ApacheSparkThroughEmail3

November 27, 2018

1 Apache Spark Through Email 3 - Explode, Shuffle Partitions, UDF, Parquet partition

```
In [1]: val records = spark.read.parquet("/datasets/enron/enron-small.parquet")
        records.cache

        //8 cached partitions
        records.count
```

```
Waiting for a Spark session to start...
```

```
records = [uuid: string, from: string ... 8 more fields]
```

```
Out[1]: 191926
```

1.1 What user has the most emails in his/her mailbox?

```
In [2]: import org.apache.spark.sql.functions._

        //explode of map creates columns "key" and "value"
        val userMailFields = records.select(explode($"mailFields")).
            where($"key" === "UserName").withColumnRenamed("value","userName").select("userNam
```

```
userMailFields = [userName: string]
```

```
Out[2]: [userName: string]
```

```
In [3]: //sanity check
        userMailFields.count
```

```
Out[3]: 191926
```

```
In [4]: val defaultShufflePartitions = spark.conf.get("spark.sql.shuffle.partitions")
```

```
defaultShufflePartitions = 200
```

```
Out[4]: 200
```

```
In [5]: //groupBy causes shuffle - default 200 partitions - ~ 2 seconds to execute
        println(s"Number of partitions before shuffle: ${userMailFields.rdd.getNumPartitions}")

        val allUserCounts = userMailFields.groupBy("userName").count

        println(s"Number of partitions after shuffle: ${allUserCounts.rdd.getNumPartitions}")

        val top10UserCounts = allUserCounts.orderBy(desc("count")).limit(10)

        println(s"Number of partitions after limit: ${top10UserCounts.rdd.getNumPartitions}")

        top10UserCounts.show
```

```
Number of partitions before shuffle: 8
Number of partitions after shuffle: 200
Number of partitions after limit: 1
+----------+-----+
|  userName|count|
+----------+-----+
|kaminski-v|28465|
|dasovich-j|28234|
|    kean-s|25351|
| germany-c|12436|
|    beck-s|11830|
|campbell-l| 6490|
|  guzman-m| 6054|
|     lay-k| 5937|
|haedicke-m| 5246|
|  arnold-j| 4898|
+----------+-----+
```

```
allUserCounts = [userName: string, count: bigint]
top10UserCounts = [userName: string, count: bigint]
```

```
Out[5]: [userName: string, count: bigint]
```

1.1.1 Adjusting default partitions

```
In [6]: spark.conf.set("spark.sql.shuffle.partitions", 8)
```

```
        val allUserCountsSmall = userMailFields.groupBy("userName").count

        println(s"Number of partitions after shuffle: ${allUserCountsSmall.rdd.getNumPartitions

        val top10UserCountsSmall = allUserCountsSmall.orderBy(desc("count")).limit(10)

        //executes in about 0.3s
        top10UserCountsSmall.show
```

```
Number of partitions after shuffle: 8
+----------+-----+
|  userName|count|
+----------+-----+
|kaminski-v|28465|
|dasovich-j|28234|
|    kean-s|25351|
| germany-c|12436|
|    beck-s|11830|
|campbell-l| 6490|
|  guzman-m| 6054|
|     lay-k| 5937|
|haedicke-m| 5246|
|  arnold-j| 4898|
+----------+-----+
```

```
allUserCountsSmall = [userName: string, count: bigint]
top10UserCountsSmall = [userName: string, count: bigint]
```

```
Out[6]: [userName: string, count: bigint]
```

1.2 Writing partitioned data with UDF

Data access patterns: By using directory structures for our typical queries we can "index" data for faster access. Let's say that for our analysis we only want "enron.com" senders and typically access the data by "from" field (and we have a *lot* of data).

```
In [7]: import org.apache.spark.sql.functions._

        def isEnronEmail(str: String): Boolean = str.endsWith("@enron.com")

        val isEnronEmailUdf = udf(isEnronEmail(_:String):Boolean)

        println(records.count)

        val enrons = records.where(isEnronEmailUdf($"from"))
```

```
        println(enrons.count)

        enrons.write.partitionBy("from").parquet("/datasets/enron/from-part")
```

```
191926
154101
```

```
isEnronEmailUdf = UserDefinedFunction(<function1>,BooleanType,Some(List(StringType)))
enrons = [uuid: string, from: string ... 8 more fields]
```

```
isEnronEmail: (str: String)Boolean
```

```
Out[7]: [uuid: string, from: string ... 8 more fields]
```

```
In [8]: val kaminskis = spark.read.parquet("/datasets/enron/from-part").where("from = 'vince.ka

        kaminskis.count
```

```
kaminskis = [uuid: string, to: array<string> ... 8 more fields]
```

```
Out[8]: 14340
```