Plan détaillé de préparation au projet big data :

Table des matières

I. Liens pour les recherches :

Usefull

https://spark.apache.org/docs/latest/sql-data-sources-csv.html

https://sparkbyexamples.com/spark/spark-read-options/n

https://spark.apache.org/docs/latest/api/scala/org/apache/spark/ContextAwareIterator.html?search=dataframe

https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html

Top

https://spark.apache.org/docs/latest/sql-getting-started.html

exemples de toutes les fonctions du dataframes , et jointures

https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html

Second

https://spark.apache.org/docs/latest/

Spark programming

https://spark.apache.org/docs/latest/sql-programming-guide.html

Complet

II. Ingestion et création du data lake

Le data lake est sur une architecture de stockage distribué (lecture et écriture) : **stocké sur hdfs , fixer les droits d'accès**

- Lister les sources

Source de données externe

Le datalake est alimente en continue (avec une architecture , orchestrateur , message queu ou équivalent) : **airflow automatique**

III. Nettoyage et validation des données

Les données nettoyées et validées sont stockées dans un output

Intermédiaire avec un format qui prend en charge le schéma

IV. Traitement des données

Construction d'application distribué : Spark- calcul distribué
L'application utilise aussi des optimisations de la distribution et de répartition de la charge

val dfPartitionned = df.repartition(200)

select avec cast :

df.select(expr(" potential >=50 ") , expr(" cast(overall as int) ") , expr("potential")).show(10)

EC Optimisations spark

Numpartitions

df= spark.read.option().csv()

val partitionedDF = df.repartition(200)

Create tamp view

df.createOrReplaceTempView(tableName)

créer le cache : df.cache()

libérer : spark.catalog.clearCache()

… autres optimisations du cours

Demarrer spark

Start-all.sh

imports nécessaires

```
import org.apache.spark.sql
import org.apache.spark.sql.types
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import org.apache.spark.sql._
import org.apache.spark.sql.expressions.Window
```

Join exemple :

```
val playersPotential = {
    players15.selectExpr(
        "sofifa_id", "short_name", "cast(overall as int) overall_15", "cast(potential as int) potential_15"
    ).join(players22.selectExpr(
        "sofifa_id", "short_name", "cast(overall as int) overall_22", "cast(potential as int) potential_22"
    ), "sofifa_id")
    .withColumn("evolution", expr("overall_22-overall_15"))
    .withColumn("potential_vs_actual", col("overall_22")-col("potential_15"))
}
```

Creation de colonne

```
Val foo = foo.withColumn( ‘status’ ,
Expr( ‘ CASE WHEN delay <=10  then ‘on-time’  else ‘on-delays’ ’ )
```

)

Développer des requêtes SQL et NoSQL pour traiter des données volumineuses

Utilisation de SQL

L'application utilise une user-defined function ou une expression en SQL

L'application utilise le DDL pour définir un schéma

// Enregistrez le DataFrame en tant que vue SQL Temporaire

Df.read()

Df.cache()

Val repartDf = Df.répartition(8 *Nbre_de_coeur*)

df.createOrReplaceTempView("nom_table")

val sqlDF = spark.sql("SELECT * FROM nom_table ")

sqlDF.show()

spark.sql("SELECT short_name , overall FROM player22 order by overall desc ").show

Donner les droits au fichier orchestrateur.sh

Etant a la racine du projet on fait : **chmod 777 -R ./**

avec Built in function

val sqldf= {spark.sql("""SELECT short_name , overall, nationality_name, rank

| from (select short_name , overall,nationality_name , RANK()

```
    | OVER( PARTITION BY nationality_name  Order BY overall DESC ) as
rank
    | from players22
    | )
    | where rank>=3
    |  ORDER BY overall desc
    | """
    | )}
val sqldf=  {spark.sql("SELECT short_name , overall, nationality_name,  rank
From (SELECT short_name , overall, nationality_name,  RANK()
OVER( PARTITION BY nationality_name Order By overall DESC) as rank
FROM player22
)
where rank  >=3
 " ) }
```

V. Stockage sur hdfs

Ecrire sur HDFS : https://sparkbyexamples.com/spark/spark-read-write-files-from-hdfs-txt-csv-avro-parquet-json/

Créer un dossier datalake en ligne de commande sur hdfs
Hdfs dfs -mkdir -p /user/ubuntu/datalake

Lui accorder tout les droits
Hdfs dfs -chmod -R 777 /user/ubuntu
Capturer les ls , et mettre dans le rapport : hdfs /user/ubuntu/

- Ubuntu/Datalake (tous les dataset)
- Cleandata
- Dataresult
    - Dataresult1
    - Dataresult2 …
    -

VI. Orchestration

Airflow ou commandes sh

Le pipeline contient plusieurs étapes et l'enchainement est automatisé via orchestrateur

- Faire plusieurs jobs, pour valider

Exemple d'orchestration avec sh

#!/bin/bash

Créer un répertoire dans HDFS

hdfs dfs -mkdir -p Nom_repertoire

Télécharger le fichier 'dracula'/Api

wget http://www.textfiles.com/etext/FICTION/dracula

donner les droits d'ecriture a HDFS sur ubuntu

su hadoop

start-all.sh

hdfs dfs chmod 777 -R /user/ubuntu

Mettre le fichier 'dracula' dans le répertoire 'Nom_repertoire 'de HDFS (Stocker sur HDFS)

hdfs dfs -put -f getdata/storage user/ubuntu/datalake

Lister les fichiers dans le répertoire ‘Nom_repertoire‘ de HDFS

hdfs dfs -ls Nom_repertoire

Récupérer le fichier 'center_earth' du répertoire 'livres' de HDFS

hdfs dfs -get livres/center_earth

Dans un fichier script.sh, puis le rendre exécutable avec la commande chmod +x run.sh. Ensuite, vous pouvez l’exécuter avec ./run.sh.

Créer un fichier sh pour chaque job, et appeler dans l’orchestrateur principale

Démarrer l’orchestrateur de façon automatisé

crontab -e

0 5 * * 1 sh /home/ubuntu/Desktop/exercice/orchestrateur.sh

Pour exécuter le fichier orchestrateur.sh on fait

./orchestrateur.sh

Crontab – l permet de vérifier si les modifications faites dans contrab ont été sauvegardé

VII. Résumé

1- avec python, récupérer les souces de données dynamique (api)

stocker en json par exemple , sur le disk

- lister les sources statiques

2- un job spark vas créer le datalake dans hdfs

(stocker toutes les sources de données recoltes dans un dossier dans Hdfs (le data lake) ,)

3- un job spark (validation des données) qui vas nettoyer et filtrer les données ,

et stocker dans un output intermédiaire (dossier dans hdfs)

4- des plusieurs jobs spark(environ 2 suivant l'épreuve)

de traitement des données du datalake (dans hdfs)

et stockage des résultat de chaque job dans un autre dossier hdfs

Optimiser (utiliser le cache, nombre de partitions parallèles pour la charge)

Savoir utiliser sparksql

exécuter les jobs en .sh ,ou relancer en automatique avec airflow

traiter les données (compétence obligatoire)

5- tout documenter dans le rapport word , a zipper

capture des résultats hdfs de chaque output ,

et tout le code dans le dossier word qui sera zipper

Consignes :

- Big Data
  - Durée : 6h
  - Travail demandé : créer un projet Spark, avec collecte transformation et présentation des données
  - Modalité de retour : un fichier Zip contenant le projet
  - Important :

 Vous travaillerez sur une machine virtuelle Linux/Ubuntu, avec Virtual Box

 Vous aurez à rendre une archive Zip : entrainez-vous à créer un Zip sur cette machine virtuelle

 Attention, veillez à bien tout prendre dans le Zip, seul son contenu sera évalué, les morceaux manquants ne seront pas récupérés.

Faire des captures

- Rapport
- built in