TP Spark Streaming

Dans ce tp, on va manipuler spark structure streaming

Manip 1 : Charger le script de génération de données
On va utiliser un script qui permet de générer des données.
Ces données vont être utilisées pour le streaming.

Format des données :

```
+---------------------+-------+----------+----+----+
|event_time           |id     |date      |val1|val2|
+---------------------+-------+----------+----+----+
|2019-02-01 07:14:46.73 |1248795|2018-02-04|57  |490 |
|2019-02-01 07:14:46.738|1248795|2018-02-05|85  |169 |
|2019-02-01 07:14:46.744|1248795|2018-02-06|7   |210 |
+---------------------+-------+----------+----+----+
```

1 - Récupérez le script « gen-rand-csv.sh «

2 - Placez-le dans la sandbox à travers ambari
Sinon, en scp :

```
scp -P 2222 @ root@localhost:/root/
```

3 - Connectez vous à la sandbox

```
ssh -p 2222 root@localhost
```

4 - Vérifier que le fichier est bien présent :

```
ls
```

5 - Créer les dossiers dans lequel les données vont être générées par le script shell

```
mkdir -p /root/cours-spark/
mkdir -p /root/cours-spark/dump/
mkdir -p /root/cours-spark/dump/gen-rand-csv/
mkdir -p /root/cours-spark/dump/gen-rand-csv/source1/
mkdir -p /root/cours-spark/dump/gen-rand-csv/source2/
mkdir -p /root/cours-spark/dump/gen-rand-csv/source3/
```

Manip 2 : Streamer les données

1 - Dans le terminal précédent, lancer le script shell qui génère des fichiers en tapant :

```
chmod +x /path/to/script # ligne à lancer la 1° fois seulement

/path/to/script
```

2 - Dans un nouveau terminal, lancer une autre connexion ssh

```
ssh -p 2222 root@localhost
```

3 - Lancer spark en mode console :

```
pyspark
```

4 - Afficher les données qui sont générées :

```
import pyspark.sql.functions as F
path = 'file:/root/cours-spark/dump/gen-rand-csv/'

# Mise sur écoute
s1 = spark\
.readStream\
.option("sep", ",")\
.schema("event_time TIMESTAMP, id INT, date DATE, val1 INT,
val2 INT")\
.csv(path+'source1/', header=True)

# Afficher le contenu des données streamées
query = s1\
.writeStream\
.format("console")\
.option("truncate","false")\
.start()
```

Observez le résultat.

Pour arrêter l'écriture, taper :

```
query.stop()
```

Important : Pensez à bien le faire à chaque manipulation des données !

Manip 2.5 :

1 - arrêter l'écriture des streams :

```
query.stop()
```

2 - arrêter le script shell (dans le terminal qui génère les fichers):

```
CTRL + C
```

3 - Nettoyer le dossier où les données ont été générées par le script :

```
find /root/cours-spark/dump/gen-rand-csv/ -type f -name
'*.csv' -delete
```

Manip 3 : Manipulation des données en streaming

Pour faire des opérations sur les données en streaming, il suffit de d'écrire les opérations à appliquer sur le s1 avant le writeStream

```
# On applique des opérations sur le s1 (comme sur un dataframe)
s2 = s1.select(…).filter(…)…

# On écrit le résultat des opérations :
query = s2.writeStream(…).start()

# plus synthétique :
```

```
s1.select(…).filter(…).writeStream(…).start()
```

Partie 1 : Manipulation de données

Q : Afficher le data frame en supprimant la colonne val2

```
query = s1\
.drop('val2')\
.writeStream\
.format("console")\
.option("truncate","false")\
.start()
```

Q : Afficher le data frame avec seulement la colonne val2

```
query = s1\
.select('val2')\
.writeStream\
.format("console")\
.option("truncate","false")\
.start()
```

Q : Afficher le data frame en filtrant les éléments val1 > 60

```
query = s1\
.filter(F.col('val1') > 60)\
.writeStream\
.format("console")\
.option("truncate","false")\
.start()
```

Q : Afficher le data frame avec la somme de val1 et val2 pour chaque id et date (addition en ligne)

```
query = s1\
.withColumn('sum_val', F.col('val1') + F.col('val2'))\
.writeStream\
.format("console")\
.option("truncate","false")\
.start()
```

Q : Calculer la racine carré de val1 :

```
query = s1\
.withColumn('sqrt_val1', F.sqrt(F.col('val1')))\
.writeStream\
.format("console")\
.option("truncate","false")\
.start()
```

Partie 2 : Fenêtre de calcul & aggregation

Q : Lancer deux terminaux, qui écrivent chacun un mode différent :

```
query = s1\
.withWatermark("event_time", "10 seconds")\
.groupBy(
F.window(F.col('event_time'), "10 seconds", "5 seconds")
)\
.count().writeStream\
.outputMode("append")\
```

```
.format("console")\
.option('truncate', False)\
.start()


query = s1\
.withWatermark("event_time", "10 seconds")\
.groupBy(
F.window(F.col('event_time'), "10 seconds", "5 seconds")
)\
.count().writeStream\
.outputMode("update")\
.format("console")\
.option('truncate', False)\
.start()
```

Observez en particulier les event_time dans chacun des terminaux (doublon).
Observez le fonctionnement des deux modes

Q : Calculer les valeurs maximales de val1 et val2 par fenêtre de temps de 10s, toutes les 5s

```
query = s1\
.withWatermark("event_time", "10 seconds")\
.groupBy(
F.window(F.col('event_time'), "10 seconds", "5 seconds")
)\
.agg(
F.max('val1').alias('max_val1'),
F.max('val2').alias('max_val2')
)\
.writeStream\
.outputMode("update")\
.format("console")\
.option('truncate', False)\
.start()
```

Qu'observez-vous ?

Q : Afficher la moyenne, l'écart-type de val1 et val2 par id, et par fenêtre de temps de 10s

```
query = s1\
.withWatermark("event_time", "10 seconds")\
.groupBy(
F.window(
      F.col('event_time'), "10 seconds", "5 seconds"),
      F.col('id')
)\
.agg(
F.mean('val1').alias('mean_val1'),
F.stddev('val1').alias('std_val1'),
F.mean('val2').alias('mean_val2'),
F.stddev('val2').alias('std_val2')
)\
.writeStream\
.outputMode("update")\
.format("console")\
.option('truncate', False)\
.start()
```

Q : Calculer le nombre d'élément par date et par fenêtre de temps de 10s

```
query = s1\
.withWatermark("event_time", "10 seconds")\
.groupBy(
F.window(
      F.col('event_time'), "10 seconds", "5 seconds"),
      F.col('date')
)\
.count()\
.writeStream\
.outputMode("update")\
.format("console")\
.option('truncate', False)\
.start()
```

Manip : Jointure avec des données statiques

1 - Envoyer le fichier static.csv sur la sandbox par Ambari
Sinon par scp :

```
scp -P 2222 chemin/local/vers/static.csv root@localhost:/root/
cours-spark/dump/
```

2 - Dans la session spark, charger le fichier static :

```
static = spark.read\
.option("delimiter", ",")\
.option("header","true")\
.schema("date DATE, busy BOOLEAN")\
.csv('file:/root/cours-spark/dump/static.csv')
```

3 - Visualisez le contenu
Sur Quelle colonne va-t-on faire la jointure avec les données streamées ?

4 - Effectuer la jointure entre les données en streaming et les données statiques, et afficher le contenu dans la console :

```
s1.join(static, "date")\
.writeStream\
.format("console")\
.option('truncate', False)\
.start()
```

Par défaut, il s'agit d'une jointure « inner ». Donc on retient seulement les éléments qui sont communs aux deux tableaux.

On pourrait préciser un autre type de jointure en 3° paramètres, par exemple :

```
s1.join(static, "date", "right_join")
```

Attention, toutes les jointures ne sont pas possibles :
https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#support-matrix-for-joins-in-streaming-queries

Manip 4 : Jointure de données en streaming
On a 2 sources de données qui donnent des infos sur le même id
On cherche à faire la jointure des informations qui proviennent de 2 sources

1 - Lancer 2 streaming :

```
s1 = spark\
.readStream\
.option("sep", ",")\
.schema("event_time TIMESTAMP, id INT, date DATE, val1 INT,
val2 INT")\
.csv(path+'source1/', header=True)

s3 = spark\
.readStream\
.option("sep", ",")\
.schema("event_time TIMESTAMP, id INT, date DATE, val3 INT,
val4 INT")\
.csv(path+'source3/', header=True)
```

2 - Effectuer la jointure :

```
s_join = s1.join(s3, ["id", "date"])
```

3 = Afficher le résultats dans la console :

```
query = s_join\
.writeStream\
.format("console")\
.option('truncate', False)\
.start()
```

Manip 5 : Sauvegarder les données dans HDFS

On va écrire en parquet dans le dossier « save_stream » d'HDFS (ne pas lancer) :

```
s1\
.writeStream\
.format("parquet")\
.option("path", "save_stream")\
```

On pourrait écrire en orc ou cdv aussi… (ne pas lancer)

```
.format("orc")
.format("csv")
```

Pour ce faire, il faut aussi indiquer un dossier de checkpointing.
Ca permet de sauvegarder les données en cas de plantage du receiver. (ne pas lancer)

```
.option("checkpointLocation", "checkpoint_stream") \
```

1 - Lancer le code suivant pour sauvegarder les données streamées :

```
query = s1\
.withWatermark("event_time", "10 seconds")\
.groupBy(
F.window(F.col('event_time'), "10 seconds", "5 seconds")
)\
.count().writeStream\
```

```
.outputMode("append")\
.format("parquet")\
.option("checkpointLocation", "checkpoint_stream") \
.option("path", "save_stream")\
.option('truncate', False)\
.start()
```

2 - Vérifier que la sauvegarde a bien été effectuée :

Dans un autre terminal de la sandbox, taper

```
hdfs dfs –ls save_stream/
```

Jeter un oeil dans le dossier de checkpoint :

```
hdfs dfs –ls checkpoint_stream/
```

Manip 6 : Connexion à Kafka

Partie 1 : Lancer un producer Kafka

1 - ouvrir un terminal, et se connecter en ssh à la sandbox :

```
scp -P 2222 @ root@localhost:/root/
```

2 - Créer un topic « console_topic » :

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --
zookeeper localhost:2181 --replication-factor 1 --partitions 1 --
topic console_topic
```

3 - Vérifier sa création :

```
 /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --list --
zookeeper localhost:2181
```

4 - Lancer un producer sur le topic :

```
/usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --
broker-list sandbox-hdp.hortonworks.com:6667 --topic console_topic
```

5 - Ecrivez n'importe quoi

6 - Vous pouvez vous amusez à vérifier le contenu avec un receiver consumer kafka

Partie 2 : Connecter Spark à Kafka

1 - ouvrir un terminal, et se connecter en ssh à la sandbox :

```
scp -P 2222 @ root@localhost:/root/
```

2 - Lancer le shell pyspark
Attention, cette fois, il faut « installer » le package qui fait la liaison spark streaming et kafka :

```
pyspark --packages org.apache.spark:spark-sql-
kafka-0-10_2.11:2.3.0
```

3 - Streamez le kafka :

```
df = spark \
.readStream \
```

```
.format("kafka") \
.option("kafka.bootstrap.servers", "sandbox-hdp.hortonworks.com:
6667") \
.option("subscribe", "console_topic") \
.load()
```

4 - Affichez les données streamées :

```
query = df\
.writeStream\
.format("console")\
.option('truncate', False)\
.start()
```

Qu'observez vous ?
Pourquoi ?

4 - Convertir les données bytes en string :

```
df = df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

5 - Affichez à nouveau les données streamées :

```
query = df\
.writeStream\
.format("console")\
.option('truncate', False)\
.start()
```

Manip 7 : Streamer les données sur l'API REST transillien

A vous de jouer !