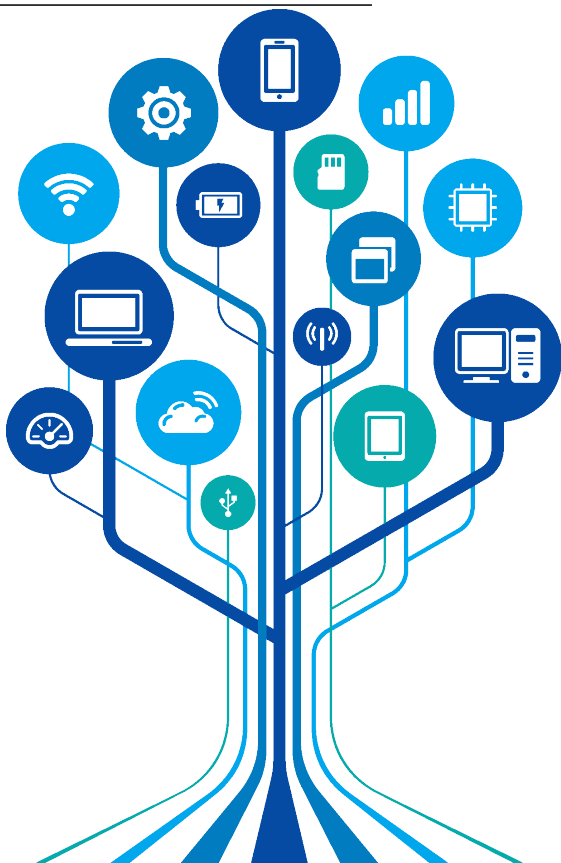
MINES ParisTech

Mastère HPC-AI

Large Scale ML

Projet Spark Dask

Élèves :
Arthur Viens
Professeur :
Jean-Marc Gratien



19 février 2022

1 Installations

Ce projet et cette matière concernaient en grande partie la débrouillardise et la compréhension informatique nécessaire pour faire les installations.
J'ai eu de nombreux problèmes avec ssh lors de ces installations. Il était impossible de se connection en ssh au *localhost* car l'accès était refusé, et un mot de passe était demandé. Après de nombreux essais, modifications, recherches sur internet et aide de votre part, il a été possible de régler les problèmes qui venaient du serveur ssh Windows. La solution fut de couper ce serveur ssh dans Powershell en administrateur en faisant la commande :

```
Stop-service ssh
```

Après redémarrage du service ssh sur Ubuntu, tout fonctionnait.

Les étapes suivantes d'installation concernaient Java, Hadoop, Spark et Anaconda. Cela nous a permis de bien comprendre comment était fait le système de fichier d'hadoop notamment, comment configurer les serveurs, comment régler les erreurs que l'on peut avoir à cause d'une mauvaise installation...
Ce travail est cependant fastidieux, et il est très pratique de pouvoir le factoriser dans un environnement Docker.

2 Spark

Dans Spark, il est possible de réaliser des fonctions en parallèle sur des partitions des données. Par exemple, pour du traitement d'images, il est possible d'appliquer un filtre sur différentes parties d'une image, avant de fusionner les résultats. C'est ce que nous faisons dans part_median_filter.
Voici le résultat de la photo filtrée sur la figure 1 :

FIGURE 1 – Lena : Bruitée (gauche) débruitée avec Spark et Dask (milieu et droite)

Cette filtration s'est fait en partitionnant l'image en 1, 2, 4 et 8 parties. Ceci a eu un effet sur la performance comme le montre la figure 2. Le temps de calcul le plus rapide fut avec 4 partitions pour 0.29s.

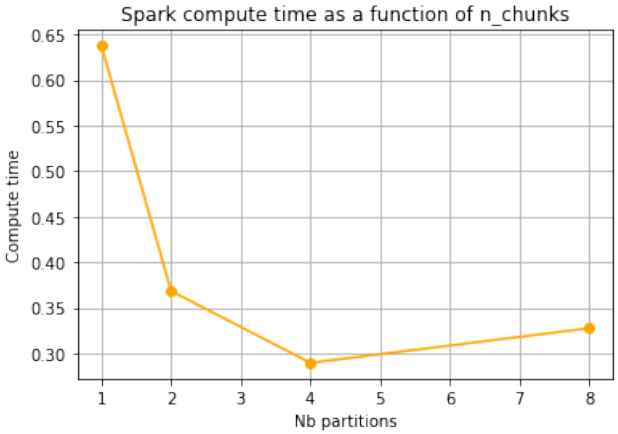


FIGURE 2 – Performance du filtrage médian avec Spark

Concernant le Machine Learning, il a été possible d'implémenter une classification sur le jeu de données Iris avec 3 algorithmes : Decision Tree, Random Forest et GradientBoostedTree sur la table 1. La classification s'est fait en prenant 80% du jeu de données en entraînement et 20% pour le test.

Decision Tree	Random Forest	GradientBoostedTree
97.1%	100%	100%

TABLE 1 – Accuracy des algorithmes de ML

Les Decision Tree et RandomForest acceptaient les sorties sous forme de label (0, 1, 2) mais pas le Gradient Boosted Tree. Afin de réaliser la classification, j'ai donc wrappé l'algorithme dans un objet OneVsRest Classifier.

3 Dask

Le filtrage avec Dask fut plus efficace. En effet, comme on le voit sur la figure 3, le minimum de temps de calcul est atteint pour 0.05s en divisant la matrice en 2 chunks. C'est plus rapide qu'avec Spark.

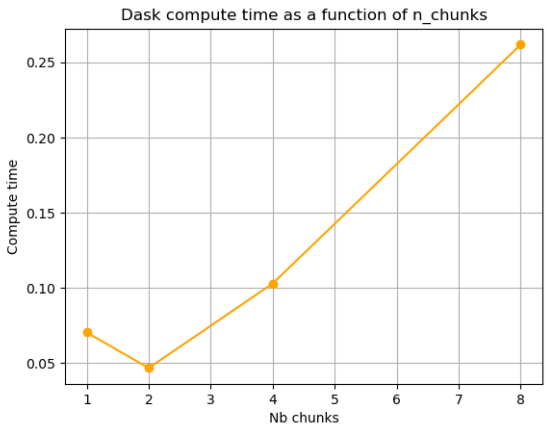


FIGURE 3 – Performance du filtrage médian avec Dask

Pour ce qui est du ML, ce fut plus simple qu'avec Spark. En effet, il est possible d'effectuer des fonctions Scikit-Learn dans un environnement Dask avec la ligne

```
with joblib.parallel_backend('dask'):
    # Sklearn code
```

Les résultats de classification ont été les suivants 2 :

Decision Tree	Random Forest	GradientBoostedTree
93.3%	96.6%	93.3%

TABLE 2 – Accuracy des algorithmes de ML avec Dask