

Déployez un modèle dans le cloud

Problématique

Script pyspark

Déploiement

Conclusions

Projet 8

Déploiement d'une chaîne de traitement d'images sur AWS

Maryse Muller | Parcours Data Scientist | 02/02/21

Problématique | une application mobile pour Big Data

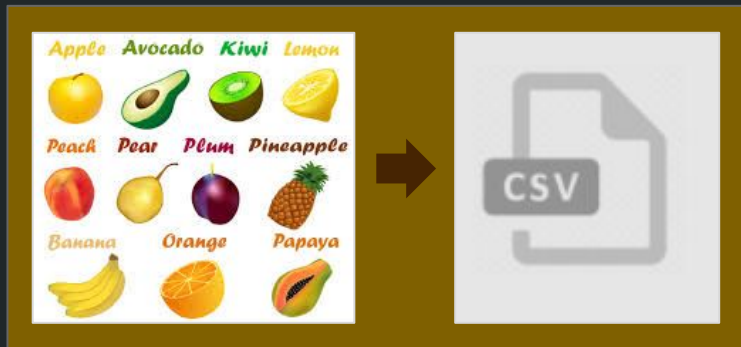
Le besoin :

- Le projet final : une **application mobile de classification d'images** de fruits
- Une **base de données** de travail de nombreux fruits disponible
- Prévion d'un accroissement progressif des données grâce aux utilisateurs
- 1ère étape : on se focalise sur la **conversion des images en données structurées**
- Ce projet doit pouvoir facilement être **mis à l'échelle** ultérieurement

Choix stratégiques :

- Utiliser les outils du clouds (AWS S3) pour le **stockage des données**
- Mettre en place une chaîne de traitement d'images **adaptée au calcul distribué** (script pyspark ou notebook python) qui permettra à terme la classification des fruits
- Construire une première version d'une **architecture Big Data** pour déployer cette chaîne de pré-traitement sur AWS (Instances distantes EC2 ou Cluster EMR couplé à un stockage S3)

Problématique | les grandes étapes



1. Construire un script de traitement des images du jeu de données d'entraînement :

- extraction de features
- réduction de dimension
- extraction de la catégorie
- exportation des données structurées



2. Mise en place d'une architecture sur AWS en vue du déploiement

- stockage S3 des données et des résultats
- exécution du code sur une instance ou sur un cluster distant (EC2 ou Cluster EMR)

Problématique | les données

- Plus de **80 000 Images** couleur sur fond blanc de fruits de 100 x 100 pixels, RGB 256 niveaux
- **séparées en deux groupes :**
 - *Training* 61k
 - *Test* 21k
- **131 catégories** (dossiers), plusieurs centaines d'images du même fruit (angle de vue variable)



Extrait de la classe "Physalis with Husk"

⇒ On extrait de ces données des échantillons de taille croissante (25, 815, 20 000 images) pour la mise au point du script

Problématique | la démarche

1

En local, puis ...

- instanciation d'une machine virtuelle Ubuntu 20.04 (VirtualBox)
- implémentation d'un notebook utilisant pyspark pour le traitement des images (échantillon)
- récupération du script et des datas depuis S3
- export des résultats sur S3
- exécution sur un échantillon de 25 images

2

... sur une instance distante EC2 ...

- installation d'un environnement adapté sur l'EC2 t2.micro, puis t2.2xlarge
- exécution du notebook ou du script sur l'instance distante
- script, datas et résultats stockés sur S3
- exécution sur un échantillon de 815 images
- optimisation du choix des partitions et des ressources executor/driver

3

... puis sur un cluster EMR

- cluster EMR préconfiguré pour l'utilisation de Spark
- utilisation d'un script stocké sur S3
- exécution sur un échantillon de 20k images
- optimisation du choix des partitions et des ressources executor/driver

Script Pyspark | qu'est-ce que Spark ?

- **Quoi ?** Un *framework* (plateforme) open source, multi langages, permettant la parallélisation de tâches sur de grosses quantités de données sur des *clusters*.
- **Qui l'utilise ?** data engineers, data scientists, ML engineers
- **Pourquoi Spark ?**
 - o Facile à utiliser et disponible avec de nombreux langages, dont Python,
 - o plus rapide (utilisation de la RAM) comparé aux solutions antérieures (Hadoop),
 - o permet d'autres opérations que MapReduce,
 - o dispose de librairies ML
 - o permet de faire du traitement en temps réel (real-time)

Script Pyspark | qu'est-ce que Spark ?

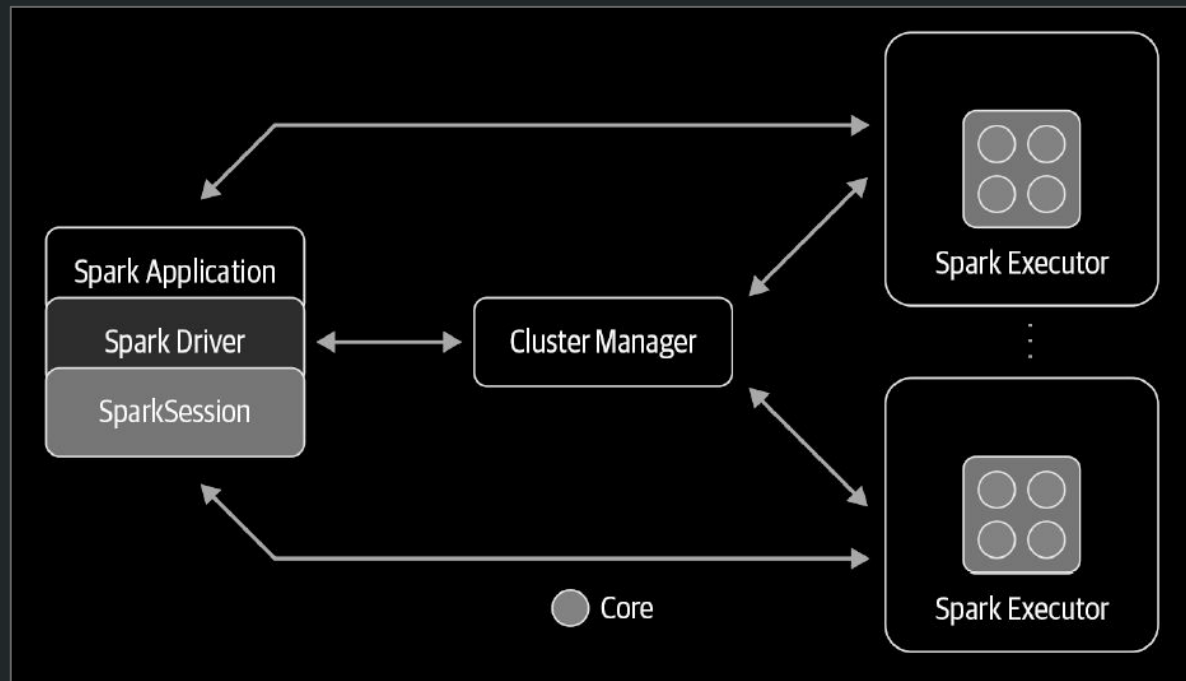
- **Qui l'utilise ?** data engineers, data scientists, ML engineers
- **Pour quoi faire ?**
 - o parallélisation de traitement de gros dataset (distribution sur un cluster)
 - o requêtes interactives pour explorer et visualiser de gros datasets (SparkSQL)
 - o modèles de ML (MLLib, Spark DL)
 - o data pipelines utilisant de nombreux flux de données (Spark streaming)
 - o analyses de "graph datasets" et des réseaux sociaux (GraphX)
- **En pratique ?**
 - o Secteur Bancaire : JP Morgan Chase & Co ⇒ détection de transactions frauduleuses, analyse des dépenses des clients pour la suggestion d'offres adaptées, décisions d'investissement
 - o E-commerce : Alibaba ⇒ analyse de gros volumes de données (détails de transactions et données de navigation en temps réel) pour la recommandation de produits.
 - o Divertissement : Netflix ⇒ recommandations de contenus
 - o Secteur médical : analyse de données médicales pour l'aide au diagnostic

Pyspark | le cluster Spark

Les ressources matérielles disponibles (coeurs d'une machine unique d'un cluster) ou forment un **cluster Spark**

Un **programme pilote** (*driver program*) sur le noeud master orchestre la parallélisation des opérations sur le cluster Spark.

Le *driver* accède aux éléments du *cluster spark* (**cluster manager** et **exécuteurs**) par le biais de l'objet *SparkSession*



Les composants du cluster Spark
(Learning Spark 2nd ed. - J.S. Damji, B.Wenig, T. Das, and D. Lee)

Pyspark | des APIs permettant le calcul distribué

3 manières de gérer les données dans Spark : RDDs, Spark DataFrames et Spark Datasets
-> on privilégiera les DataFrames dans ce projet (données structurées, haut-niveau, optimisation implicite)



Opérations sur les RDD représentées par un graphe orienté acyclique (DAG)

Deux types d'opérations peuvent être appelées sur ces objets :

❑ les transformations

(map; filter, join, union)

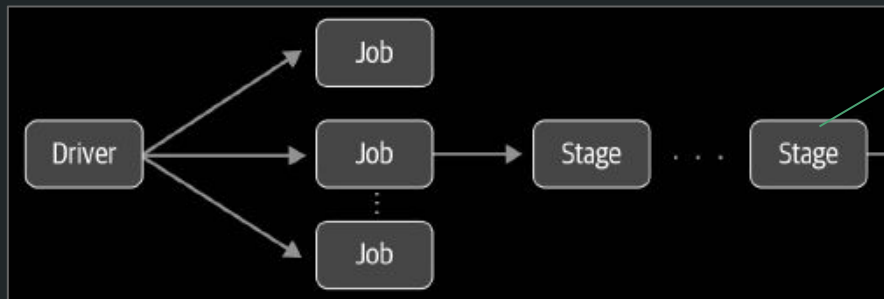
aboutit à une nouvelle RDD

❑ les actions

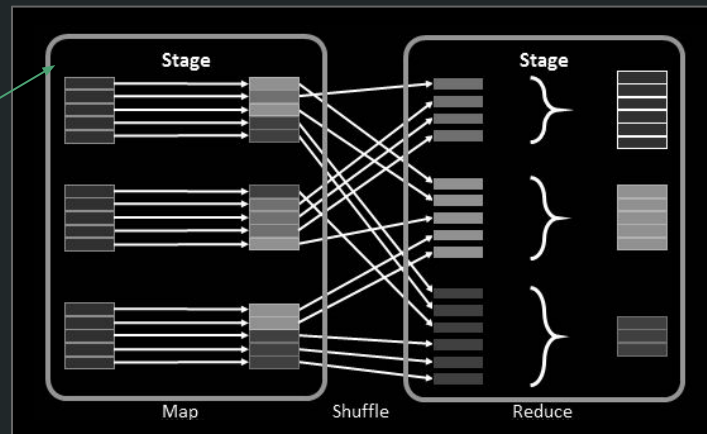
(reduce, first, count, collect)

aboutit à l'exécution d'un calcul concret et renvoie une valeur

Pyspark | l'application et les jobs Spark



Chaque **application** Spark est un **ensemble de jobs**
Chaque **job** correspond à une **action** sur une RDD
→ chaque job on peut associer un DAG
Chaque **job** est découpé en **étapes** (stages)



Les étapes sont **délimitées par des shuffle**
Chaque étape est subdivisée en **tâches** (tasks), basées sur le **nombre de partitions**.

Pyspark | les modes de déploiement

Un programme pilote (driver program) orchestre la parallélisation des opérations sur le cluster Spark/
Le driver accède aux éléments du cluster spark (cluster manager et aux exécuteurs) par le biais de l'objet SparkSession

Mode	Spark driver	Spark executor	Cluster manager
Local	Runs on a single JVM, like a laptop or single node	Runs on the same JVM as the driver	Runs on the same host
Standalone	Can run on any node in the cluster	Each node in the cluster will launch its own executor JVM	Can be allocated arbitrarily to any host in the cluster
YARN (client)	Runs on a client, not part of the cluster	YARN's NodeManager's container	YARN's Resource Manager works with YARN's Application Master to allocate the containers on NodeManagers for executors
YARN (cluster)	Runs with the YARN Application Master	Same as YARN client mode	Same as YARN client mode
Kubernetes	Runs in a Kubernetes pod	Each worker runs within its own pod	Kubernetes Master

Les différents modes de déploiements avec Spark
(Learning Spark 2nd ed. - J.S. Damji, B.Wenig, T. Das, and D. Lee)

Script Pyspark | la chaîne de pré-traitement

Chargement
des images

Extraction de
features

Réduction
dimensionnelle

Extraction des
classes

Exportation au
format parquet

chargement des images
depuis S3

extraction de features par transfer learning sur un
réseau pré-entraîné (ResNet50)

PCA sur l'ensemble des features
(réduction à 8 features)

les classes correspondent au noms de dossier contenant les images

base de donnée contenant la classe, les 8 features et le chemin de l'image

Script Pyspark | le code 1/3

1

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

spark = SparkSession.builder.appName('FeatExtraction').getOrCreate()
sc = spark.sparkContext.getOrCreate()
hadoopConf = sc._jsc.hadoopConfiguration()
hadoopConf.set("fs.s3a.access.key", ID)
hadoopConf.set("fs.s3a.secret.key", KEY)
```

1. Création et configuration de la *SparkSession* et d'un *SparkContext*

⇒ Configuration pour l'accès à S3

2

```
import sparkdl
from pyspark.ml.image import ImageSchema
PREFIX = 'Test'
bucket='ocfruitpictures'
n_dir='data'
data_path = 's3a://{}/{}/{}/'.format(bucket, n_dir, PREFIX)
images_df = ImageSchema.readImages(data_path, recursive=True)
```

2. Chargement des images du bucket S3

⇒ utilisation de la fonction

```
import os
path_cred = os.path.join(os.getcwd(),
                          "Maryse_P8_credentials.csv")
with open(path_cred, 'r') as f:
    msg = f.read()
ID = str(msg).split('\n')[1].split(',')[2]
KEY = str(msg).split('\n')[1].split(',')[3]
```

Script Pyspark | le code 2/3

3

```
from sparkdl import DeepImageFeaturizer
feat = DeepImageFeaturizer(inputCol="image",
                           outputCol="image_features",
                           modelName="ResNet50")
```

4

```
from pyspark.ml.feature import PCA
pca = PCA(k=8,
          inputCol="image_features",
          outputCol="pca_features")
```

5

```
from pyspark.ml import Pipeline
pipe = Pipeline(stages=[feat, pca])
extractor = pipe.fit(images_df)
pca_feat_df = extractor.transform(images_df)
```

3. Extraction de features

⇒ utilisation du package *SparkDL* (Deep Learning Pipelines)

4. PCA sur les features extraites

⇒ utilisation

5. Entraînement et exécution des tâches dans un pipeline pyspark

⇒ exportation au format optimisé par défaut (.parquet)

Script Pyspark | le code 3/3

6

```
import pyspark.sql.functions as pspfunc
orig_col = pca_feat_df['image']['origin']
split_col = pspfunc.split(orig_col, PREFIX+'/')
df_ = pca_feat_df.withColumn('labels', split_col.getItem(1))
split_col = pspfunc.split(df_['labels'], '/')
df_ = df_.withColumn('labels', split_col.getItem(0))
df_ = df_.withColumnRenamed("image", "path")
results_df = df_.select('path', 'pca_features', 'labels')
```

6. Extraction des classes de chaque image

⇒ utilisation des fonctions du module
pyspark.sql.functions

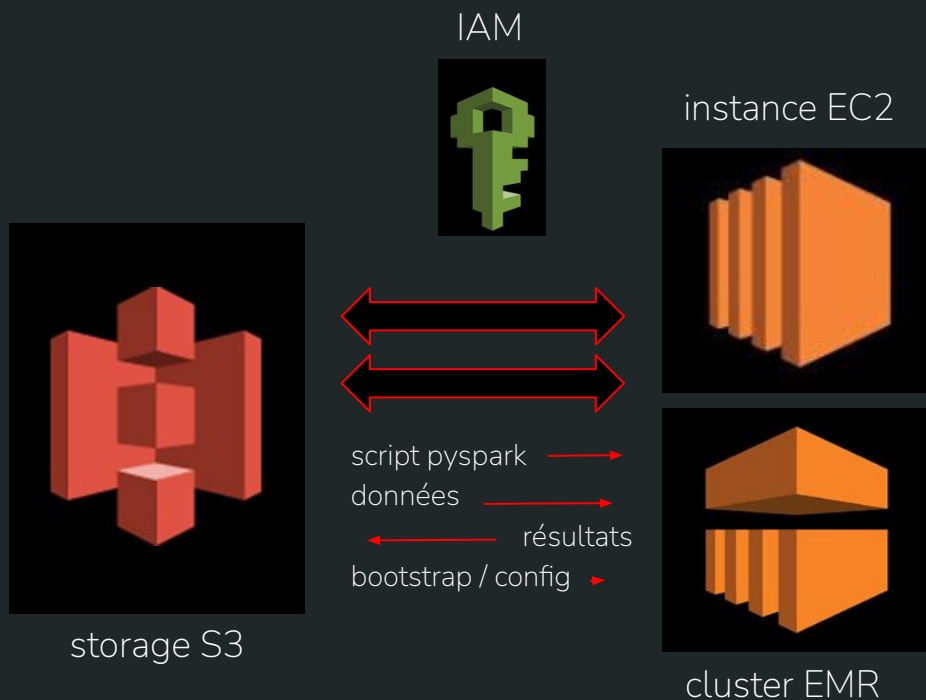
7

```
path_res = "s3a://ocfruitpictures/results/my_res"
results_df.write.mode('overwrite').parquet(path_res)
```

7. Exportation des résultats dans S3

⇒ exportation au format optimisé par défaut (.parquet)

Déploiement | les services AWS









- EC2 : Elastic Cloud Computing
- S3 : Simple Storage Service
- IAM : Identity and Access Management
- EMR : Elastic Map Reduce une plateforme


Déploiement | le stockage des données sur S3

Compartiments (1)

Les compartiments sont des conteneurs pour les données stockées dans S3. [En savoir plus](#)

  Copier l'ARN  Vider  Supprimer  Créer un compartiment

< 1 > 

	Nom ▾	Région AWS ▾	Accéder ▾	Date de création ▾
	ocfruitpictures	USA Est (Virginie du Nord) us-east-1	Les objets peuvent être publics	24 Feb 2021 10:40:21 AM GMT

Contenu du stockage sur S3 :




- les données (Samples, Test, Train)
- le notebook
- les scripts
- les fichiers de configurations pour le cluster EMR (config.json, bootstrap.sh)
- les logs
- les résultats (fichiers parquet)

ocfruitpictures

Objets (9)

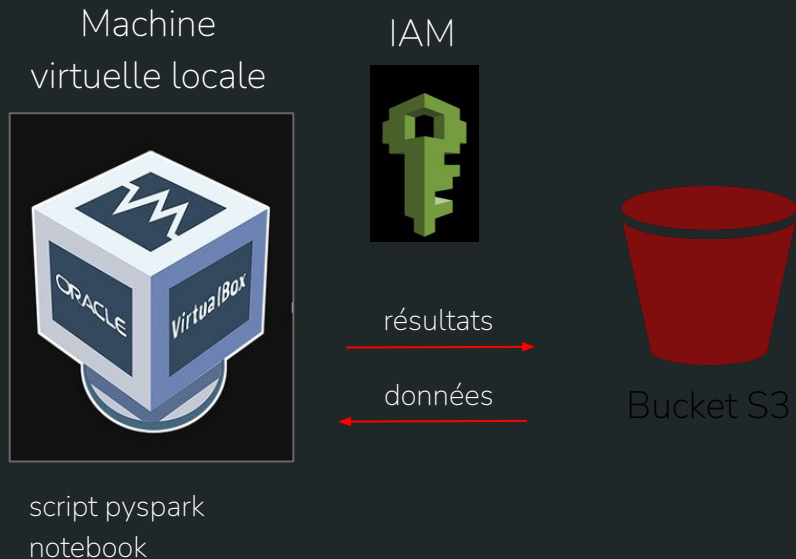
Les objets sont les entités fondamentales stockées dans Amazon S3. Vous pouvez utiliser l'[inventaire Amazon S3](#) pour obtenir de votre compartiment. Pour que d'autres personnes puissent accéder à vos objets, vous devez leur accorder explicitement des auto

  Supprimer  Actions  Créer un dossier  Charger

	Nom	Type ▾	Dernière modification ▾	Taille ▾
	 app_local.py	py	23 Mar 2021 03:38:14 PM GMT	2.2 Ko
	 app_s3.py	py	24 Mar 2021 05:32:57 PM GMT	2.4 Ko
	 bashmodel.txt	txt	17 Mar 2021 05:24:32 PM GMT	201.0 o
	 bootstrap.sh	sh	24 Mar 2021 04:49:17 PM GMT	545.0 o
	 configuration_cluster.json	json	22 Mar 2021 09:03:38 PM GMT	467.0 o
	 data/	Dossier	-	-
	 logs/	Dossier	-	-
	 old_scripts/	Dossier	-	-
	 results/	Dossier	-	-

Déploiement | 1 : en local (principe)

- instanciation d'une machine virtuelle Ubuntu 20.04 (VirtualBox)
- implémentation d'un notebook utilisant pyspark pour le traitement des images (échantillon)
- récupération du script et des datas depuis S3
- export des résultats sur S3
- exécution sur un échantillon de 25 images



Déploiement | 2 : sur un EC2 t2x.2xlarge

Instances (2) Informations

Se connecter

État de l'instance

Actions

Lancer des instances

Filtrer les instances

< 1 >

	Name	ID d'instance	État de l'inst...	Type ...	Contrôle des s...	Statu...	Zone de dispo...	Adresses IP ...	Surveillance	Nom du groupe de ...	Nom de clé	Heure de lancement
	EC2_us_east...	i-0f9d0c4f548ec9174	Arrêté(e)	t2.micro	-	Au...	us-east-1b	-	disabled	launch-wizard-2	P8_us_east_...	2021/03/15 16:04 GMT+0
	EC2_t2_xlar...	i-02476e51c45e5bdfc	En cours...	t2.2xlarge	2/2 vérifica...	Au...	us-east-1b	-	disabled	launch-wizard-2	P8_us_east_...	2021/03/26 10:35 GMT+0

- location d'une instance t2x.2xlarge :
32 Go de RAM, 8 CPU (3GHz)
- installation d'un environnement adapté sur l'EC2
 - o Java 8
 - o Spark 2.4.7
 - o Conda
 - o environnement conda avec Python 3.7
- exécution du notebook ou du script sur l'instance distante
- script, datas et résultats stockés sur S3
- exécution sur un échantillon de 815 images
- optimisation du choix des partitions et des ressources executor/driver



Déploiement | 3 : sur un cluster EMR (1/3)

Choix de l'architecture du cluster

Cluster EMR préconfiguré pour l'utilisation de Spark

- utilisation d'un script stocké sur S3
- exécution sur un échantillon de 815 images
- optimisation du choix des partitions et des ressources executor/driver

YARN est le manager des ressources du cluster (cluster resource manager) de Hadoop2.

Déploiement | 3 : sur un cluster EMR (2/3)

Configuration du cluster

Choix des programmes et modules pré-installés : Hadoop, Ganglia, Spark, Livy, Tensorflow

Ajout de programmes et modules supplémentaires à installer via un fichier bootstrap.sh : Pillow, Keras, Wrapt, Pandas (compatibilité avec Spark DL)

Paramètres de configuration (configuration.json) : variables d'environnement et propriétés Spark

Attachement d'une paire de clés AWS

Attachement des rôles par défaut

Groupes de sécurités permettant la communication entre les instances + autorisation entrante de ssh

```
images_df = ImageSchema.readImages(data_path, recursive=True).repartition(10)
```

Déploiement | 3 : sur un cluster EMR (3/3)

Choix des paramètres du *spark-submit*

Ressources du cluster:

3 instances EC2 m5.xlarge

Pour chaque instance :

4 vCPU (16 ECU) 16Go de RAM, 0,192\$/h

Recommandations :

Nombre de partitions : maximum 128 Mo par partition

Avoir plus d'un coeur par exécuteur c'est à dire par JVM

Nombre de coeurs par exécuteurs : de 4 à 6

Garder un coeur et 1Go de RAM par noeud

```
spark-submit --deploy-mode client --master yarn
--packages
com.amazonaws:aws-java-sdk-pom:1.10.34,org.apache.ha
doop:hadoop-aws:2.7.2,databricks:spark-deep-learning
:1.5.0-spark2.4-s_2.11 --deploy-mode client
--executor-memory 7g --num-executors 5
--executor-cores 4 s3://ocfruitpictures/app_s3.py
```

Nombre total de coeurs disponibles : $(4-1) * 3 = 9$

Nombre de coeurs par exécuteurs : 4

Nombre total d'exécuteurs : 2

Mémoire allouée à chaque exécuteur : $(16-1) * 3 / 4 = 11$



Déploiement | 3 : sur un cluster EMR

Ganglia

Module permettant une mesure en temps réel de la performance des clusters Spark



Déploiement | 3 : sur un cluster EMR

Spark UI

Interface web associée à une application Spark permettant d'analyser et d'inspecter l'exécution des jobs Sparks dans un navigateur web.

Conclusions

Dans le cadre de ce projet, nous avons pu apprendre à :

- installer et utiliser pypark dans le cadre d'un projet de machine learning
- communiquer avec une machine distante ou un cluster AWS en ssh
- exécuter un script python de machine learning sur :
 - o une machine virtuelle (**VirtualBox**) locale
 - o une instance **AWS EC2** distante
 - o un **cluster EMR**
- lire et écrire sur un **bucket AWS S3**
- **lancer, configurer et utiliser un cluster EMR** dans le cadre d'une mission simple de machine learning avec Spark.

Limites et améliorations possibles

- **Optimisation du code**
 - analyse fine de SparkUI pour détecter le code redondant
 - optimisation des opérations (mise en cache, persistance en mémoire etc.)
- **Optimisation de la parallélisation et ressources du cluster**
 - partitionnement optimisé en fonction du volume
 - optimisation de l'architecture du cluster choisie
 - optimisation des options du spark submit
 - utilisation de la fonction autoscaling du cluster EMR

Annexe | Les erreurs d'options du spark-submit

Granularité maximale : les exécuteurs les plus petits possibles (le plus grand nombre possible)

Nombre de coeurs par exécuteur (--executor-cores) -> 1 coeur par exécuteur

Nombre d'exécuteur (--num-executors) -> 96 executors ?

Mémoire allouée à chaque exécuteur (--executor-memory) -> 64/16 = 4Go par exécuteur

-> Non, Erreur ! car ainsi on n'utilise qu'une seule JVM par tâche

Granularité minimale : les exécuteurs les plus gros possibles (le plus petit nombre possible)

Nombre de coeurs par exécuteur (--executor-cores) -> 16 coeur par exécuteur

Nombre d'exécuteur (--num-executors) -> 6 executors (1 par machine) ?

Mémoire allouée à chaque exécuteur (--executor-memory) -> 64Go par exécuteur=machine

-> Non car il faut garder de la mémoire pour Hadoop et tous les autres daemons tournant sur les noeuds

Granularité minimale mais en laissant des ressources (proc et ram) pour chaque noeud

Nombre de coeurs par exécuteur (--executor-cores) -> 15 coeurs par exécuteur

Nombre d'exécuteur (--num-executors) -> 6 executors

Mémoire allouée à chaque exécuteur (--executor-memory) -> 63Go par exécuteur

-> Non plus

SOLUTION : entre 4 et 6 coeurs par exécuteurs seulement ! calculer le nombre d'exécuteurs correspondant en fonction des ressources