode|Measure|    Value|Cumulative|
+---------+----+----------+--------+-------+---------+-------------+-------+---------+----------+
|  Exports|2015|01/01/2015|Thursday|    All|      All|          All|      $|104000000| 104000000|
|  Exports|2015|02/01/2015|  Friday|    All|      All|          All|      $| 96000000| 200000000|
|  Exports|2015|03/01/2015|Saturday|    All|      All|          All|      $| 61000000| 262000000|
|  Exports|2015|04/01/2015|  Sunday|    All|      All|          All|      $| 74000000| 336000000|
|  Exports|2015|05/01/2015|  Monday|    All|      All|          All|      $|105000000| 442000000|
+---------+----+----------+--------+-------+---------+-------------+-------+---------+----------+
only showing top 5 rows
```

---

## Le dataframe

Le dataframe possede donc des caracteristiques communes à un RDD :

- **Immutabilité** : une fois créé, il ne peut pas être modifié
- **Distribution** : les données sont distribuées sur plusieurs noeuds
- **Lazy evaluation** : les transformations sont executées à la demande

---

## Le dataframe

Avantages par rapport aux RDD :

- **Schema** : les données sont structurées
- **Performance** : optimisation des opérations de calcul

---

## Le dataframe

Avantages par rapport aux RDD :

- **Manipulation** : les methodes sont proches des requetes SQL ce qui les rend plus facile d'utilisation
- **Interoperabilité** : les dataframes peuvent etre fabriqués à partir de, ou convertis en d'autres structures de données (RDD, pandas, SQL, ...)

---

## Le dataframe

Inconvénients par rapport aux RDD :

- **Flexibilité** : les dataframes utilises des données structurées et sont moins flexibles que les RDD.
- **Performance** : les dataframes sont optimisées pour les opérations de requetes, le RDD est plus performant pour certaines opérations de base comme le filtrage ou certaines transformations.

---

Utilisation RDD