8INF919 – APPRENTISSAGE AUTOMATIQUE POUR LE BIGDATA - HIVER 2022
Professeur A. Bouzouane

UNIVERSITÉ DU QUÉBEC À CHICOUTIMI
Département d'Informatique et de Mathématique

Devoir #1 - Classification distribuée par arbre de décision

MEDM28048006 - Marcelo Medeiros de Vasconcellos

Sommaire

1. Synthèse de l'article

L'apprentissage automatique au moyen d'arbres de décision avec une grande quantité de données n'est pas courant car le système doit effectuer de nombreux calculs. Hadoop est une plateforme informatique distribuée capable de fonctionner sur des grappes de matériel et peut être utilisée dans des tâches d'exploration de données avec de grandes quantités de données. L'article présente un algorithme parallèle d'apprentissage d'arbre de décision supervisé, développé avec le modèle MapReduce qui fonctionne sur Hadoop et présente une bonne évolutivité avec la taille des données.

Nous représentons mathématiquement un modèle par un ensemble de valeurs numériques qui représentent un objet ou un phénomène du monde réel. Cette représentation se présente généralement sous la forme d'une matrice de p valeurs ou, dans le cas d'une base de données, d'un *tuple* de p attributs. Dans le contexte de l'apprentissage supervisé, le but de la reconnaissance des formes est de trouver un objet classificateur capable d'assigner une classe à une nouvelle forme sur la base des informations d'un ensemble d'apprentissage. Dans ce contexte, l'un des plus grands défis est la classification des formes en classes non linéaires séparables. Les arbres binaires sont une bonne approche pour résoudre les problèmes de classification de motifs dans des classes séparables non linéaires.

L'algorithme MR-Tree comporte trois sections : le traitement des paramètres d'entrée, l'induction récursive de l'arbre et l'élagage.

La fonction d'induction est appelée ID3 et reçoit les paramètres : *filters*, *inputPath*, *mostCommonClass* et *attributes*. Lors de son premier appel, les filtres sont vides et le paramètre attributs est complet, si le paramètre attributs est vide, une nouvelle feuille est créée et étiquetée avec la classe la plus courante de l'interaction précédente, sinon il exécute la fonction *MapReduce* calculant :

- l'entropie de la classe
- gain d'informations pour chaque attribut
- la classe la plus courante pour l'ensemble des données actuelles
- le nombre de motifs traités
- la valeur de division (seuil) pour chaque attribut numérique

Si l'entropie de classe est égale à 0 ou n'a pas de paramètres à traiter, l'algorithme renvoie une nouvelle feuille étiquetée avec la classe la plus courante pour l'ensemble de données. Sinon, il choisit l'attribut qui a le gain d'information le plus élevé, crée un nœud et le divise avec la valeur *maxattr*. Cette fonction est récursive et est appelée pour chaque valeur de l'attribut *maxattr.* Dans le cas de valeurs numériques, la fonction est appelée deux fois, la première fois avec des valeurs plus petites ou égales et la seconde fois avec des valeurs plus grandes.

La fonction *map* traite chaque enregistrement selon les filtres passés et calcule la distribution de classe pour chaque attribut, la fonction *clean* sort les paires (clé-valeur) pour chaque attribut et la fonction *reduce* agrège la distribution de classe totale pour chaque attribut.

Dans cet algorithme, ils ont utilisé la méthode de l'erreur réduite, qui consiste à remplacer un nœud par une feuille étiquetée avec la classe la plus courante dans le sous-arbre, où ce

nœud est la racine. Le processus d'élagage est interactif et s'arrête lorsque la prédiction ne s'améliore plus.

Pour réaliser l'expérience, 400 enregistrements ont été sélectionnés au hasard dans l'ensemble de données du US Census Bureau. Ces enregistrements ont été divisés en 200 enregistrements d'entraînement et 200 enregistrements de test. Chaque ensemble de travail a été multiplié par plusieurs fois. Un facteur d'échelle a été défini pour mesurer la taille de l'ensemble de données, ce facteur allant de 1 à 168. Le cluster Hadoop a été configuré avec 13 nœuds, l'un étant le maître et les autres les esclaves. Chaque nœud esclave est composé de deux processeurs Quad Core Xeon de 2 GHz, de 2 Go de RAM et d'un système d'exploitation Linux. Le cluster totam dispose de 96 emplacements *MapReduce*.

Vingt-trois tests distincts ont été effectués pour diverses charges de travail et le temps d'exécution a été enregistré. Les résultats ont montré que l'algorithme a une évolutivité linéaire avec la taille de l'ensemble de données. Cela est dû aux fonctions *MapReduce* qui ont réduit les frais généraux sur les opérations d'entrée et de sortie du cluster Hadoop. En conséquence, il a été identifié que le facteur d'échelle 1 correspond au temps de 40 minutes.

En conclusion, l'excellente évolutivité de l'algorithme avec la taille de l'ensemble de données a été mise en évidence, de sorte que l'algorithme peut être utilisé pour créer des modèles d'arbres de décision avec de très grands ensembles de données dans la plate-forme Hadoop.

2. Introduction

L'ensemble de données utilisé était l'ensemble de données sur les adultes, disponible à l'adresse suivante : https://archive.ics.uci.edu/ml/datasets/adult. Cet ensemble comprend 48 842 échantillons et 14 caractéristiques pour chaque échantillon. L'objectif est de déterminer si la personne gagne plus de 50 000 dollars par an. Dans notre expérience, seules les caractéristiques numériques continues ont été sélectionnées.

2.1. Installation

En pensant au contrôle de l'expérience et à sa reproductibilité, nous avons créé un fichier d'installation shell (install.sh) et un conteneur avec OS Ubuntu 20.04 et installé spark, pyspark et jupyter notebook.

Pour l'installation à l'aide du fichier install.sh sur le système d'exploitation Ubuntu 20.04 :

Vous devez enregistrer les fichiers dans le système d'exploitation et accéder au dossier à partir de la ligne de commande, puis exécuter les commandes suivantes :

Donnez au fichier la permission de s'exécuter

```
chmod 777 install.sh
```

Exécuter le fichier

```
./install.sh
```

Pour effectuer une installation à l'aide de Docker, vous devez dans ce cas disposer d'un bureau Docker installé sur votre ordinateur :

Cette commande va créer l'image du container

```
docker build -t "pyspark-distributed-classification:latest" .
```

Cette commande va exécuter le conteneur en limitant sa mémoire à 8GB. **Attention : Avant d'exécuter docker, assurez-vous que Jupyter-notebook n'est pas en cours d'exécution sur votre ordinateur, si c'est le cas arrêtez l'exécution afin que vous puissiez avoir accès à jupyter-notebook à l'intérieur du conteneur.**

```
docker run -d -p 8888:8888 --memory=8GB
"pyspark-distributed-classification:latest"
```

Vous pouvez également exécuter docker via l'image existante du docker hub, en exécutant simplement la commande suivante.

```
docker run -d -p 8888:8888 --memory=8GB
marcelovasconcellos/pyspark-distributed-classification:latest
```

2.2. Expérimentations

Les expériences ont été menées dans trois environnements différents :

- CLOUD: nuage Digital Ocean (www.digitalocean.com), avec un processeur dédié de 32 cœurs et 64 Go de RAM. Le système d'exploitation choisi était Ubuntu 20.04 LTS 64 bits.
- LOCAL: Ordinateur 3.7GHz quad-core avec 64 GB RAM et SDD 250 GB.
- DOCKER: Container Docker limité à 8 Go de RAM utilisant les 4 cœurs de 3,7 GHz.

Toutefois, les données figurant dans ce rapport ne concernent que les données de l'environnement CLOUD. Cela s'explique par le fait que les processus exécutés dans les environnements LOCAL et DOCKER se sont fermés de manière inattendue lorsque les fichiers des ensembles de données étaient supérieurs à 10 Go, probablement en raison d'un manque d'espace disque (LOCAL et DOCKER). Les données des exécutions dans ces environnements sont accessibles dans les dossiers "logs", "results" et "terminal_output".

2.3. Paramètres

Pour configurer les paramètres, vous devez créer ou modifier le fichier .env à la racine de l'application et inclure les données comme ci-dessous :

```
ENVIRONMENT=LOCAL
NUMBER_OF_CORES=1,2,3,4
MULTIPLICATION_FACTORS=1,10,20,30,40,50,60,70,80,90,100
```

Des exemples de configurations se trouvent dans les fichiers : env_cloud, env_docker et env_local.

3. Garantie du passage à l'échelle

3.1. Expérimentation

Dans cette expérience, nous avons exécuté l'algorithme d'apprentissage en changeant la taille du fichier, dans ce cas nous avons considéré le facteur 1 pour le fichier original et crée un facteur de multiplication qui variait entre 1x et 1000x la taille du fichier original, après quoi l'ensemble de données a été divisé en ensembles de données d'apprentissage (50%) et de test (50%), comme dans l'article. L'expérience a été réalisée avec 32 cores.

3.2. Exécution du script

Le script est exécuté à l'aide de la commande suivante :

```
python3 garantie-du-passage-a-l-echelle.py
```

Les données sont enregistrées dans le dossier "résultats" et peuvent être vérifiées et visualisées à l'aide du *notebook*:

```
garantie-du-passage-a-l-echelle.ipynb
```

Pendant l'exécution des expériences, les données sont présentées à l'écran au moyen de logs et enregistrées dans des fichiers dans le dossier "log/" avec le nom de l'environnement.

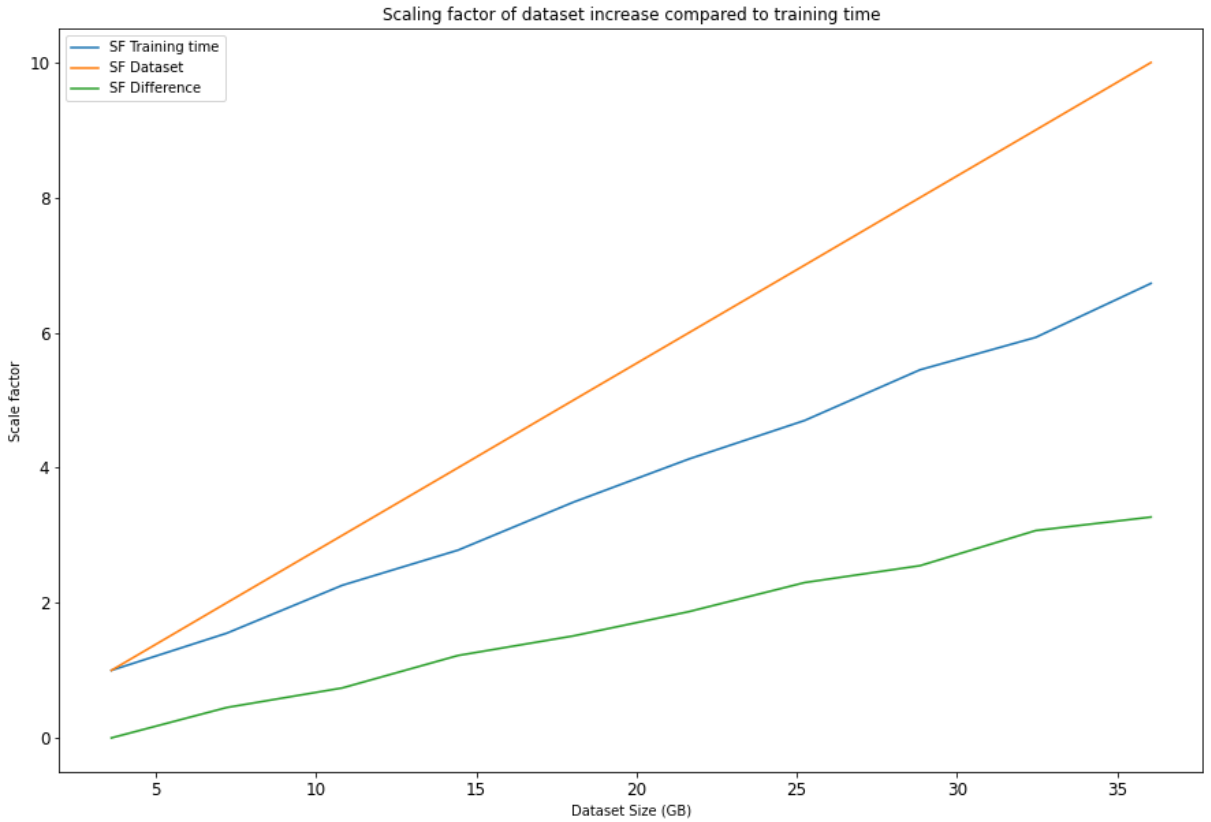
```
2022-10-17 18:26:11,324 [INFO] DecisionTreePySpark : Get metrics
2022-10-17 18:26:12,118 [INFO] Metrics: Clusters 4 - Dataset size 60x - Time 17.92649 seconds
2022-10-17 18:26:12,118 [INFO] Dataset : Delete copy dataset/adult_60x.data
2022-10-17 18:26:12,176 [INFO] Dataset : Starting
2022-10-17 18:26:12,189 [INFO] Dataset : Create copy dataset/adult_70x.data
2022-10-17 18:26:13,238 [INFO] Dataset : Loading Dataset dataset/adult_70x.data
2022-10-17 18:26:13,296 [INFO] Dataset : Loading Pandas Dataset dataset/adult_70x.data
2022-10-17 18:26:17,474 [INFO] Dataset : Select Only Numerical Features
2022-10-17 18:26:17,601 [INFO] DecisionTreePySpark : Starting
2022-10-17 18:26:17,601 [INFO] DecisionTreePySpark : Training
2022-10-17 18:26:17,602 [INFO] DecisionTreePySpark : Setting Labeled Point
2022-10-17 18:26:17,639 [INFO] DecisionTreePySpark : Splitting
2022-10-17 18:26:17,640 [INFO] DecisionTreePySpark : Assembling
2022-10-17 18:26:38,949 [INFO] DecisionTreePySpark : Train time 21.235517 seconds
2022-10-17 18:26:38,949 [INFO] DecisionTreePySpark : Get metrics
2022-10-17 18:26:39,868 [INFO] Metrics: Clusters 4 - Dataset size 70x - Time 21.235517 seconds
2022-10-17 18:26:39,869 [INFO] Dataset : Delete copy dataset/adult_70x.data
2022-10-17 18:26:39,937 [INFO] Dataset : Starting
2022-10-17 18:26:39,952 [INFO] Dataset : Create copy dataset/adult_80x.data
2022-10-17 18:26:41,149 [INFO] Dataset : Loading Dataset dataset/adult_80x.data
2022-10-17 18:26:41,240 [INFO] Dataset : Loading Pandas Dataset dataset/adult_80x.data
2022-10-17 18:26:46,043 [INFO] Dataset : Select Only Numerical Features
2022-10-17 18:26:46,183 [INFO] DecisionTreePySpark : Starting
2022-10-17 18:26:46,183 [INFO] DecisionTreePySpark : Training
2022-10-17 18:26:46,184 [INFO] DecisionTreePySpark : Setting Labeled Point
2022-10-17 18:26:46,220 [INFO] DecisionTreePySpark : Splitting
2022-10-17 18:26:46,221 [INFO] DecisionTreePySpark : Assembling
```

3.3. Résultats

Pour calculer le facteur d'échelle, nous considérons qu'un facteur d'échelle de 1 correspond à 100 fois la taille de l'ensemble de données d'origine. Ainsi, le facteur d'échelle 1 correspond à un ensemble de données de 3,61 Go de données qui a été entraîné en 9,36 secondes. Nous divisons ensuite la taille de l'ensemble de données par le facteur d'échelle 1 de la taille de l'ensemble de données pour identifier la proportion dans laquelle l'ensemble de données augmente au cours de l'expérience, calculant ainsi la colonne "SF Dataset". Nous avons fait de même avec le temps d'entraînement pour calculer la colonne "SF Training Time".

	Dataset Size (GB)	Training time (s)	SF Dataset	SF Training time	SF Difference
0	3.61	9.36	1.0	1.00	0.00
1	7.21	14.47	2.0	1.55	0.45
2	10.82	21.15	3.0	2.26	0.74
3	14.43	26.03	4.0	2.78	1.22
4	18.03	32.69	5.0	3.49	1.51
5	21.64	38.64	6.0	4.13	1.87
6	25.25	43.96	7.0	4.70	2.30
7	28.85	51.05	8.0	5.45	2.55
8	32.46	55.54	9.0	5.93	3.07
9	36.06	62.97	10.0	6.73	3.27

Comme nous pouvons le voir dans le graphique ci-dessous, le facteur d'échelle de l'ensemble de données croît à un rythme plus élevé que le facteur d'échelle du "Training Time".



3.4. Est-ce-que vous avez obtenu les mêmes tendances que l’article ?

Oui, il a été possible d'identifier la même tendance dans le document en observant l'augmentation du jeu de données et l'augmentation de la différence entre les deux facteurs d'échelle, ce qui entraîne une tendance du temps d'exécution à de plus grands volumes de données, car le temps d'exécution n'augmente pas au même rythme que le jeu de données.

4. Nécessité de la distribution de l’apprentissage dans le cas du BigData

4.1. Expérimentation

Dans cette expérience, nous exécutons l'algorithme d'apprentissage en fixant la taille de l'ensemble de données à 1000x la taille originale de la base de données, c'est-à-dire que nous utilisons la base de données de 36,06 Go et nous changeons le nombre de cœurs de traitement en faisant varier sa valeur entre 4 et 32 cœurs.

4.2. Exécution du script

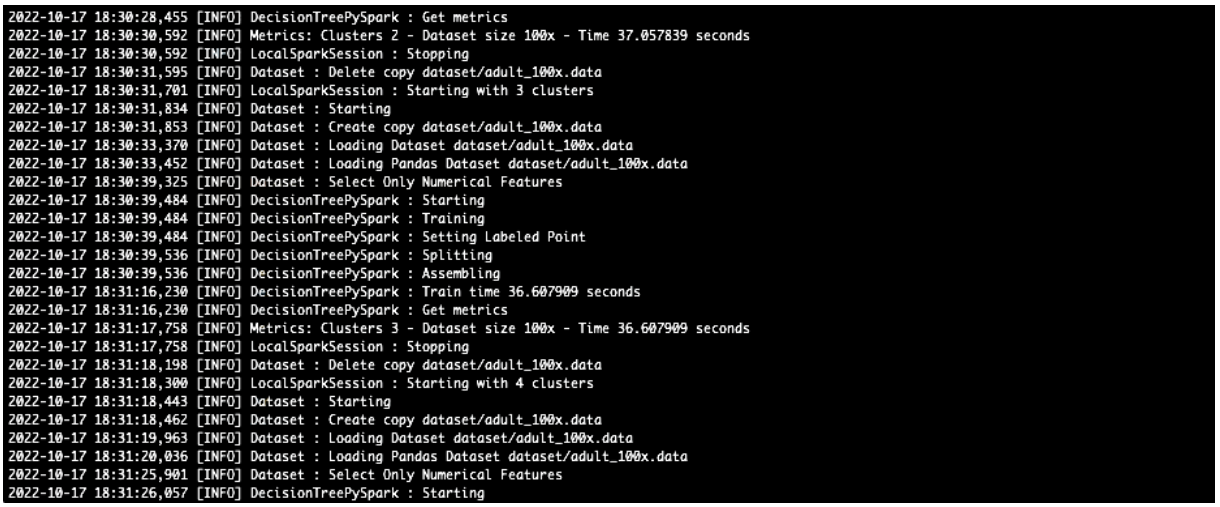
Le script est exécuté à l'aide de la commande suivante :

```
python3 necessite-de-la-distribution-de-l-apprentissage.py
```

Les données sont enregistrées dans le dossier "résultats" et peuvent être vérifiées et visualisées à l'aide du *notebook*:

```
necessite-de-la-distribution-de-l-apprentissage.ipynb
```

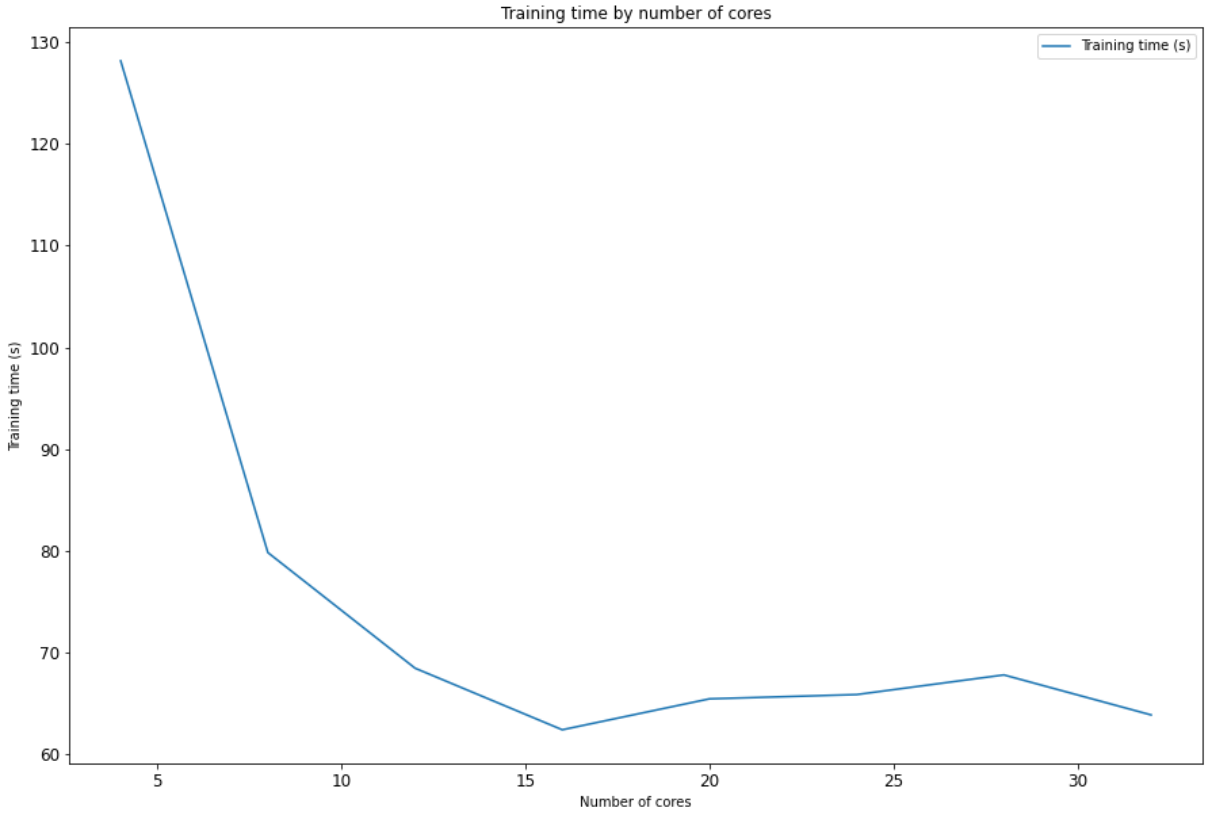
Pendant l'exécution des expériences, les données sont présentées à l'écran au moyen de logs et enregistrées dans des fichiers dans le dossier "log/" avec le nom de l'environnement.



4.3. Résultats

Dans le tableau ci-dessous nous pouvons observer que le script avec l'augmentation du nombre de cœurs réduit le temps d'exécution de l'algorithme, ceci peut être encore mieux vu dans le graphique ci-dessous. Mais ce temps semble se stabiliser pour une quantité supérieure à 12 cores, restant entre 60 et 70 secondes, dans ce cas apparemment il subit de petits changements dus à d'autres facteurs.

	Number of cores	Dataset Size (GB)	Training time (s)
0	4	36.06	128.16
1	8	36.06	79.85
2	12	36.06	68.49
3	16	36.06	62.44
4	20	36.06	65.48
5	24	36.06	65.90
6	28	36.06	67.83
7	32	36.06	63.90



4.4. Est-ce que la performance du modèle appris augmente du fait de l'entraînement sur un volume important de données ?

A travers les expériences ci-dessus, nous pouvons voir que l'augmentation du nombre de cœurs influence directement la réduction du temps d'exécution de l'algorithme. De cette manière, il devient indispensable d'exécuter le modèle distribué dans le cas de gros volumes de données.

5. Comparaison avec scikit-learn (python) et MLlib (pyspark)

Le temps de l'opération de réglage des paramètres a une relation directe avec les temps d'entraînement et de prédiction. En étudiant MLlib (pyspark), j'ai vu qu'il dispose de deux bibliothèques d'arbres de décision, l'une basée sur les dataframes (https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-tree-classifier) et l'autre sur les RDD (https://spark.apache.org/docs/latest/mllib-decision-tree.html). Les opérations de réglage des hyperparamètres de pyspark (https://spark.apache.org/docs/latest/ml-tuning.html) sont liées aux modèles d'apprentissage automatique basés sur le cadre de données distribué.

Puisqu'au début de notre article nous nous sommes concentrés sur l'utilisation d'arbres de décision basés sur des RDD, nous avons décidé d'effectuer une comparaison entre les temps d'entraînement et de prédiction de l'arbre de décision de pyspark basé sur des RDD et les temps d'entraînement et de prédiction de l'arbre de décision implémenté par la bibliothèque Sklearn.

5.1. Expérimentation

Dans cette expérience, nous avons exécuté l'algorithme d'apprentissage en changeant la taille du fichier, dans ce cas nous avons considéré le facteur 1 pour le fichier original et crée un facteur de multiplication qui variait entre 1x et 1000x la taille du fichier original, après quoi l'ensemble de données a été divisé en ensembles de données d'apprentissage (50%) et de test (50%), comme dans l'article. L'expérience a été réalisée avec 32 cores. Pour chaque ensemble de données, le temps d'entraînement et de prédiction a été calculé pour l'arbre de décision basé sur RDD de pyspark et l'arbre de décision implémenté par sklearn.

5.2. Exécution du script

Le script est exécuté à l'aide de la commande suivante :

```
python3 decisiontree_compare.py
```

Les données sont enregistrées dans le dossier "résultats" et peuvent être vérifiées et visualisées à l'aide du *notebook*:

```
decisiontree_compare.ipynb
```

Pendant l'exécution des expériences, les données sont présentées à l'écran au moyen de logs et enregistrées dans des fichiers dans le dossier "log/" avec le nom de l'environnement.

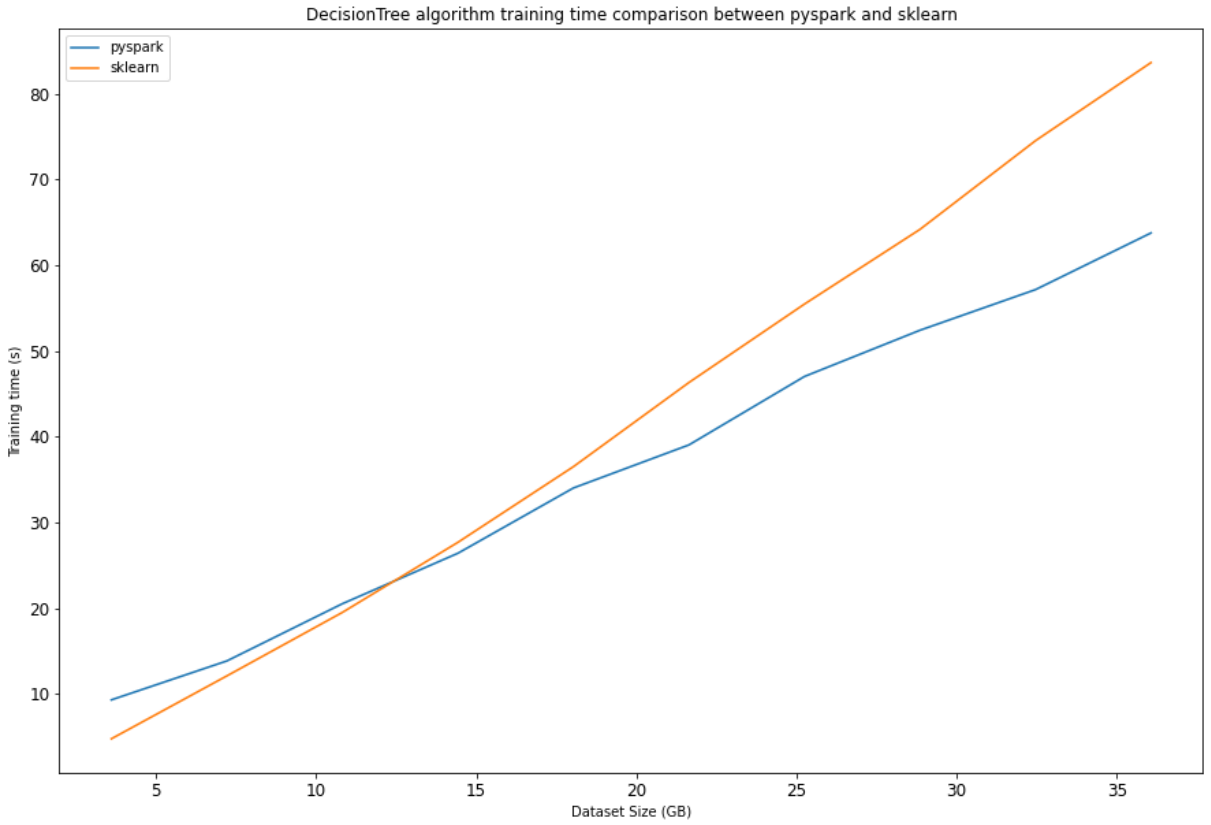
```
2022-10-17 18:35:16,420 [INFO] DecisionTreePySpark : Training
2022-10-17 18:35:16,421 [INFO] DecisionTreePySpark : Setting Labeled Point
2022-10-17 18:35:16,457 [INFO] DecisionTreePySpark : Splitting
2022-10-17 18:35:16,458 [INFO] DecisionTreePySpark : Assembling
2022-10-17 18:35:42,491 [INFO] DecisionTreePySpark : Train time 25.961877 seconds
2022-10-17 18:35:42,491 [INFO] DecisionTreePySpark : Predicting
2022-10-17 18:35:42,507 [INFO] DecisionTreePySpark : Predict time 0.015652 seconds
2022-10-17 18:35:42,508 [INFO] DecisionTreePySpark : Get metrics
2022-10-17 18:35:42,508 [INFO] DecisionTreeSklearn : Starting
2022-10-17 18:35:42,508 [INFO] DecisionTreeSklearn : Training
2022-10-17 18:35:42,508 [INFO] DecisionTreeSklearn : Setting X and y
2022-10-17 18:35:42,540 [INFO] DecisionTreeSklearn : Splitting
2022-10-17 18:35:50,747 [INFO] DecisionTreeSklearn : Train time 7.256327 seconds
2022-10-17 18:35:50,747 [INFO] DecisionTreeSklearn : Predicting
2022-10-17 18:35:50,839 [INFO] DecisionTreeSklearn : Predict time 0.091352 seconds
2022-10-17 18:35:50,839 [INFO] DecisionTreeSklearn : Getting metrics
2022-10-17 18:35:50,882 [INFO] Dataset : Delete copy dataset/adult_90x.data
2022-10-17 18:35:50,949 [INFO] Dataset : Starting
2022-10-17 18:35:50,950 [INFO] Dataset : Create copy dataset/adult_100x.data
2022-10-17 18:35:52,457 [INFO] Dataset : Loading Dataset dataset/adult_100x.data
2022-10-17 18:35:52,510 [INFO] Dataset : Loading Pandas Dataset dataset/adult_100x.data
2022-10-17 18:35:58,311 [INFO] Dataset : Select Only Numerical Features
2022-10-17 18:35:58,996 [INFO] DecisionTreePySpark : Starting
2022-10-17 18:35:58,997 [INFO] DecisionTreePySpark : Training
2022-10-17 18:35:58,997 [INFO] DecisionTreePySpark : Setting Labeled Point
2022-10-17 18:35:59,035 [INFO] DecisionTreePySpark : Splitting
2022-10-17 18:35:59,035 [INFO] DecisionTreePySpark : Assembling
```

5.3. Résultats

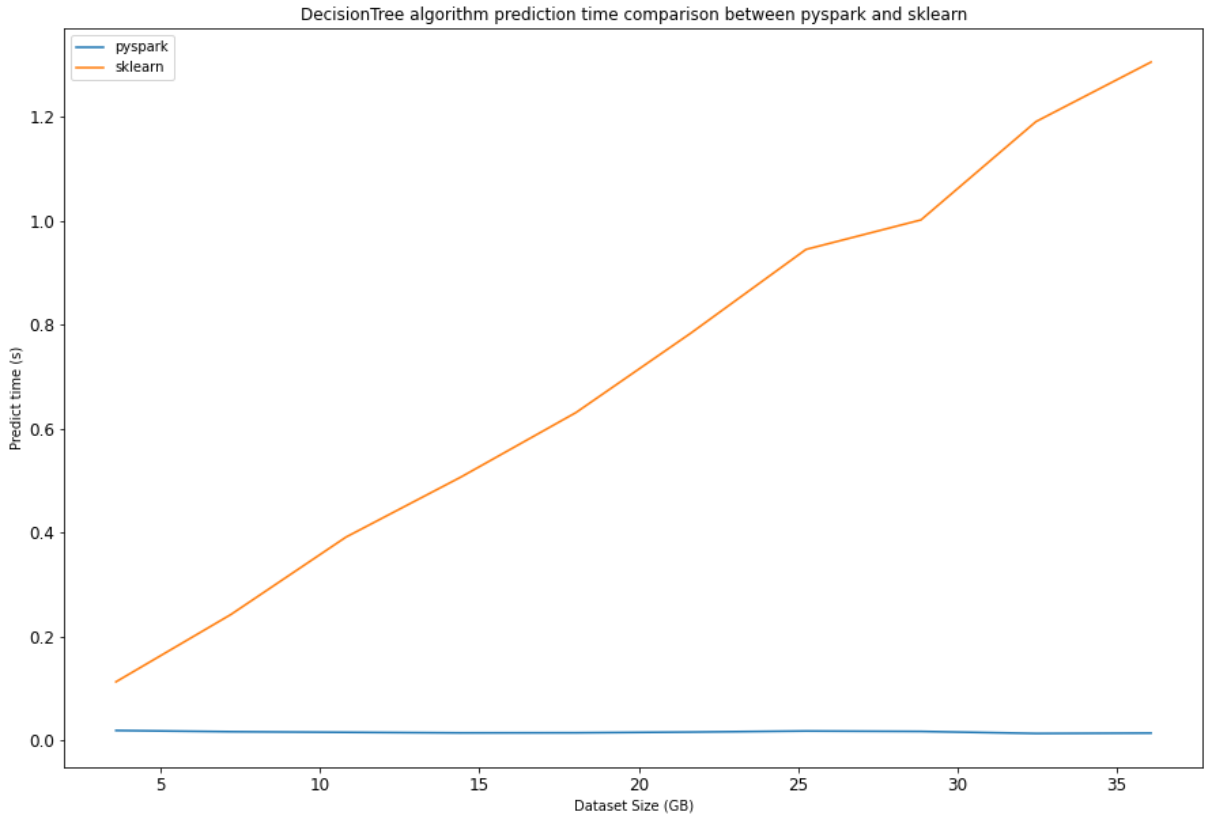
Comme nous pouvons l'observer dans les expériences précédentes, le temps d'entraînement est directement proportionnel à la taille de la base de données, mais croît à un rythme plus faible. Cela se produit dans les deux bibliothèques.

	Dataset Size (GB)	PySpark Total time (s)	PySpark Training time (s)	PySpark Predict time (s)	Sklearn Total time (s)	Sklearn Training time (s)	Sklearn Predict time (s)
0	3.6064	9.3077	9.2882	0.0195	4.8605	4.7474	0.1131
1	7.2129	13.8471	13.8299	0.0172	12.3286	12.0859	0.2427
2	10.8193	20.5202	20.5041	0.0160	19.8822	19.4906	0.3915
3	14.4257	26.4062	26.3913	0.0149	28.1720	27.6649	0.5071
4	18.0322	34.0198	34.0047	0.0151	37.1242	36.4931	0.6311
5	21.6386	39.0455	39.0290	0.0165	47.0924	46.3083	0.7841
6	25.2450	47.0292	47.0107	0.0185	56.4024	55.4575	0.9449
7	28.8514	52.4215	52.4038	0.0177	65.1727	64.1710	1.0017
8	32.4579	57.1572	57.1434	0.0138	75.7063	74.5154	1.1908
9	36.0643	63.7693	63.7549	0.0145	84.9322	83.6269	1.3053

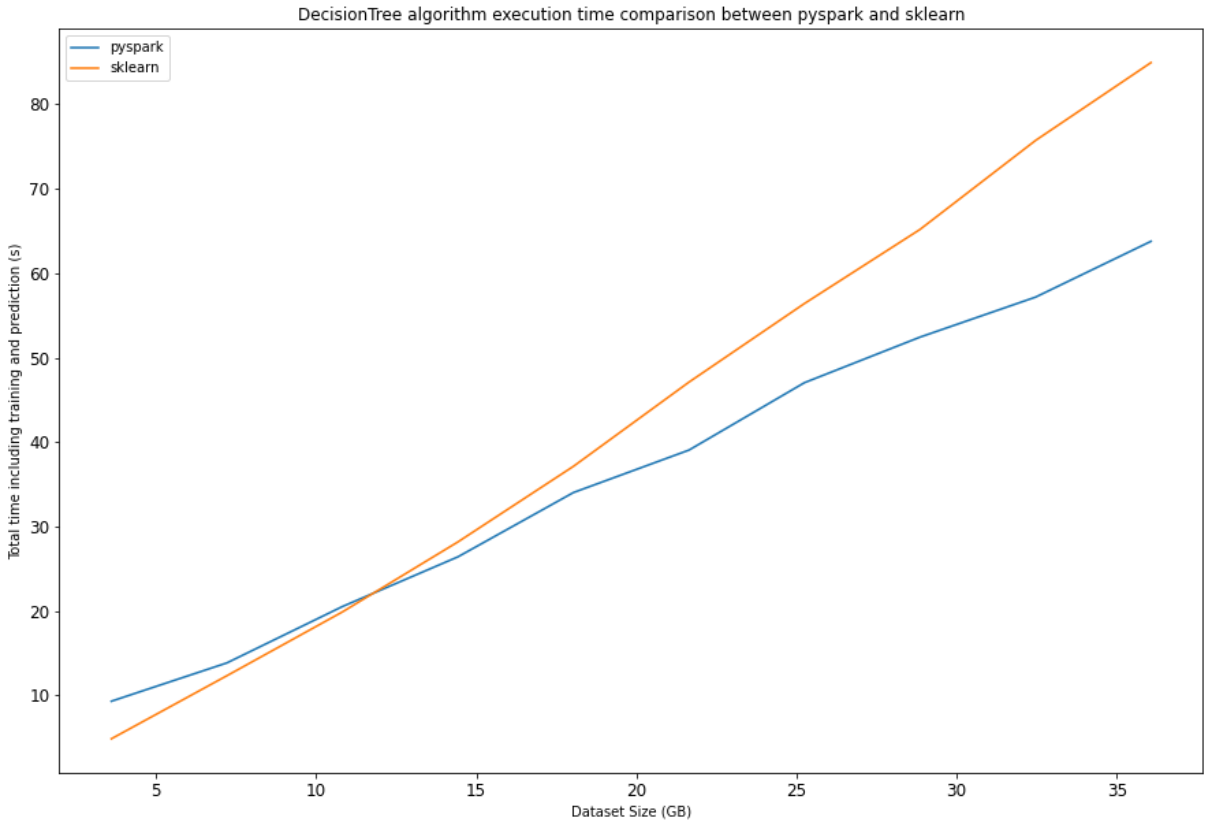
En effectuant l'expérience, nous avons pu vérifier que pour cet ensemble de données, l'implémentation de pyspark est plus rapide que l'implémentation de sklearn de l'arbre de décision pour les bases de données de plus de 12 Go environ.



Cependant, le temps de prédiction utilisant la bibliothèque pyspark est resté constant pendant la majeure partie de l'expérience, tandis que le temps de prédiction utilisant la bibliothèque sklearn, bien que faible par rapport au temps d'entraînement, a augmenté en fonction de l'augmentation de la taille de la base de données. Cela indique que Pyspark n'est pas seulement bon pour réduire le temps d'entraînement avec un grand volume de données, il est également rapide dans le processus de prédiction.

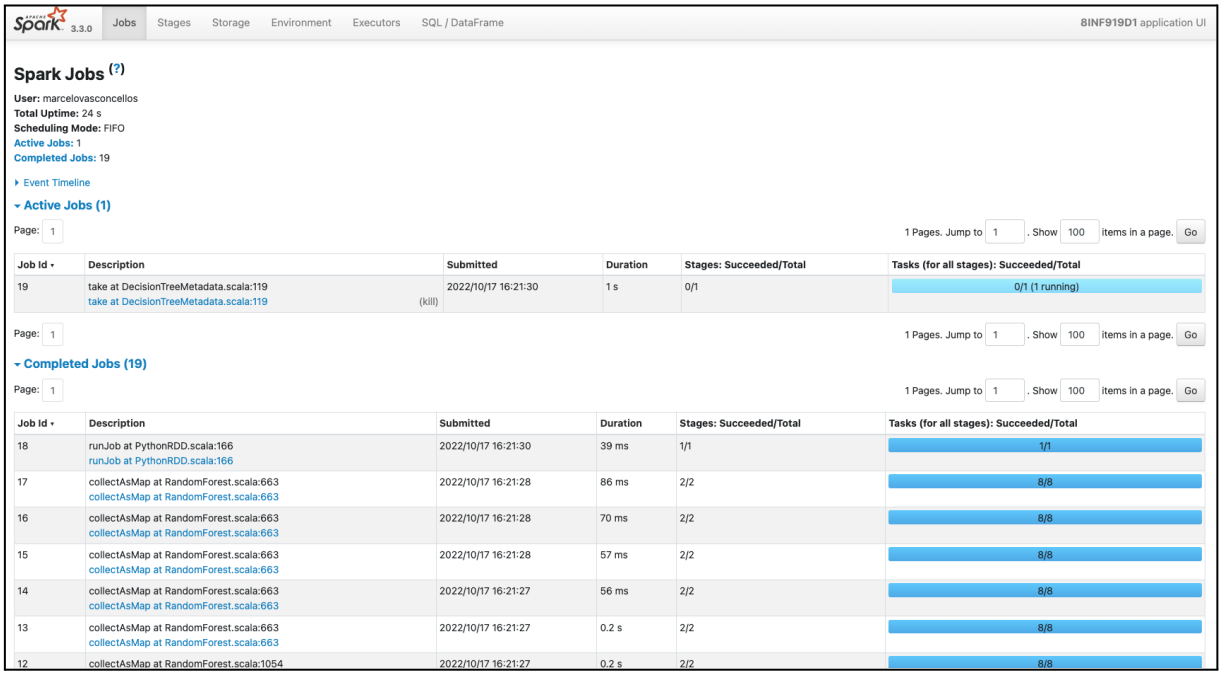


À la suite de cette expérience, nous pouvons constater l'importance d'utiliser des bibliothèques d'apprentissage automatique distribuées si vous devez travailler avec des volumes de données. Dans notre cas, l'apprentissage automatique est devenu plus efficace en utilisant des ensembles de données de plus de 12 Go environ. Dans notre expérience, nous n'avons pas évalué l'exactitude et la précision des prédictions.


DecisionTree algorithm execution time comparison between pyspark and sklearn
pyspark
sklearn
Total time including training and prediction (s)
Dataset Size (GB)
10
20
30
40
50
60
70
80
5
10
15
20
25
30
35

6. Spark-UI

Grâce à l'outil Spark-UI, il a été possible d'accéder à spark et de suivre l'évolution des travaux.



Spark 3.3.0 — Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame — 8INF919D1 application UI

Spark Jobs (?)

User: marcelovasconcellos
Total Uptime: 24 s
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 19

Event Timeline

Active Jobs (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

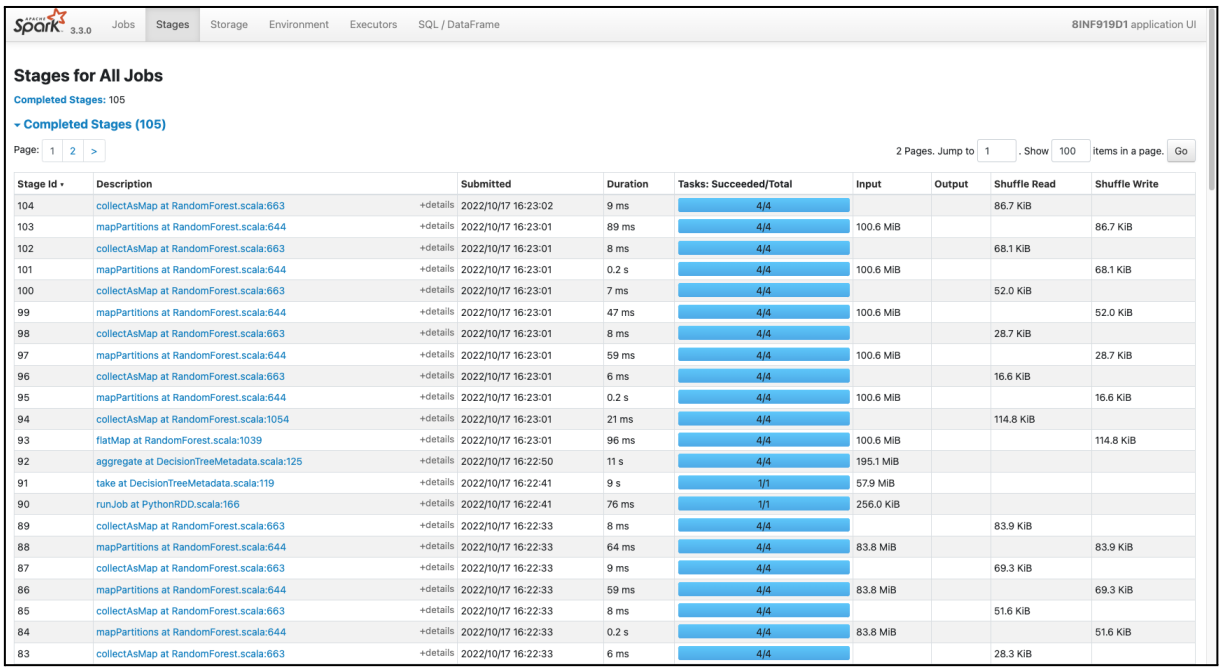
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
19	take at DecisionTreeMetadata.scala:119 take at DecisionTreeMetadata.scala:119 (kill)	2022/10/17 16:21:30	1 s	0/1	0/1 (1 running)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Completed Jobs (19)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
18	runJob at PythonRDD.scala:166 runJob at PythonRDD.scala:166	2022/10/17 16:21:30	39 ms	1/1	1/1
17	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2022/10/17 16:21:28	86 ms	2/2	8/8
16	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2022/10/17 16:21:28	70 ms	2/2	8/8
15	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2022/10/17 16:21:28	57 ms	2/2	8/8
14	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2022/10/17 16:21:27	56 ms	2/2	8/8
13	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2022/10/17 16:21:27	0.2 s	2/2	8/8
12	collectAsMap at RandomForest.scala:1054	2022/10/17 16:21:27	0.2 s	2/2	8/8



Spark 3.3.0 — Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame — 8INF919D1 application UI

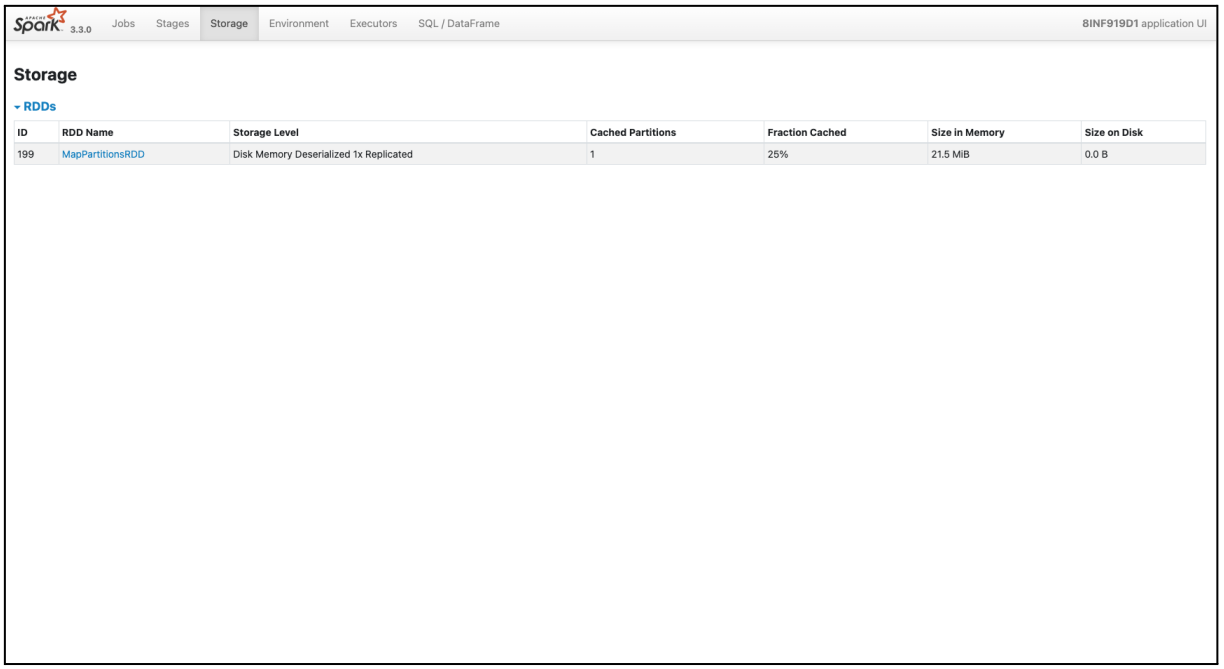
Stages for All Jobs

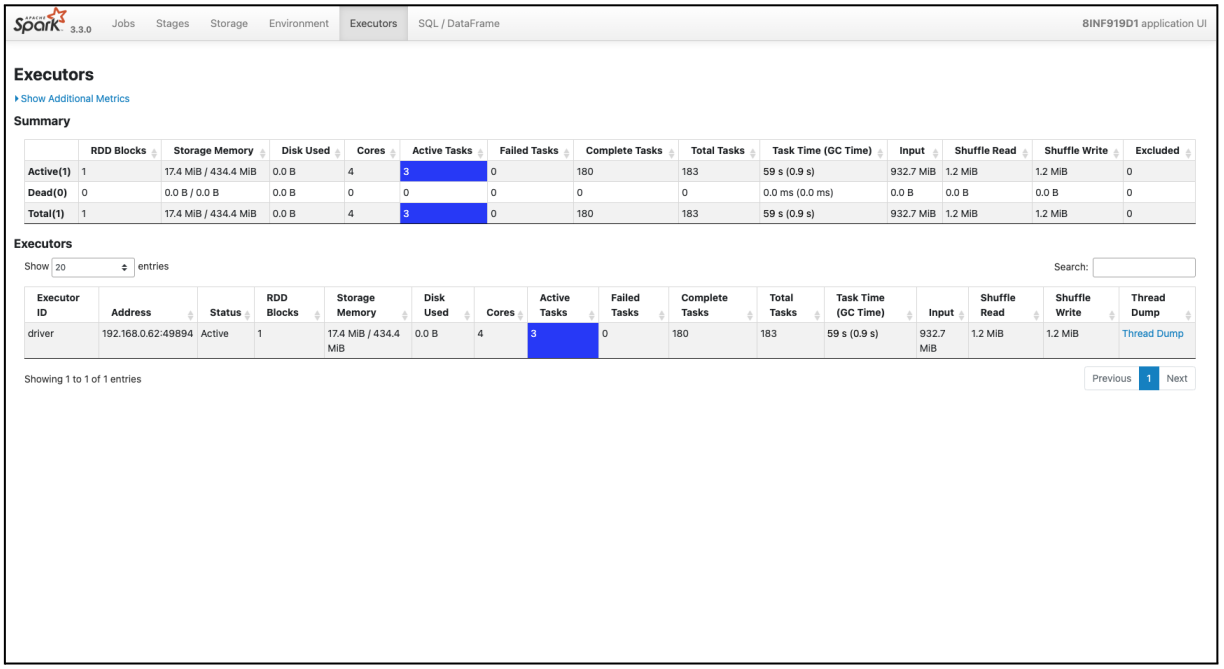
Completed Stages: 105

Completed Stages (105)

Page: 1 2 > — 2 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
104	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:23:02	9 ms	4/4			86.7 KiB	
103	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:23:01	89 ms	4/4	100.6 MiB			86.7 KiB
102	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:23:01	8 ms	4/4			68.1 KiB	
101	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:23:01	0.2 s	4/4	100.6 MiB			68.1 KiB
100	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:23:01	7 ms	4/4			52.0 KiB	
99	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:23:01	47 ms	4/4	100.6 MiB			52.0 KiB
98	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:23:01	8 ms	4/4			28.7 KiB	
97	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:23:01	59 ms	4/4	100.6 MiB			28.7 KiB
96	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:23:01	6 ms	4/4			16.6 KiB	
95	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:23:01	0.2 s	4/4	100.6 MiB			16.6 KiB
94	collectAsMap at RandomForest.scala:1054 +details	2022/10/17 16:23:01	21 ms	4/4			114.8 KiB	
93	flatMap at RandomForest.scala:1039 +details	2022/10/17 16:23:01	96 ms	4/4	100.6 MiB			114.8 KiB
92	aggregate at DecisionTreeMetadata.scala:125 +details	2022/10/17 16:22:50	11 s	4/4	195.1 MiB			
91	take at DecisionTreeMetadata.scala:119 +details	2022/10/17 16:22:41	9 s	1/1	57.9 MiB			
90	runJob at PythonRDD.scala:166 +details	2022/10/17 16:22:41	76 ms	1/1	256.0 KiB			
89	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:22:33	8 ms	4/4			83.9 KiB	
88	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:22:33	64 ms	4/4	83.8 MiB			83.9 KiB
87	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:22:33	9 ms	4/4			69.3 KiB	
86	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:22:33	59 ms	4/4	83.8 MiB			69.3 KiB
85	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:22:33	8 ms	4/4			51.6 KiB	
84	mapPartitions at RandomForest.scala:644 +details	2022/10/17 16:22:33	0.2 s	4/4	83.8 MiB			51.6 KiB
83	collectAsMap at RandomForest.scala:663 +details	2022/10/17 16:22:33	6 ms	4/4			28.3 KiB	


Spark 3.3.0
Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
8INF919D1 application UI
Storage
RDDs
ID
RDD Name
Storage Level
Cached Partitions
Fraction Cached
Size in Memory
Size on Disk
199
MapPartitionsRDD
Disk Memory Deserialized 1x Replicated
1
25%
21.5 MiB
0.0 B


Spark 3.3.0
Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
8INF919D1 application UI
Executors
Show Additional Metrics
Summary
RDD Blocks
Storage Memory
Disk Used
Cores
Active Tasks
Failed Tasks
Complete Tasks
Total Tasks
Task Time (GC Time)
Input
Shuffle Read
Shuffle Write
Excluded
Active(1) 1 17.4 MiB / 434.4 MiB 0.0 B 4 3 0 180 183 59 s (0.9 s) 932.7 MiB 1.2 MiB 1.2 MiB 0
Dead(0) 0 0.0 B / 0.0 B 0.0 B 0 0 0 0 0 0.0 ms (0.0 ms) 0.0 B 0.0 B 0.0 B 0
Total(1) 1 17.4 MiB / 434.4 MiB 0.0 B 4 3 0 180 183 59 s (0.9 s) 932.7 MiB 1.2 MiB 1.2 MiB 0
Executors
Show 20 entries
Search:
Executor ID
Address
Status
RDD Blocks
Storage Memory
Disk Used
Cores
Active Tasks
Failed Tasks
Complete Tasks
Total Tasks
Task Time (GC Time)
Input
Shuffle Read
Shuffle Write
Thread Dump
driver 192.168.0.62:49894 Active 1 17.4 MiB / 434.4 MiB 0.0 B 4 3 0 180 183 59 s (0.9 s) 932.7 MiB 1.2 MiB 1.2 MiB Thread Dump
Showing 1 to 1 of 1 entries
Previous
1
Next