QUICK REFERENCE CARD

-APACHE- Spark

Appliqué à la Data Science

THERE IS A BETTER WAY

octo.com I blog.octo.com

NOUS CROYONS QUE
l'informatique transforme
NOS SOCIÉTÉS
nous savons QUE
LES RÉALISATIONS
marquantes SONT
LE FRUIT DES SAVOIRS
ET DU PLAISIR À
+RAVAILLER ENSEMBLE
NOUS ***recherchons***
EN PERMANENCE
DE MEILLEURES
façons **DE FAIRE**

OCTO Technology
Part of Accenture

tribu-rad@octo.com I science.at.scale@octo.com

Cette Refcard est imprimée sur du papier certifié PEFC.

Sur quelle architecture fait-on tourner Spark ?

On premise

Pour utiliser le stockage et les ressources (CPU, RAM) répartis entre plusieurs machines, Spark a besoin de deux composants : un gestionnaire de ressources et un système de fichiers distribué. Historiquement, l'écosystème Hadoop fournissait ces deux outils. Pour la gestion de ressources, d'autres alternatives telles que Kubernetes ont émergé depuis.

Spark		**Moteur de traitement des données :** opère sur de grands volumes de données
Hadoop	**YARN**	**Gestionnaire de ressources :** alloue les ressources du cluster Hadoop
	HDFS	**Système de fichiers distribué :** partitionne, réplique et répartit les données
Machines		**Cluster d'ordinateurs :** commodity hardware

Sur quelle architecture fait-on tourner Spark ?

Sur le Cloud

Aujourd'hui, grâce au Cloud, les utilisateurs de Spark n'ont plus besoin d'acheter et de gérer un cluster de machines pour traiter leurs données. Les fournisseurs de Cloud proposent différents types de services, qui permettent de s'affranchir d'un nombre plus ou moins important de composants nécessaires au fonctionnement de Spark.

Clusters standalone / Hadoop / Kubernetes
Il est possible de louer des clusters de machines, sur lesquels on peut installer soi-même Hadoop ou Kubernetes pour faire tourner Spark.

Les services semi-managés
Il est possible de louer un cluster sur lequel Hadoop ou Kubernetes est déjà installé.

Les services managés
Ces services permettent de faire totalement abstraction des composants sur lesquels s'appuie Spark, sans fournir d'accès direct aux machines.

Bienvenue chez Spark

Prenez un ticket et découvrez Spark à travers les différentes attractions !

Pourquoi Spark ?

Spark est un framework de calcul distribué qui permet de traiter des volumes de données dépassant le téraoctet sur un cluster de machines.

Si je veux exécuter une opération de transformation sur un gros volume de données, comment faire pour que plusieurs ordinateurs se partagent la tâche ?

C'est d'abord l'algorithme MapReduce qui a réussi à accomplir cette tâche sur des machines généralistes (commodity hardware). Hadoop est le framework Open Source qui a permis de populariser son utilisation. Spark est venu après pour pallier certaines faiblesses d'Hadoop.

Remerciements

Nous souhaitons remercier nos précieux relecteurs, sans l'aide desquels ce projet n'aurait pu aboutir : Marc Alonso, Meriem Berkane, Nicolas Brohée, Ali El Moussaoui, Augustin Grimprel, Bartosz Konieczny, Matthieu Lagacherie, Thomas Vial. Nous remercions également Camille Rocchi, directrice artistique de la publication.

En espérant que ce Spark d'attraction aura su démystifier certaines notions liées aux calculs distribués et qu'il vous aidera dans vos projets de Data.

Signé, les auteurs principaux : Jennifer Vial, Amric Trudel

Comment fonctionne Spark ?

Les APIs

Spark dispose d'APIs dans cinq environnements différents (Java / Scala / Python / R / .Net).

L'accès et la manipulation des données dans Spark sont possibles grâce à trois types d'abstraction : RDD, DataFrame et Dataset. Aujourd'hui, il est conseillé d'utiliser les deux derniers, plus simples d'utilisation et plus optimisés.

RDD - Spark 1.0
Le **Resilient Distributed Dataset** est la structure primitive de Spark. Cet objet Scala est immuable, et chaque transformation d'un RDD en génère un nouveau.

DataFrame - Spark 1.3
Le **DataFrame** est une interface introduite avec Spark SQL pour faciliter la manipulation des données grâce à ses colonnes nommées.

DataSet - Spark 1.6
Le **Dataset** est disponible en Scala et en Java. C'est un DataFrame qui associe à chacune de ses colonnes un type de données qui sera vérifié à la compilation.

Qu'est ce que Spark apporte au Data Scientist ?

Distribution des calculs
Permet de traiter des volumes importants avec rapidité. Les opérations qui sont facilement parrallélisables, telles que l'optimisation d'hyperparamètres et le traitement de séries temporelles indépendantes, gagnent également en rapidité.

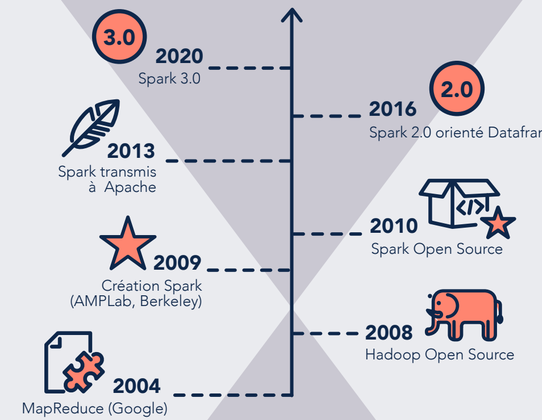
Compatibilité
Spark dispose de nombreux connecteurs vers de nombreuses DataSources ex: csv, parquet, json, ES, C*, JDBC, MySQL, PostGre.

Tolérance aux pannes
Spark recalcule facilement toute partition de RDD perdue en remontant la chaîne de transformations qui l'avait produite.

Un écosystème et des outils propices au Data Scientist

Spark SQL Manipulation des données avec des requêtes SQL.	**Spark ML** Librairie de Machine Learning adaptée pour le calcul distribué.
Spark Structured Streaming Traitement de flux de données.	**Koalas** Réplique les fonctionnalités du DataFrame de Pandas et permet aux Data Scientists d'utiliser avec Spark une syntaxe familière.

L'évolution de Spark



Quelle est la spécificité de Spark ?

L'algorithme MapReduce implémenté par Hadoop stocke tous les résultats intermédiaires sur le disque lors des calculs en plusieurs étapes.

Spark, quant à lui, conserve ces résultats en mémoire vive et y effectue la plupart des opérations de base, ce qui accélère considérablement le traitement.

Attention !

Cet outil doit être utilisé lorsque le volume de données le justifie, car Spark est contraignant sur certains aspects :

1. L'installation, l'industrialisation et le monitoring de clusters Spark demande des compétences OPS et une quantité de travail non négligeable*.

   *Cette difficulté peut toutefois être atténuée en utilisant un service managé dans le Cloud. Voir la section "Sur quelle architecture fait-on tourner Spark?"

2. L'infrastructure distribuée sur laquelle Spark repose est moins performante sur les opérations nécessitant que toute la donnée soit en mémoire sur une seule machine (tri, traitement de séries temporelles).

Quand utiliser SparkML pour le Machine Learning ?

Spark est souvent nécessaire pour le traitement et la transformation des données brutes. Cependant, une fois cette étape passée, il est fréquent que le jeu de données résultant, qui servira à entraîner un modèle, soit suffisamment petit pour permettre l'utilisation d'outils de Machine Learning qui ne tournent pas sur Spark.

Volume du jeu de données d'entraînement encore très important après transformation ?
(de l'ordre du téraoctet)

- Non → Certains algorithmes de Spark ML sont moins précis car ils ont dû être adaptés au calcul distribué. Utiliser Scikit-learn ou TensorFlow.
- Oui → **Algorithme disponible dans Spark ML ?**
  - Non → Utiliser un échantillon des données avec Scikit-learn ou TensorFlow. Spark ML est bien moins mature pour le Machine Learning et le Deep Learning.
  - Oui → Utiliser Spark ML.

Feature Engineering avec PySpark

1• Importer et initialiser PySpark

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession

sc = SparkContext('local')
spark = SparkSession(sc)
```

2 • Charger les données

Charger un fichier CSV dans un DataFrame :

```
df = spark.read.csv('file.csv',
     schema=None, samplingRatio=0.3)
```

schema : schéma du DataFrame à charger

samplingRatio : ratio des données utilisées pour déduire le schéma si ce dernier n'est pas précisé

3 • Visualiser le contenu du DataFrame

Afficher les données du DataFrame :

```
df.show(n, truncate)
```

truncate=True : tronque les chaînes de plus de 20 caractères par défaut

Afficher le schéma du DataFrame :

```
df.printSchema()
```

4 • Explorer les données

Description statistique des données :

```
df.describe()
```

Tableau croisé entre deux colonnes :

```
df.crosstab('col_1', 'col_2')
```

5 • Nettoyer les données

Données manquantes :

```
df = df.na.drop()   # suppression
df = df.na.fill()   # remplacement
```

Données dupliquées :

```
df.dropDuplicates()
```

6 • Sélectionner des données

Sélectionner des colonnes :

```
df.select('col_nom')
```

Filtrer selon une condition :

```
df.filter(condition)
```

Partitionner et agréger :

```
df.groupby('col_nom')
.count()    | .skewness()
.sum()      | .stddev()
.max()      | .variance()
```

7 • Opérations sur les colonnes

Créer une nouvelle colonne :

```
df = df.withColumn('col_nom', fonction)
```

Renommer une colonne :

```
df.withColumnRenamed('nom','nouveau_nom')
```

Supprimer des colonnes :

```
df = df.drop('col_nom_1', 'col_nom_2')
```

Reformater une colonne de dates :

```
df.select(date_format('date_col',
    'MM/dd/yyy').alias('date'))
```

8 • Opérations coûteuses pour Spark

Trier le DataFrame :

```
df.orderBy(col)          # ordre ascendant
df.orderBy(col.desc())   # ordre descendant
```

Faire une jointure :

```
df.join(df_2, col_1 == col_2, how='inner')
```

9 • Rassembler les features en une seule colonne

Le modèle de régression requiert que les features de chaque exemple soient regroupées dans un seul et même vecteur.

Le VectorAssembler est un Transformer qui permet d'assembler plusieurs colonnes du jeu de données en une seule colonne de vecteurs, que l'on nommera features.

```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
 inputCols=['feature_1','feature_2'],
 outputCol='features'
)

data = assembler.transform(df)
```

10 • Diviser le jeu de données entre train et test

```
train, test = df.randomSplit(
    [0.7, 0.3], seed
)
```

11 - Définir un modèle de régression logistique

```
from pyspark.ml.classifiation
import LogisticRegression

lr = LogisticRegression(
   featuresCol='features',
   labelCol='label', maxIter
)
```

12 • Optimiser les hyperparamètres avec une Validation Croisée

```
from pyspark.ml.tuning import
ParamGridBuilder

import CrossValidator

grid = ParamGridBuilder().addGrid(
   lr.regParam,[0.5,0.02]).build()

cv = CrossValidator(
   lr,
   estimatorParamMaps=grid,
   evaluator=BinaryClassificationEvaluator(),
   numFolds
)
```

13 • Entraîner le modèle et prédire les résultats

```
lr_model = cv.fit(train)

predictions = lr_model.transform(test)
```

14 • Évaluer le modèle

```
from pyspark.ml.evaluation import
BinaryClassificationEvaluator

ev = BinaryClassificationEvaluator()

ev.evaluate(predictions)
```

Estimators

Les objets PySpark pouvant traiter les données des DataFrames sont soit des **Transformers**, soit des **Estimators**.

Ils implémentent la méthode **fit()**, qui entraîne un modèle à partir des données d'un DataFrame.

Exemples :

Classification :
- Classification
- LogisticRegression
- DecisionTreeClassifier
- GBTClassifier
- RandomForestClassifier
- NaiveBayes
- MultilayerPerceptronClassifier
- OneVsRest

Régression :
- AFTSurvivalRegression
- DecisionTreeRegressor
- GBTRegressor
- GeneralizedLinearRegression
- IsotonicRegression
- LinearRegression
- RandomForestRegressor

Transformers

Ils implémentent la méthode **transform()**, qui convertit un DataFrame en un autre, en modifiant une ou plusieurs colonnes.

Exemples :

`Binarizer, IDF, CountVectorizer, FeatureHasher, Imputer, OneHotEncoding, StringIndexer, HashingTF`

Conversion en Pandas DataFrame

Si on a réduit suffisamment la taille du jeu de données, l'entraînement peut se faire via Scikit-learn et TensorFlow. Cela implique que les données soient sur la même machine. On peut alors convertir le Spark DataFrame en Pandas DataFrame.

```
df_pandas = df.toPandas()
```

ML Pipelines

Les Transformers et Estimators peuvent être assemblés dans des **Pipelines**.

Ils permettent de réunir plusieurs Transformers et Estimators pour former une séquence d'étapes de transformation. On peut ensuite appeler les méthodes **fit()** et **transform()** directement sur le Pipeline.

```
from pyspark.ml import Pipeline

etapes = []
transformer = CountVectorizer()          # exemple
estimator = LogisticRegression()         # exemple

etapes += [transformer, estimator]
pipeline = Pipeline(stages = etapes)
modele_pipeline = pipeline.fit(df)
nouveau_df = modele_pipeline.transfom(df)
```