L'univers Spark

ESILV

nicolas.travers (at) devinci.fr

Transparents inspirés de Philippe Rigaux, Raphaël Fournier S'niehotta & Nicolas Travers

`http://b3d.bdpedia.fr/spark-batch.html?highlight=spark`

Qu'est-ce que Spark ?

Moteur d'exécution basé sur des opérateurs de haut niveau (Pig, Hive, etc.)

Repose sur *Map/Reduce* pour l'exécution distribuée (Yarn/Hadoop).

Nouveau concept de collection résident en mémoire, les RDD/DataFrame

⇒ Améliore les traitements, spécialement les itérations.

De nombreuses librairies d'interrogation (`Spark SQL`), de fouille de données (`MLib`), de traitement de graphes, etc.

Versions

v0.1 2010

v0.2 2011 - `RDD`

v0.6 2012 - compatibilité avec `Java` / `Hadoop 2`

v0.8 2013 - `DataFrame`, `Yarn`, `Spark Streaming` & `PySpark`

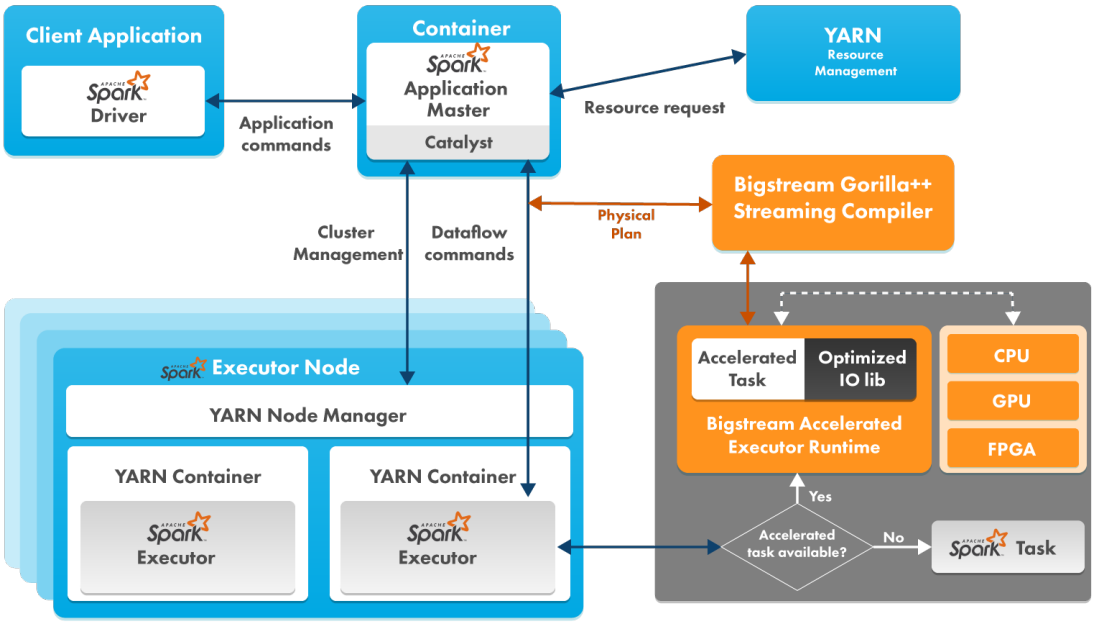
v1.0 2014 - `Spark SQL`, Apache

v1.4 2015 - `DataSet`, `SparkR`

v2.0 2016 - API usability (DataFrame), SQL 2003, perf, structured streaming, `R` UDF

v3.0 2020 - Adaptive query execution, dynamic partition pruning, ANSI SQL, pandas APIs, accelerator-aware scheduler

Architecture



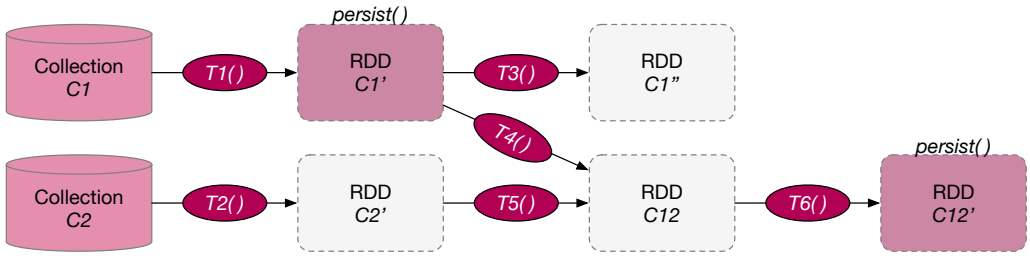
Les *Resilient Distributed Datasets* (RDD)

C'est le concept central : Un RDD est une collection calculée à partir d'une source de données (MongoDB, un flux, un RDD).

- `RDD persistant` : placé en mémoire RAM et conservé par Spark
- L'historique des opérations constituant un RDD est préservé
  - Permet la reprise sur panne, reconstruction du flux d'opération
- Un RDD est un "bloc" non modifiable
  - Si nécessaire il est entièrement recalculé.

Un *workflow* avec RDD dans Spark

Des transformations/opérateurs créent des RDD à partir d'une ou deux sources de données.



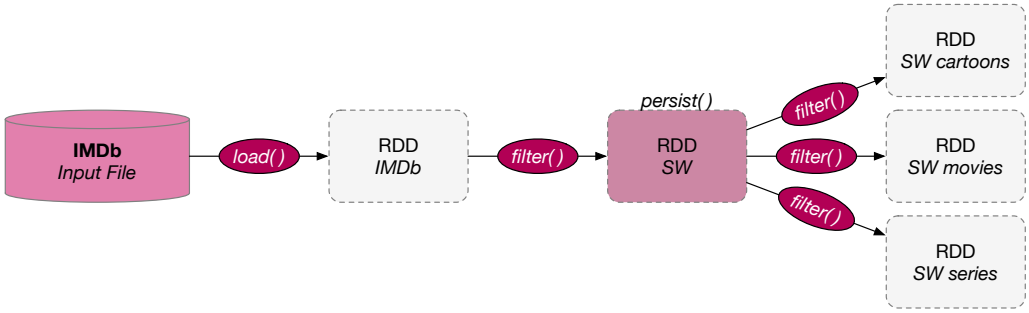
Les RDD persistants sont préservés en mémoire RAM, et peuvent être réutilisés par plusieurs traitements.
Le *workflow* généré doit être un DAG (Directed Acyclic Graph).

Exemple : analyse de films

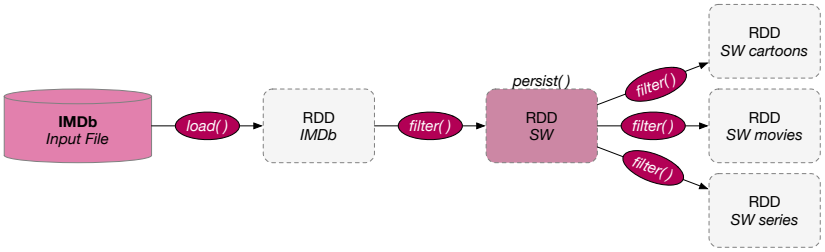
On veut analyser les informations de IMDb particulièrement sur un univers dédié (SW).

On construit un programme qui charge le fichier, ne conserve que les instances de l'univers *SW* et les analyse.

On peut analyser par séries, par films, par dessins animés, etc.



Spécification avec Spark

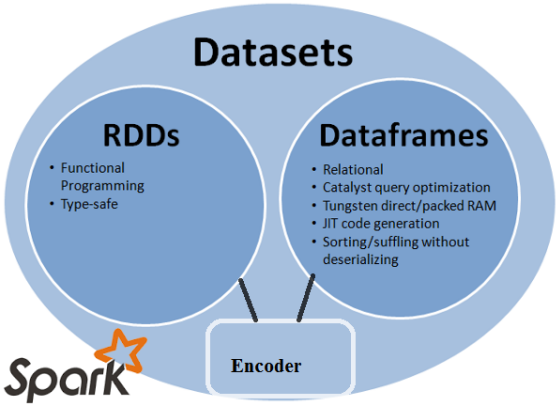



```
// Chargement de la collection
IMDb = sc.textFile("IMDb.csv")
// Filtrage du contenu de IMDb avec "Star Wars" dans chaque ligne (Lambda expression)
SW = IMDb.filter(lambda line: "Star Wars" in line)
// On rend SW persistant !
SW.persist();
```

Analyse à partir de `SW` :

```
// Filtrage par type de film
SW_cartoons = SW.filter(lambda line: "cartoon" in line)
// .. analyse du contenu
```

DataFrame



DataFrame : RDD + schéma ⇒ `Spark SQL`

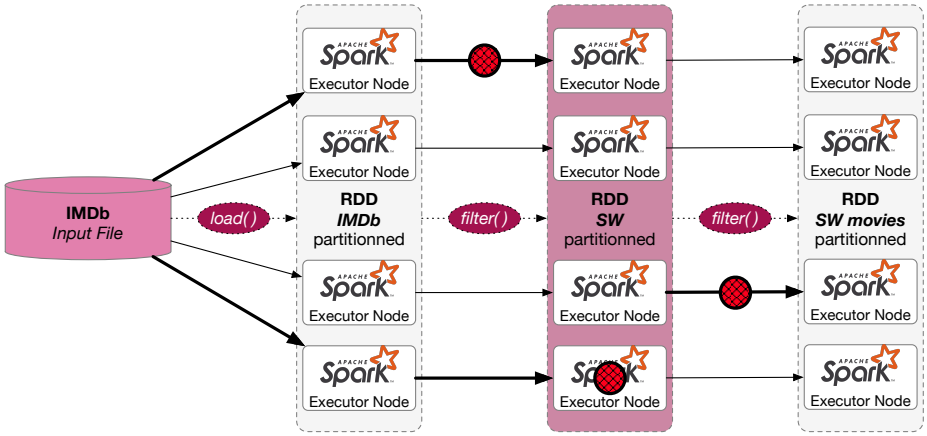
Le plus simple est de lire un fichier CSV avec entête

Manipulation des DataFrame avec des requêtes SQL

```
IMDb = sparkSession.read.option("header",true").csv("IMDb.csv")
```

Reprise sur panne dans Spark

Un RDD est une collection partitionnée (chuncks) et distribuée sur les `Spark nodes`.



Si une panne affecte un calcul s'appuyant sur un fragment F de RDD persistant, il suffit de le relancer à partir de F.

Si une panne affecte un fragment de RDD persistant, Spark ré-applique la chaîne de traitement ayant constitué le RDD.

Scala - un Langage de programmation

C'est le langage d'interrogation d'origine créé à l'EPFL en 2004.
Programmation orientée objet et fonctionnelle.

Appliquable directement dans la console Spark

```
Spark context Web UI available at http://devicimongodbXXX.internal.cloudapp.net:4040
Spark context available as 'sc' (master = spark://devicimongodbXXX:7077,
                        app id = app-20201204211603-0001).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 11.0.9.1)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```


Commandes de base

Chargement d'un fichier texte dans un RDD (1 doc / ligne).

```
val imdb = sc.textFile("IMDb.csv")
```

Quelques actions appliquées au RDD.

```
// Nombre de documents dans ce RDD
imdb.count()
// Le premier document
imdb.first()
// Tous les documents du RDD
imdb.collect()
```

Les transformations (presque comme Pig).

```
val SW = imdb.filter(line => line.contains("Star Wars"))
// Combinaison transf./action
SW.filter(line => line.contains("Star Wars")).count()
```

Le compteur de mots

Comment compter les mots avec Spark :

Extraction des termes (`flatMap`).

```
val terms = SW.flatMap(line => line.split(" "))
```

Initialisation du comptage par mot (`map`).

```
val termUnit = terms.map(word => (word, 1))
```

Regroupement par terme et comptage (`reduce`).

```
val countTerms = termUnit.reduceByKey((a, b) => a + b)
```

Et je peux rendre persistant les compteurs de termes.

```
countTerms.persist()
```


Parallélization avec Spark

Un RDD/DataFrame est une structure en mémoire distribuée sur le cluster Spark. Le traitement est déjà parallélisé.

Pour paralléliser une structure (ex. liste), il faut le donner au *SparkContext*.

```
df = sc.parallelize(Seq(("foo", 1), ("bar", 2)).toDF("k", "v"))
```

toDF $\Rightarrow$ to DataFrame ("key", "value")

Environnement Python/Spark SQL

Possible de manipuler Spark avec Python (ici dans `JupyterNotebook`)

Importation de l'environnement Spark

```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession
```

Création d'une application Spark

```
app_name = "IMDb"
master = "spark://devicimongodbXXX:7077"
conf = SparkConf().setAppName(app_name).setMaster(master)
conf = conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
...
```

Lancement de la session Spark

```
sc = SparkContext(conf=conf)
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```


DataFrame & Importation de données

Importation d'un fichier CSV
⇒ Création d'un DataFrame (*RDD* + *schéma*)

```
path = "./Spark/IMDb_title.basics_short.tsv"
moviesDF = spark.read.format("csv").options(delimiter="\t",header=True,inferSchema=True)
               .load(path)
```

Extraction du schéma

```
moviesDF.dtypes

[ ('tconst', 'string'),
  ('titleType', 'string'),
  ('primaryTitle', 'string'),
  ('originalTitle', 'string'),
  ('isAdult', 'int'),
  ('startYear', 'string'),
  ('endYear', 'string'),
  ('runtimeMinutes', 'string'),
  ('genres', 'string')]
```

L'option "*enforceSchema*" permet de forcer le typage du schéma.

Manipulation de DataFrame en Python

Filtre sur un attribut

```
moviesDF.filter(moviesDF.startYear>2000)
```

Afficher des "lignes"

```
moviesDF.show(1, vertical=True)
```

Grouper, compter et trier

```
moviesDF.groupBy("startYear").count().orderBy("startYear")
```


Spark & jointures

Il est possible d'effectuer des jointures entre *DataFrames*.

```
joinMovies = moviesDF.join(movies_title, moviesDF.tconst == movies_title.titleId,"inner")
```

⚠Cela reste une opération couteuse pour de grandes collections. Penser à rendre certaines parties persistantes et à dimensionner le cluster en conséquence.

Autres types de jointures : *Outer, Full, Fullouter Join*

Spark SQL

Création de "tables" à partir d'un DataFrame à manipuler en SQL.

Transformer en table relationnelle temporaire :

```
moviesDF.createOrReplaceTempView("movies")
```

Création de la table "*movies*" à utiliser dans le `FROM`.

⚠La collection (RDD) n'est pas indexée comme dans une BDD
→ Dans ce cas, faire la requête sur la BDD en premier, *puis* utiliser Spark.

Spark SQL : Exemples

Appliquer une requête SQL sur une table temporaire

```
SW = spark.sql("SELECT primaryTitle, startYear, titleType FROM movies"\
               "WHERE primaryTitle LIKE 'Star Wars%'")
```

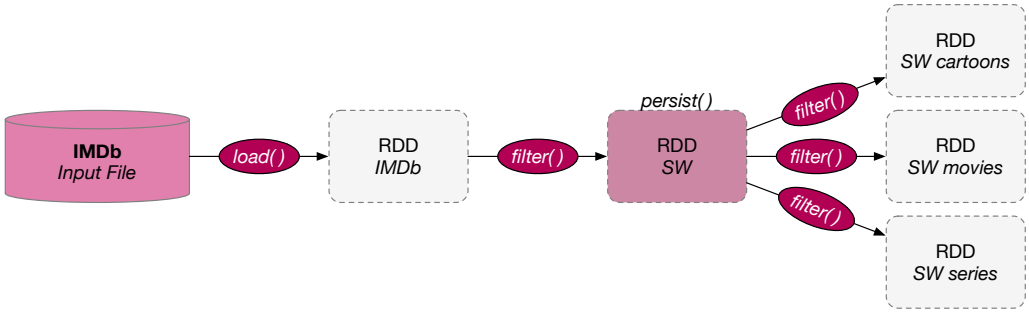
Création d'un RDD persistant à partir du résultat

```
SW.createOrReplaceTempView("SW").persist()
```

Appliquation récursive de requêtes

```
SW_movies = spark.sql("SELECT primaryTitle, startYear, titleType FROM SW"\
                      "WHERE titleType = 'movie'")
SW_cartoons = spark.sql("SELECT primaryTitle, startYear, titleType FROM SW"\
                      "WHERE titleType = 'cartoons'")
SW_series = spark.sql("SELECT primaryTitle, startYear, titleType FROM SW"\
                      "WHERE titleType = 'tvSeries'")
```

Spark Workflow résultat



Connexion à MongoDB

Plutôt qu'une importation classique d'un fichier CSV, il est possible de se connecter à une Bases de données : MySQL, MongoDB, Cassandra, etc. Ou d'un système de fichier distribué : HDFS.

La connexion doit pouvoir se faire à partir de n'importe quel noeud du cluster Spark pour faciliter l'accès distribué.

But de Spark : une surcouche d'exécution au-dessus d'une BDD.

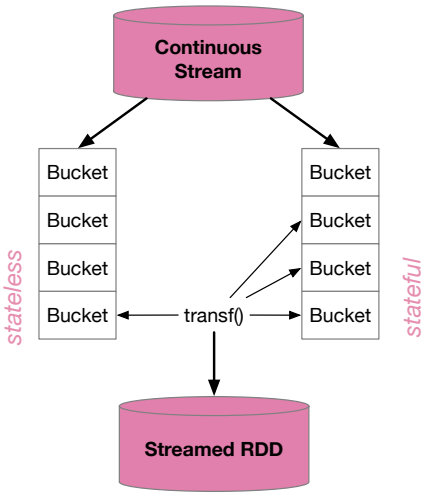
mongo-spark-connector

Pour MongoDB, il faut les drivers MongoDB (package jars), le driver connector Spark/Mongo et établir la connexion dans le *SparkSession* :

```
mongodb_connection = "mongodb://adresse_mongodb:27017/tourPedia.paris?replicaSet=RS1"

conf = conf.set("spark.mongodb.input.uri",mongodb_connection)
conf = conf.set("spark.mongodb.output.uri",mongodb_connection)
conf = conf.set("spark.jars.packages","org.mongodb.spark:mongo-spark-connector_2.12:3.0.0")

sc = SparkContext(conf=conf)
spark = SparkSession.builder.config(conf=conf).getOrCreate()

df = spark.read.format("com.mongodb.spark.sql.DefaultSource")\
        .option("database","tourPedia").option("collection", "paris").load()
```

Streaming : Stateless vs Stateful

Données traitées par paquets (batch) vs par fenêtres (window)



Des opérations dédiées à chaque type :

- Stateless : filter, map, reduceByKey, groupByKey, etc.
- Stateful : SlidingWindow, window, reduceByWindow, countByWindow, reduceByKeyAndWindow, etc.

Changement de *SparkContext* :

```
from pyspark.streaming import StreamingContext
ssc = StreamingContext (sc, batchDuration=5)
#temps en secondes
```


Streaming vs Python

Python n'est pas adapté au streaming :

- *Garbage collector* bloquant
- Gère seul l'allocation mémoire
- Possibilité de swap

Préférer *Scala* :

- Variables fortement typées
- Langage fonctionnel (pas de partage de variables)
- *Garbage collector* en temps-réel
- Programmation avec producteur / consommateur

Spark & Machine Learning

Possibilité d'appliquer des fonctions ML avec Spark [1].

ML distribué sur les DataFrame + MapReduce.

But de Spark : distribuer le calcul du ML.

1. "*Spark - Valorisez vos données en temps réel avec Spark ML et Hadoop*", Romain Jouin, Edition Dunod, 2020 (Chapitre 9)

pyspark.ml

Importation des librairies :

```
from pyspark.ml import clustering
# Gaussian, KMeans, Bisecting, LDA

from pyspark.ml import classification
# TreeClassifier, DecisionTree, RandomForest, GBT, Perceptron

from pyspark.ml import regression
# Linear, Isotonic, DecisionTree, RandomForest, GBT, AFT, etc.

from pyspark.ml import stat
# KolmogorovSmirnoff
```

Les *DataFrame* sont "modifiés" par les méthodes ML en ajoutant des colonnes : features.

Conclusion

Spark est un framework d'exécution distribué

Repose sur une BDD pour l'accès aux données

Création d'un workflow faisant l'interface entre les données et le processus (ML ?)