Praktikum 3 Apache Spark – Data Frames und Structured Streaming

Ziel des Praktikums ist es, Datenströme zu generieren, einzulesen und mit einfachen statistischen Methoden zu verarbeiten.

Vorbereitung

Machen Sie sich mit den Inhalten des Foliensatzes **SparkStructuredStreaming** zu Kapitel 3 des aktuellen Semesters und den Pyspark-Klassen `pyspark.sql.streaming.DataStreamReader` und `pyspark.sql.streaming.DataStreamWriter` vertraut.

In *Aufgabe 2* wird für den Versand von Datenpaketen **Netcat** verwendet – Informationen zu Netcat finden Sie hier: https://linux.die.net/man/1/nc

Für *Aufgabe 3 und 4* wird ein **PageView-Simulator** von imaginären Seitenbesuchen der URLs von https://cookiedream.net verwendet – machen Sie sich dessen Funktionalität klar anhand des **Scala Sourcecodes** auf der letzten Seite dieses Aufgabenblatts.

Durchführung des Praktikums

Loggen Sie sich auf Zeppelin ein und kopieren Sie das Notebook praktikum3/_istaccount.

Aufgabe 1 – Word Count – data at rest *(Wiederholung …)*

… s. auch das Notebook Einführung zum Spark Big Data Cluster:
https://141.100.62.85:7070/#/notebook/2E8PGDY1N

Kopieren Sie zunächst den *Word Count*-Paragraphen in Ihr Notebook und machen Sie sich die Wirkungsweise klar – der hinterlegte *lorem ipsum*-Text befindet sich wie angegeben im Verzeichnis `hdfs://sunspear:9000/data/lorem/lorem.txt`.

Erstellen Sie nun eine **Pyspark-Variante in einem neuen *Word Count*-Paragraphen**, und nutzen Sie dabei die **sql-functions *explode* und *split*** aus dem Package *pyspark.sql.functions.* Arbeiten Sie mit DataFrames.

WICHTIG: Stoppen Sie im Folgenden Ihre laufende Streaming Query immer, bevor Sie eine neue starten oder ein Window schließen!

Aufgabe 2 – Word Count – streaming data (Netcat)

Nutzen Sie für die Implementierung des *Word Count* auf streaming data die sql-functions *explode* und *split* analog zu Aufgabe 1 – nur das Einlesen und die Ausgabe der Daten ändert sich nun im Streaming-Modus:

Als Datenquelle nutzen Sie einen TCP Socket und lassen die Ergebnisse an Ihrer Console als Datensenke ausgeben:

a) Bereiten Sie in Ihrem Notebook den entsprechenden Python-Code vor. Verwenden Sie DataStreamReader und -Writer in Analogie zu dem Scala-Beispiel aus dem Foliensatz **SparkStructuredStreaming, Folie 12**. Ihr Pyspark-Paragraph sollte mit dem Start Ihrer query enden.

Bearbeitungsdatum: 08.05. (Do5y), 15.05. (Mi6y) und 22.05.2019 (Mi6x)

b) Bevor Sie den Pyspark-Paragraphen laufen lassen, verbinden Sie sich mit Putty oder ssh auf einem Nodes und starten Sie dort **netcat** im Listening Mode (= server) auf Port 22XX. Denken Sie daran als Datenquelle die entsprechende IP-Adresse des Hostes anzugeben:

```
nc -lk 22XX
```

c) Führen Sie nun Ihren vorbereiteten Pyspark-Paragraphen aus, der nach dem Start der query zeilenweise die Texte empfängt, die über das netcat Terminal Window nun eingegeben werden:

Geben Sie in Ihrem netcat Terminal Window einen beliebigen Fließtext ein, dessen Eingabe Sie zeilenweise mit <return> beenden.

d) Zur **Anzeige der Ergebnisse** Starten Sie im nächsten SQL-Paragraphen (%SQL) eine Select-Anweisung und lassen Sie diese als Tabelle visualisieren (Achtung es kommt hierbei zu leichten Verzögerungen):

```
Select * from <queryName>
```

Beobachten Sie die Ausgabe (ein refresh der Ausgabe erreichen Sie jeweils durch wiederholten Aufruf der Select-Anweisung), und fahren Sie mit Ihren Eingaben fort:
Sie sollten eine Tabelle mit den von Ihnen eingegebenen Worten und deren Häufigkeit sehen, absteigend sortiert nach der Häufigkeit, die sich jeweils aktualisiert, sobald Sie mit <return> die nächste Fließtexteingabe abschließen.

Sie können auch einen beliebigen Text in das Terminal-Window hineinkopieren, um entsprechende Wortfrequenzen zu ermitteln.

Wenn Sie Ihre Query modifizieren wollen, stoppen Sie sie zunächst, modifizieren Sie den Code und fügen ihn erneut ein. Netcat bleibt aufgrund des Parameters k geöffnet und wartet auf die nächste Query.

e) **Beenden** Sie Ihre **Streaming Query** bei Bedarf wie folgt:

```
query.stop()
```

f) Explizites **Beenden von netcat: <Ctrl+C>**

Ausarbeitung

Dokumentieren Sie die Wirkungsweise der Parameter `"complete"` für den `outputMode` und `"memory"` für das `format`.

Aufgabe 3 – PageViews · Seitenaufrufe *cookiedream.net*

a) **Ausgabe der Anzahl von Seitenaufrufen per URL**
cookiedream.net möchte, dass Sie ein Monitoring von deren Website durchführen. Sie haben Zugriff auf einen Event Stream mit folgendem Event-Schema, in dem `"ts"` den Timestamp des Events darstellt:

```
PageView(ts:String, url:String, status:Int, country:String, userID:Int)
```

Auf den Event Stream kann via TCP **Port 3333** auf dem **Host *starfall.fbi.h-da.de*** zugegriffen werden. Zu Testzwecken können Sie sich mit dem Stream verbinden, indem Sie netcat im Client Mode nutzen:

```
nc starfall.fbi.h-da.de 3333
```

Hinweis: Der Stream wird zufällig generiert, sodass alle Gruppen einen unterschiedlichen Stream haben werden. Sie können den zur Verfügung gestellten PageViewGenerator Code auch selbst in einer eigenen Spark-Scala-Shell ausführen (s. Seite 4 unten und Seite 5).

Verbinden Sie sich – in Analogie zu Aufgabe 2 – mit dem Stream und geben Sie die Anzahl der Aufrufe per URL an:

- Verwenden Sie wieder die API der `pyspark.sql.functions` `explode` und `split`.
- Splitten Sie zunächst den Datenstrom zeilenbasiert, d.h. Delimiter hierfür ist `"\n"`.
- Generieren Sie dann spaltenweise entsprechende Splits pro Zeile für jedes Attribut des Event-Schemas – mindestens jedoch für das Attribut `url` – und verwenden als Delimiter hierfür `'\t'`.

Starten Sie Ihre Query …. und **stoppen Sie die Query wieder, bevor Sie weitermachen mit Teilaufgabe b)!**

Zur Anzeige der `counts` pro `url` starten Sie wieder eine entsprechende Select-Anweisung auf Ihrer Query, die ein refresh des Ergebnisses jeweils in einer Tabelle anzeigt.

b) **Ausgabe der Anzahl von Seitenaufrufen per URL pro Minute und Sliding Windows mit einer Frequenz (Schrittweite) von 30 Sekunden**
In Ergänzung zum Zählen der Seitenaufrufe der einzelnen URLs möchte das Unternehmen die Anzahl der Seitenaufrufe in kleineren Zeitfenstern beobachten, um die Zeiträume mit der Hauptlast zu erkennen. Modifizieren Sie den Code unter a) durch Einfügen von 1-minütigen Sliding Windows mit einer Frequenz (Schrittweite) von 30 Sekunden. Sortieren Sie anschließend die Anzahl der Seitenaufrufe nach Zeitfenstern:
*Hinweis: Zur **Definition eines Sliding Window** übergeben Sie als erstes Argument an die groupBy-Methode* `window(<time_column>, <window_size>, <sliding_interval>)`

- Visualisieren Sie die Ergebnisse zunächst in einer Tabelle.
- Welche der grafischen Visualisierungsmöglichkeiten sind für diesen Datenstrom sinnvoll? Testen Sie auch unterschiedliche `Settings` bei den Visualisierungen.

Ausarbeitung
Dokumentieren Sie die sinnvollen Visualisierungsmöglichkeiten und erläutern Sie jeweils die Wirkungsweise von möglichen Settings in Ihrer Ausarbeitung.

Aufgabe 4 – PageViews · Seitenaufrufe *cookiedream.net*

a) **Visualisierung - getrennt nach Ländercodes**

Erweitern Sie Ihre Query-Deklaration aus Aufgabe 3 so, dass insbesondere auch das Attribut mit dem Ländercode in die Query mit einbezogen werden kann.

- Erstellen Sie Visualisierungen, die die Anzahl der Aufrufe pro Ländercode im zeitlichen Verlauf anzeigen.

b) **Visualisierung – Statuscode, nur Seitenaufrufe aus Deutschland**

Beziehen Sie nun in Ihre Query unterschiedliche Status-Codes mit ein und schränken Sie die Selektion auf die Aufrufe aus Deutschland ein.

- Visualisieren Sie die Häufigkeit unterschiedlicher Status-Codes der auf deutsche Aufrufe eingeschränkten Selektion.

Ausarbeitung
Dokumentieren Sie in Ihrer Ausarbeitung die Wirkungsweise der unterschiedlichen Ausgabemodi `Append`, `Update` und `Complete` im Zusammenhang mit Aggregationen auf der Event-Time in unserem Beispiel-Stream.

Erläutern Sie, wie sich diese jeweils im Zusammenhang mit *Late Data Handling* verhalten würden.

Orientieren Sie sich am Foliensatz *SparkStructuredStreaming* und den darin angegeben Links auf die aktuellsten Spark-Dokumentationen.

Testat: Protokollieren Sie spätestens 1 Woche nach dem Praktikumstermin alle Anweisungen aus den Aufgaben 1-4 in Ihrem Notebook und geben Sie dieses der Gruppe „Dozent“ frei. Laden Sie weiterhin ein Dokument (pdf) in Moodle hoch, das die genannten AusarbeitungsTasks für die einzelnen Aufgaben enthält.

Auf der folgenden Seite finden Sie den **PageViewGenerator Code**, den Sie optional selbst in einer eigenen Scala-Spark-Shell laufen lassen können (Achtung – dann eigene Porteinstellung verwenden). Eine Scala-Spark-Shell erhalten Sie auf allen Nodes durch die Eingabe des Alias `spark-shell`.

PageViewGenerator Code*
https://cookiedream.net

```
import java.io.PrintWriter
import java.net.ServerSocket
import java.util.Random

case class PageView(val ts: String, val url: String, val status: Int, val country: String, val
userID: Int) extends Serializable {
  override def toString(): String = {
    "%s\t%s\t%s\t%s\t%s\n".format(ts, url, status, country, userID)
  }
  def this(in: String) {
    this(in.split("\t")(0), in.split("\t")(1), in.split("\t")(2).toInt, in.split("\t")(3),
in.split("\t")(4).toInt)
  }
}

val pages = Map("https://cookiedream.net/" -> .7,
                "https://cookiedream.net/order" -> 0.2,
                "https://cookiedream.net/about" -> .1)
val httpStatus = Map(200 -> .95,
                     404 -> .05)
val userCountry = Map("Germany" -> .5,
                      "Austria" -> .5)
val userID = Map((1 to 100).map(_ -> .01): _*)

def pickFromDistribution[T](inputMap: Map[T, Double]): T = {
    val rand = new Random().nextDouble()
    var total = 0.0
    for ((item, prob) <- inputMap) {
        total = total + prob
        if (total > rand) {
            return item
        }
    }
    inputMap.take(1).head._1 // Shouldn't get here if probabilities add up to 1.0
}

def getNextClickEvent(): String = {
    val ts = java.time.LocalDateTime.now.toString
    val id = pickFromDistribution(userID)
    val page = pickFromDistribution(pages)
    val status = pickFromDistribution(httpStatus)
    val country = pickFromDistribution(userCountry)
    new PageView(ts, page, status, country, id).toString()
}

val port = xxxx // bitte xxxx durch einen Port > 1000 und != 3333 ersetzen!
val viewsPerSecond = 10.0
val sleepDelayMs = (1000.0 / viewsPerSecond).toInt
val listener = new ServerSocket(port)
println("Listening on port: " + port)

while (true) {
  val socket = listener.accept()
  new Thread() {
    override def run(): Unit = {
      println("Got client connected from: " + socket.getInetAddress)
      val out = new PrintWriter(socket.getOutputStream(), true)

      while (true) {
        Thread.sleep(sleepDelayMs)
        out.write(getNextClickEvent())
        out.flush()
      }
      socket.close()
    }
  }.start() // Don't execute in Zeppelin! Will block interaction!
}
```

* Based on
https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/streaming/clickstream/PageViewGenerator.scala