ApacheSparkThroughEmail3

July 10, 2024

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

[1]:
```
val homeDir = sys.props("user.home")
val records = spark.read.parquet("/datasets/enron/enron-small.parquet")
records.cache

//8 cached partitions
records.count
```

```
homeDir = /home/medale
records = [uuid: string, from: string … 8 more fields]
```

[1]:
```
191926
```

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

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

// |-- mailFields: map (nullable = true)
//|    |-- key: string
//|    |-- value: string (valueContainsNull = true)
records.printSchema

/*
root
 |-- key: string (nullable = false)
 |-- value: string (nullable = true)
 */
records.select(explode(records("mailFields"))).printSchema

records.select(explode(records("mailFields"))).show(10)

//explode of map creates columns "key" and "value"
val userMailFields = records.select(explode($"mailFields")).
```

```
    where($"key" === "UserName").withColumnRenamed("value","userName").
↪select("userName")
```

```
root
 |-- uuid: string (nullable = true)
 |-- from: string (nullable = true)
 |-- to: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- cc: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- bcc: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- dateUtcEpoch: long (nullable = true)
 |-- subject: string (nullable = true)
 |-- mailFields: map (nullable = true)
 |    |-- key: string
 |    |-- value: string (valueContainsNull = true)
 |-- body: string (nullable = true)
 |-- attachments: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- fileName: string (nullable = true)
 |    |    |-- size: integer (nullable = true)
 |    |    |-- mimeType: string (nullable = true)
 |    |    |-- data: binary (nullable = true)

root
 |-- key: string (nullable = false)
 |-- value: string (nullable = true)

+--------------------+--------------------+
|                 key|               value|
+--------------------+--------------------+
|            UserName|             blair-l|
|               X-bcc|                    |
|        Mime-Version|                 1.0|
|          X-FileName|lblair (Non-Privi…|
|              X-From|Anderson, Gary E…|
|Content-Transfer-…|               7bit|
|          Message-ID|<3452380.10758593…|
|            FileName|                 7.|
|        Content-Type|text/plain; chars…|
|                X-To|Blair, Lynn </O=E…|
+--------------------+--------------------+
only showing top 10 rows

userMailFields = [userName: string]
```

[2]: [userName: string]

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

[3]: 191926

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

```
defaultShufflePartitions = 200
```

[4]: 200

[5]:
```
// disable adaptive query engine (AQE since 3.0.0)
spark.conf.set("spark.sql.adaptive.enabled","false")  // default true

//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: 20
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]
```

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

1.1.1 Adjusting default partitions

[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

// now - leave default AQE enabled
spark.conf.set("spark.sql.adaptive.enabled","true")  // default true
```

```
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]
```

[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).

[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(s"$homeDir/datasets/enron/from-part")
```

```
191926
154101
isEnronEmailUdf = SparkUserDefinedFunction($Lambda$6752/
  ↪0x00007ca0fd649b98@6f1dfce3,BooleanType,List(Some(class[value[0]:␣
  ↪string])),Some(class[value[0]: boolean]),None,false,true)
enrons = [uuid: string, from: string … 8 more fields]


isEnronEmail: (str: String)Boolean
```

[7]: [uuid: string, from: string … 8 more fields]

[8]:
```
val kaminskis = spark.read.parquet("/datasets/enron/from-part").where("from =␣
  ↪'vince.kaminski@enron.com'")

kaminskis.count
```

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

[8]: 14340

[ ]: