"Azure - Databricks - Cheat Sheet"

12 3 2022

Introduction

Apache Spark is a unified analytics engine for large-scale data processing and machine learning.

The Three V's of Big Data: Volume, Velocity, and Variety.

Understanding the architecture of spark job

- Spark is distributed computing environment.
- The unit of distribution is a Spark Cluster.
- Every Cluster has a Driver and one or more executors.
- Work submitted to the Cluster is split into as many independent Jobs as needed - this is how work is distributed across the Cluster's nodes.
- Jobs are further subdivided into tasks.
- The input to a job is partitioned into one or more partitions.

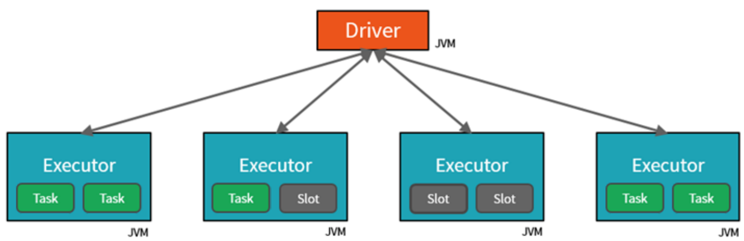

Figure 1: Spark - Cluster - Tasks

Jobs & stages

- Each parallelized action is referred to as a **Job**.
- The results of each Job (parallelized/distributed action) is returned to the Driver.
- Depending on the work required, multiple Jobs will be required.
- Each Job is broken down into **Stages**.

Reading Data

```
fileName = "dbfs:/mnt/training/wikipedia/clickstream/2015_02_clickstream.tsv"
csvSchema = StructType([
  StructField("prev_id", IntegerType(), False),
  StructField("curr_id", IntegerType(), False),
  StructField("n", IntegerType(), False),
  StructField("prev_title", StringType(), False),
  StructField("curr_title", StringType(), False),
  StructField("type", StringType(), False)
])

testDF = (spark.read          #The DataFrameReader
  .option('header', 'true')   #Ignore line #1 - it's a header
  .option('sep', "\t")        #Use tab delimiter (default is comma-separator)
  .schema(csvSchema)          #Use the specified schema
  .csv(fileName)              #Creates a DataFrame from CSV after reading in the file
)

#Display data
testDF.printSchema()
```

DataFrames vs SQL & Temporary Views

```
dataDF.createOrReplaceTempView("name_of_db")

#Use simple sql

%sql
SELECT * FROM pagecounts
```

Convert from SQL back to a `DataFrame`

```
tableDF = spark.sql("SELECT DISTINCT project FROM pagecounts ORDER BY project")
display(tableDF)
```

Exercise

```
(source, sasEntity, sasToken) = getAzureDataSource()
spark.conf.set(sasEntity, sasToken)

path = source + "/wikipedia/pagecounts/staging_parquet_en_only_clean/"

# 1. Define data frame
#
df = (spark                    # Our SparkSession & Entry Point
  .read                        # Our DataFrameReader
  .parquet(path)               # Read in the parquet files
  .select("article")           # Reduce the columns to just the one
  .distinct()                  # Produce a unique set of values
```

```
)
totalArticles = df.count() # Identify the total number of records remaining.

print("Distinct Articles: {0:,}".format(totalArticles))
```

Describe the difference between eager and lazy execution

Fundamental to Apache Spark are the notions that

- Transformations are LAZY - like creating data - They eventually return another `DataFrame`.
- Actions are EAGER - display data (*touch data*)

Transformations applied to `DataFrames` are lazy, meaning they will not trigger any jobs. If you pass the `DataFrame` to a display function, a job will be triggered because display is an action.

Types of Transformations

- A transformation may be wide or narrow.
- A wide transformation requires sharing data across workers.
- A narrow transformation can be applied per partition/worker with no need to share or shuffle data to other workers.

Narrow Transformations

The data required to compute the records in a single partition reside in at most one partition of the parent Dataframe.

```
from pyspark.sql.functions import col
display(countsDF.filter(col("NAME").like("%TX%")))
```

Wide Transformations

The data required to compute the records in a single partition may reside in many partitions of the parent Dataframe. These operations require that data is shuffled between executors. Wide transformation shares data across workers by shuffling data between executors.

```
from pyspark.sql.functions import col
display(countsDF.groupBy("UNIT").sum("counts"))
```

Catalyst Optimizer

Among the most powerful components of Spark are Spark SQL. At its core lies the Catalyst optimizer. This extensible query optimizer supports both rule-based and cost-based optimization. Spark SQL uses Catalyst's general tree transformation framework in four phases - Analysis, Logical Optimization, Physical Planning, and Code Generation. Our code is evaluated and optimized by the Catalyst Optimizer.

UnsafeRow (also known as Tungsten Binary Format)

- Sharing data from one worker to another can be a costly operation.
- Spark has optimized this operation by using a format called Tungsten.
- Tungsten prevents the need for expensive serialization and de-serialization of objects in order to get data from one JVM to another.