Apache Spark 3.0 Preview2 Notable Features

Spark 3.0.0-preview2 download
Released on 23rd Dec 2019

Observable Metrics for Spark Streaming

Observable metrics are named aggregate function that can be defined on a query. As soon as the execution of a Dataframe reaches a completion point (e.g. finishes batch query or reaches streaming epoch) a named event is emitted that contains the metrics for the data processed since the last completion point.

A user can observe these metrics by attaching a listener to spark session, it depends on the execution mode which listener to attach:

- Batch: QueryExecutionListener. This will be called when the query completes. A user can access the metrics by using the QueryExecution.observedMetrics map.
- (Micro-batch) Streaming: StreamingQueryListener. This will be called when the streaming query completes an epoch. A user can access the metrics by using the StreamingQueryProgress.observedMetrics map. Please note that we currently do not support continuous execution streaming.

```
spark.range(100)
.observe(
  name = "my_event",
  min($"id").as("min_val"),
  max($"id").as("max_val"),
  sum($"id").as("sum_val"),
  count(when($"id" % 2 === 0, 1)).as("num_even"))
.observe(
  name = "other_event",
  avg($"id").cast("int").as("avg_val"))
```

Spark Plugin Interface for Driver

Spark now provides an extension API to implement Driver Plugins. This makes Monitoring much easier and smoother. Interface define `DriverPlugin`.

API for Using Existing Data Partition while CoGroup

Currently if users want to do cogroup on DataFrames, there is no good way to do except for KeyValueGroupedDataset.

1. KeyValueGroupedDataset ignores existing data partition if any. That is a problem.
2. groupByKey calls typed function to create additional keys. You can not reuse existing columns, if you just need grouping by them

```
val df3 = df1.groupByRelationKey("a", "b")
  .cogroup(df2.groupByRelationKey("a", "b")) { case (key, data1, data2) =>
    data1.zip(data2).map { p =>
      p._1.getInt(2) + p._2.getInt(2)
    }
}
```

Support Column position in DataSource V2

Now DS v2 Api supports add/alter column with Column Position. It's absolutely an important feature where schema consistency is required.

Spark with Hadoop 3.0

Spark now fully supports Hadoop 3.0

Avro Schema Evolution

Now `from_avro` and `AvroDataToCatalyst` allow schema evolution.
Doc: *Converts a binary column of Avro format into its corresponding catalyst value. If a schema is provided via the option actualSchema, a different (but compatible) schema can be used for*
reading. If no actualSchema option is provided, the specified schema must match the read data,
otherwise the behavior is undefined: it may fail or return arbitrary result.

Avro Schema Evolution.

```
import scala.collection.JavaConverters._
df1.select(from_avro(data = 'eventBytes, jsonFormatSchema = schema2, options = Map("actualSchema" ->
schema1).asJava) as "decoded")
```

Schema2 is evolved version of Schema1

Create Table Like Using Provider

```
CREATE TABLE tb1 LIKE tb2
```

It was already available in Hive and its extremely convenient feature for SparkSQL. Now Spark has adopted it and added a native feature.

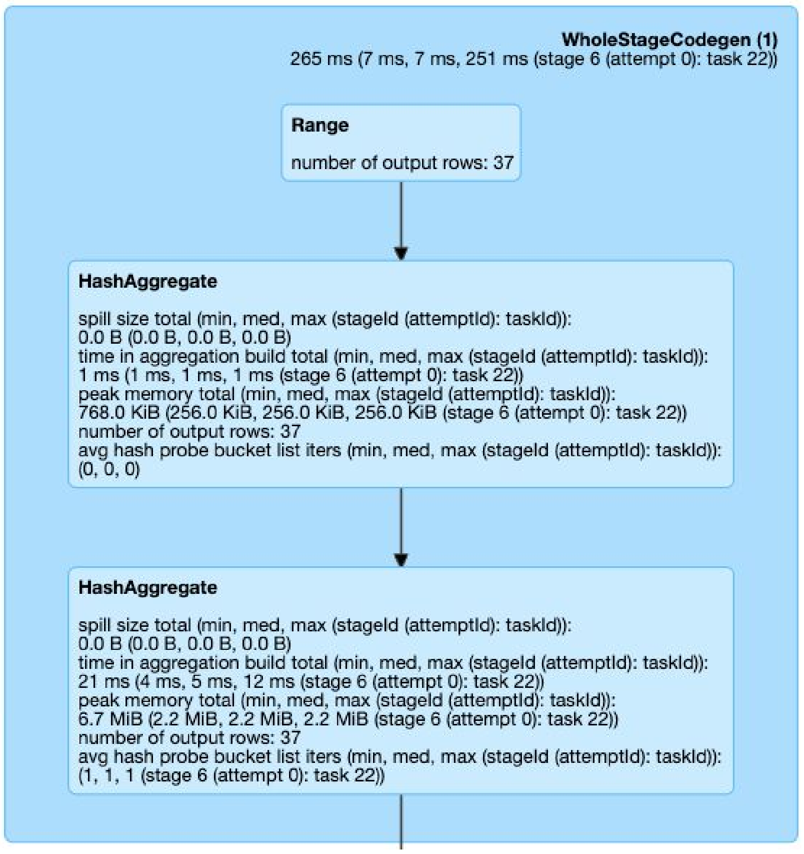
```
# 1
CREATE TABLE IF NOT EXISTS table1 LIKE table2 USING parquet
CREATE TABLE tbl1 LIKE another USING org.apache.spark.sql.hive.orc

# 2
CREATE GLOBAL TEMP VIEW src AS SELECT 1 AS a, '2' AS b
CREATE TABLE tbl LIKE globaldb.src USING parquet
```

[Internal] New Block Manager Heartbeat endpoint

BlockManagerMasterEndpoint handles many events from executors like RegisterBlockManager, GetLocations, RemoveShuffle, RemoveExecutor etc. In a heavy cluster/app, it is always busy. The BlockManagerHeartbeat event also was handled in this endpoint. These tasks might trigger timeout when its too busy.

New Display Metrics in Spark UI



```
WholeStageCodegen (1)
265 ms (7 ms, 7 ms, 251 ms (stage 6 (attempt 0): task 22))

Range
number of output rows: 37

HashAggregate
spill size total (min, med, max (stageId (attemptId): taskId)):
0.0 B (0.0 B, 0.0 B, 0.0 B)
time in aggregation build total (min, med, max (stageId (attemptId): taskId)):
1 ms (1 ms, 1 ms, 1 ms (stage 6 (attempt 0): task 22))
peak memory total (min, med, max (stageId (attemptId): taskId)):
768.0 KiB (256.0 KiB, 256.0 KiB, 256.0 KiB (stage 6 (attempt 0): task 22))
number of output rows: 37
avg hash probe bucket list iters (min, med, max (stageId (attemptId): taskId)):
(0, 0, 0)

HashAggregate
spill size total (min, med, max (stageId (attemptId): taskId)):
0.0 B (0.0 B, 0.0 B, 0.0 B)
time in aggregation build total (min, med, max (stageId (attemptId): taskId)):
21 ms (4 ms, 5 ms, 12 ms (stage 6 (attempt 0): task 22))
peak memory total (min, med, max (stageId (attemptId): taskId)):
6.7 MiB (2.2 MiB, 2.2 MiB, 2.2 MiB (stage 6 (attempt 0): task 22))
number of output rows: 37
avg hash probe bucket list iters (min, med, max (stageId (attemptId): taskId)):
(1, 1, 1 (stage 6 (attempt 0): task 22))
```

Spark UI is now equipped with Stageid, attempid & taskid for better debugging.For a given operator, it will be easy to identify the task which is taking maximum time to complete from the SQL tab itself

Clustering ML supports Weighting

KMeans now support weighting. This is inspired by scikit-learn and it helps to handle skewed or imbalance data, ensemble methods etc.

```
df = spark.createDataFrame(data, ["features", "weighCol"])
kmeans = KMeans(k=2)
kmeans.setWeightCol("weighCol")
```

Python3 Spark Docker Release Image

Now Spark Docker image supports Python3.
`dev/make-distribution.sh` and `python/setup.py` uses python3

Advanced modes in Spark Dataset Explain

Spark Dataset Explain mode now supports multiple formats -

- Simple
- Extended
- Codegen

```
Found 1 WholeStageCodegen subtrees.
== Subtree 1 / 1 (maxMethodCodeSize:282; maxConstantPoolSize:175(0.27% used);
numInnerClasses:0) ==
*(1) Range (0, 100, step=1, splits=8)
Generated code:
/* 001 */ public Object generate(Object[] references) {
/* 002 */   return new GeneratedIteratorForCodegenStage1(references);
/* 003 */ }
/* 004 */
/* 005 */ // codegenStageId=1
/* 006 */ final class GeneratedIteratorForCodegenStage1 extends
org.apache.spark.sql.execution.BufferedRowIterator {
/* 007 */   private Object[] references;
```

- Cost

```
== Optimized Logical Plan ==
Range (0, 100, step=1, splits=Some(8)), Statistics(sizeInBytes=800.0 B)
```

- Formatted

```
== Physical Plan ==
* Range (1)


(1) Range [codegen id : 1]
Output: [id#0L]
```

PySpark Explain now supports Extended and Mode as above

PySpark is now supporting scala similar mode and extended params.

```
df.explain(mode='formatted')
df.explain(extended=True)
```

Unify toString methods ML Models

Now all Spark ML Models toString methods expose basic information.

Stream-Stream Join Inconsistent Output Fix

Stream-Stream joins using Left outer Join gives inconsistent output.

PushedFilters to metadata in Parquet DSv2

Spark 3.0 filter now supports Pushed down.

```
== Parsed Logical Plan ==
RelationV2[id#83L] parquet file:/Users/chouabh/Downloads/pqt

== Optimized Logical Plan ==
RelationV2[id#83L] parquet file:/Users/chouabh/Downloads/pqt

== Physical Plan ==
*(1) ColumnarToRow
+- BatchScan[id#83L] ParquetScan Location:
```

```
InMemoryFileIndex[file:/Users/chouabh/Downloads/pqt], ReadSchema:
struct<id:bigint>, PushedFilters: []
```

SaveAsTable supports append Mode

`saveAsTable` now supports Append.

```
df.write.mode("append").saveAsTable(tbl)
```

PySpark UDF now supports more than 256 Arguments

Pyspark UDF now supports more than 256 arguments.
Mlflow has a requirement for this.

```
from pyspark.sql.types import *
from pyspark.sql.functions import udf

data = [["test-%d" % i for i in range(N)]] * 5
df = self.spark.createDataFrame(data)

def test_f(*a):
    return "success"

test_udf = udf(test_f, StringType())
df.select(test_udf(*df.columns))
```

Spark download with user-provided Hadoop Fails with Kubernetes

Spark-submit with user-provided hadoop fails on Kubernetes because of missing hadoop libraries on Spark Classpath.

The resolution was to hack the spark entry-point but now its fixed. User provided Hadoop can include SPARK_DIST_CLASSPATH in classpath to include the required jars.

Other Notable features

- Default enabling of Nested Schema Pruning and Nested Pruning
- Task Summary table now contains only successful tasks' metrics
- Single branch CaseWhen transformed to optimized IfElse internally
- Spark Executor binAddress is now Configurable
- Speculative Execution is available with one task as well.
- dataframe.na.drop will remove all rows if no column is defined, by resolving ambiguity.
- Maven upgraded to 3.6.3
- Lz4-java upgraded to 1.7.0
- K8s client upgraded to 4.6.4
- Jersey upgraded to 2.29.1
- Bug Fix of pyspark setJobGroup not matching with Java threads
- toPandas get correct dtypes with empty DF
- Bug Fix Query on External Json Partitioned table