Projet Big Data

Olivier NEVE
Miguel CARDEROT
Hence MULUNDA

Introduction

Ce projet vise à offrir une expérience pratique dans la conception et la mise en œuvre d'un pipeline de données robuste en utilisant des technologies comme : PySpark, Hadoop Distributed File System (HDFS) et MongoDB Atlas.

L'objectif principal de ce projet est d'extraire les ensembles de données sélectionnés d'IMDb, de les traiter et de les nettoyer à l'aide de PySpark, puis de charger les données raffinées à la fois dans HDFS et dans une base de données MongoDB hébergée sur Atlas. À la fin de ce projet, nous visons à démontrer notre capacité à gérer efficacement des solutions de traitement et de stockage de données à grande échelle, en assurant la qualité, la cohérence et l'évolutivité des données.

Objectifs du Projet

- **Exploration et compréhension des jeux de données** : Analyser les ensembles de données IMDb pour identifier la structure, le schéma et les éventuels problèmes de qualité des données.
- **Configuration de HDFS** : Configurer HDFS sur une machine locale ou un cluster pour stocker les ensembles de données IMDb.
- **Pipeline de données PySpark** : Développer un programme PySpark pour l'extraction, la transformation, le nettoyage et le chargement des données.
- **Chargement des données dans HDFS et MongoDB** : Assurer le stockage et la récupération efficaces des données traitées.
- **Qualité et validation des données** : Mettre en œuvre des contrôles pour maintenir l'intégrité et la cohérence des données.

Structure et schéma des datasets

Les datasets non-commerciaux IMDb sont disponibles sous forme de fichiers TSV (tab-separated values) compressés en gz. Chaque fichier contient une ligne d'en-tête décrivant les colonnes. Les valeurs manquantes sont notées `\N`.

Etape de nettoyage de données

Extraction : Télécharger et décompresser l'ensemble de données à l'aide du script Python fourni.

Transformation : Nettoyer et prétraiter les données à l'aide de PySpark. Les étapes incluent la gestion des valeurs manquantes, la suppression des doublons et la correction des incohérences.

Chargement : Charger les données nettoyées dans HDFS et MongoDB pour le stockage et une analyse ultérieure.

Processus ETL

Le processus ETL a été mis en œuvre à l'aide des scripts suivants :

```python
import requests
import gzip
import shutil

class IMDBDatasetDownloader:
    def __init__(self, url, gz_file_name, tsv_file_name):
        self.url = url
        self.gz_file_name = gz_file_name
        self.tsv_file_name = tsv_file_name

    def download_file(self):
        response = requests.get(self.url, stream=True)
        with open(self.gz_file_name, 'wb') as file:
            file.write(response.content)
        print(f"Downloaded {self.gz_file_name}")

    def decompress_file(self):
        with gzip.open(self.gz_file_name, 'rb') as f_in:
            with open(self.tsv_file_name, 'wb') as f_out:
                shutil.copyfileobj(f_in, f_out)
        print(f"Decompressed {self.gz_file_name} to {self.tsv_file_name}")

    def download_and_decompress(self):
        self.download_file()
        self.decompress_file()
```

pipeline.py : Gère les opérations ETL à l'aide de PySpark.

hdfs_manager.py : Gère les opérations HDFS telles que la création de répertoires et la validation du stockage des données.

main.py : Le script principal pour orchestrer le pipeline ETL.

Procédures de Chargement des Données

Configuration de HDFS :

- Configuré HDFS en utilisant core-site.xml et hdfs-site.xml

```xml
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://hadoop-master:9000</value>
  </property>
</configuration>
```

```xml
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///usr/local/hadoop/data/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///usr/local/hadoop/data/datanode</value>
  </property>
</configuration>
```

Chargement dans MongoDB : Pour charger les données nettoyées dans MongoDB, nous utilisons PySpark pour établir une connexion à MongoDB et écrire les données transformées dans une collection MongoDB. Le processus inclut la lecture des données, leur nettoyage et transformation, puis l'utilisation du connecteur MongoDB pour Spark afin de les enregistrer dans la base de données MongoDB.

```python
    def transform_data(self, df):
        df = df.withColumn("birthYear", col("birthYear").cast("int"))

        df = df.withColumn("numKnownForTitles", col("knownForTitles").isNotNull().cast("int"))

        return df

    def save_to_hdfs(self, df):
        df.write.mode("overwrite").parquet(self.hdfs_path)
        print(f"Data saved to HDFS at {self.hdfs_path}")

    def save_to_mongodb(self, df):
        output_uri = f"{self.mongo_uri}.{self.mongo_collection}"
        df.write.format("mongo").mode("overwrite").option("uri", output_uri).save()
        print("Data saved to MongoDB")

    def run(self):
        df = self.read_data()
        df = self.clean_data(df)
        df = self.transform_data(df)

        self.save_to_hdfs(df)
        self.save_to_mongodb(df)

```

Qualité et Validation des Données

Nous avons mis en œuvre des contrôles de qualité des données pour garantir l'intégrité et la cohérence des données stockées dans HDFS et MongoDB. Cela inclut la validation du schéma, la vérification des valeurs manquantes et la comparaison des données entre HDFS et MongoDB pour identifier les divergences. Les tes

Exemple :

```
30      def clean_data(self, df):
31          df = df.na.fill("Unknown", subset=["primaryName", "birthYear", "deathYear", "primaryProfession", "knownForTitles"])
32          df = df.withColumn("birthYear", when(col("birthYear") == "\\N", None).otherwise(col("birthYear").cast("int")))
33          df = df.withColumn("deathYear", when(col("deathYear") == "\\N", None).otherwise(col("deathYear").cast("int")))
34          return df
35
36      def transform_data(self, df):
37          df = df.withColumn("birthYear", col("birthYear").cast("int"))
38          df = df.withColumn("numKnownForTitles", col("knownForTitles").isNotNull().cast("int"))
39          return df
```

Les défis potentiels et problèmes de qualité des données

1. Données Manquantes

Les champs manquants (représentés par `\N`) peuvent nécessiter des stratégies de traitement spécifiques telles que l'imputation, l'exclusion de lignes ou l'utilisation de modèles robustes aux données manquantes. Les valeurs manquantes peuvent entraîner des biais si elles ne sont pas gérées correctement.

2. Cohérence des Données

La cohérence des identifiants (`tconst` et `nconst`) est cruciale pour des jointures précises entre les fichiers. Des identifiants incorrects ou incohérents peuvent conduire à des erreurs dans les analyses et à des résultats inexacts. Une vérification régulière de l'intégrité référentielle est essentielle.

3. Volume des Données

Les grands volumes de données peuvent poser des défis en termes de stockage et de traitement. Les opérations sur de grands ensembles de données peuvent nécessiter des solutions de traitement distribué comme Hadoop ou Spark, ainsi que des bases de données optimisées pour le Big Data (par exemple, NoSQL).

4. Évolution du Schéma

Les changements dans la structure des fichiers, tels que l'ajout de nouvelles colonnes ou la modification des valeurs existantes, peuvent nécessiter des mises à jour des pipelines de traitement des données. Une gestion proactive des versions et une adaptation dynamique des scripts sont nécessaires pour maintenir la compatibilité.

5. Encodage et Format

Assurer une gestion correcte de l'encodage UTF-8 est essentiel pour éviter les erreurs de lecture et de transformation des données, surtout lorsqu'on traite des caractères spéciaux.

De plus, la manipulation de fichiers TSV nécessite des outils capables de gérer correctement les séparateurs de tabulation et les caractères d'échappement.

Stratégies pour gérer les défis

1. **Pour les données manquantes** : Utiliser des techniques de nettoyage des données, telles que l'imputation statistique ou l'utilisation d'algorithmes capables de traiter les valeurs manquantes.
2. **Pour la cohérence des données** : Mettre en place des vérifications régulières de l'intégrité des clés et utiliser des outils de validation des données.
3. **Pour le volume des données** : Utiliser des architectures distribuées et des outils de traitement de Big Data pour gérer efficacement le stockage et le traitement.
4. **Pour l'évolution du schéma** : Mettre en place un suivi des versions et des tests d'intégration continue pour détecter les changements de schéma.
5. **Pour l'encodage et le format** : Utiliser des bibliothèques et des outils robustes qui prennent en charge l'encodage UTF-8 et le format TSV de manière fiable.

Les difficultés rencontrées

- Problème de segmentation de mémoire RAM empêchant l’utilisation de Docker
- Les tests ne sont pas entièrement terminés car ont a été pris de court à cause de quelques soucis avec l'infrastructure docker
- Intégré de multiples technologies peut être complexe et nécessite une configuration minutieuse (HDFS et MongoDB avec PySpark)