Understanding the explain plan and spark UI

by: Karl Arao

*(best viewed using View -> Web Layout)*

# Generating explain plan

root@704a9920dffd:/spark/bin# ./spark-shell

## create times5 dataframe

scala> val simpleNumbers = spark.range(1,1000)

**simpleNumbers**: **org.apache.spark.sql.Dataset[Long]** = [id: bigint]

scala> val times5 = simpleNumbers.selectExpr("id \* 5 as id")

**times5**: **org.apache.spark.sql.DataFrame** = [id: bigint]

scala> times5.explain()

== Physical Plan ==

\*(1) Project [(id#0L \* 5) AS id#2L]

+- \*(1) Range (1, 1000, step=1, splits=4)

## create moreNumbers dataframe

scala> val moreNumbers = spark.range(1,5000,2)

**moreNumbers**: **org.apache.spark.sql.Dataset[Long]** = [id: bigint]

## join the data frames

scala> val joined = times5.join(moreNumbers, "id")

**joined**: **org.apache.spark.sql.DataFrame** = [id: bigint]

## generate explain plan – here the join is not yet executed. It needs an action like show()

scala> joined.explain()

== Physical Plan ==

\*(2) Project [id#2L]

+- \*(2) BroadcastHashJoin [id#2L], [id#6L], Inner, BuildLeft

:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false])), [id=#56]

: +- \*(1) Project [(id#0L \* 5) AS id#2L]

: +- \*(1) Range (1, 1000, step=1, splits=4)

+- \*(2) Range (1, 5000, step=2, splits=4)

scala> joined.selectExpr("sum(id)").explain()

== Physical Plan ==

\*(3) HashAggregate(keys=[], functions=[sum(id#2L)])

+- Exchange SinglePartition, true, [id=#101]

+- \*(2) HashAggregate(keys=[], functions=[partial\_sum(id#2L)])

+- \*(2) Project [id#2L]

+- \*(2) BroadcastHashJoin [id#2L], [id#6L], Inner, BuildLeft

:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false])), [id=#95]

: +- \*(1) Project [(id#0L \* 5) AS id#2L]

: +- \*(1) Range (1, 1000, step=1, splits=4)

+- \*(2) Range (1, 5000, step=2, splits=4)

## Get type of a variable

scala> sum\_two.getClass

**res12**: **Class[\_ <: org.apache.spark.sql.DataFrame]** = class org.apache.spark.sql.Dataset

scala> times5.getClass

**res13**: **Class[\_ <: org.apache.spark.sql.DataFrame]** = class org.apache.spark.sql.Dataset

scala> simpleNumbers.getClass

**res14**: **Class[\_ <: org.apache.spark.sql.Dataset[Long]]** = class org.apache.spark.sql.Dataset

scala> joined.getClass

**res15**: **Class[\_ <: org.apache.spark.sql.DataFrame]** = class org.apache.spark.sql.Dataset

## this is an RDD

scala> parallel1000.getClass

**res16**: **Class[Long]** = long

# Generate explain and execute

scala> val sum\_two = joined.selectExpr("sum(id)")

**sum\_two**: **org.apache.spark.sql.DataFrame** = [sum(id): bigint]

## this will execute the job

scala> sum\_two.show()

+-------+

|sum(id)|

+-------+

|1250000|

+-------+

## generate only the physical plan

scala> sum\_two.explain()

== Physical Plan ==

\*(3) HashAggregate(keys=[], functions=[sum(id#2L)])

+- Exchange SinglePartition, true, [id=#194]

+- \*(2) HashAggregate(keys=[], functions=[partial\_sum(id#2L)])

+- \*(2) Project [id#2L]

+- \*(2) BroadcastHashJoin [id#2L], [id#6L], Inner, BuildLeft

:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false])), [id=#188]

: +- \*(1) Project [(id#0L \* 5) AS id#2L]

: +- \*(1) Range (1, 1000, step=1, splits=4)

+- \*(2) Range (1, 5000, step=2, splits=4)

## generate the logical (extended=true) and physical plan

scala> sum\_two.explain(extended=true)

== Parsed Logical Plan ==

'Project [unresolvedalias('sum('id), Some(org.apache.spark.sql.Column$$Lambda$2324/1825090914@49cb7d64))]

+- Project [id#2L]

+- Join Inner, (id#2L = id#6L)

:- Project [(id#0L \* cast(5 as bigint)) AS id#2L]

: +- Range (1, 1000, step=1, splits=Some(4))

+- Range (1, 5000, step=2, splits=Some(4))

== Analyzed Logical Plan ==

sum(id): bigint

Aggregate [sum(id#2L) AS sum(id)#20L]

+- Project [id#2L]

+- Join Inner, (id#2L = id#6L)

:- Project [(id#0L \* cast(5 as bigint)) AS id#2L]

: +- Range (1, 1000, step=1, splits=Some(4))

+- Range (1, 5000, step=2, splits=Some(4))

== Optimized Logical Plan ==

Aggregate [sum(id#2L) AS sum(id)#20L]

+- Project [id#2L]

+- Join Inner, (id#2L = id#6L)

:- Project [(id#0L \* 5) AS id#2L]

: +- Range (1, 1000, step=1, splits=Some(4))

+- Range (1, 5000, step=2, splits=Some(4))

== Physical Plan ==

\*(3) HashAggregate(keys=[], functions=[sum(id#2L)], output=[sum(id)#20L])

+- Exchange SinglePartition, true, [id=#194]

+- \*(2) HashAggregate(keys=[], functions=[partial\_sum(id#2L)], output=[sum#26L])

+- \*(2) Project [id#2L]

+- \*(2) BroadcastHashJoin [id#2L], [id#6L], Inner, BuildLeft

:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false])), [id=#188]

: +- \*(1) Project [(id#0L \* 5) AS id#2L]

: +- \*(1) Range (1, 1000, step=1, splits=4)

+- \*(2) Range (1, 5000, step=2, splits=4)

# Explaining the text explain plan

* Read the indentations like an Oracle execution plan
* \*(1) \* (2) \*(3) \*(4) means stages or the DAG dependencies
* “Exchange“ - means a shuffle happened which is an expensive operation in spark world

scala> sum\_two.explain()

== Physical Plan ==

\*(3) HashAggregate(keys=[], functions=[sum(id#2L)])

+- Exchange SinglePartition, true, [id=#194]

+- \*(2) HashAggregate(keys=[], functions=[partial\_sum(id#2L)])

+- \*(2) Project [id#2L]

+- \*(2) BroadcastHashJoin [id#2L], [id#6L], Inner, BuildLeft

:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false])), [id=#188]

: +- \*(1) Project [(id#0L \* 5) AS id#2L]

: +- \*(1) Range (1, 1000, step=1, splits=4)

+- \*(2) Range (1, 5000, step=2, splits=4)

# Exploring spark-ui explain plan

* Access spark-ui
  + http://localhost:4040/jobs/

## Simple RDD

scala> sc.parallelize(1 to 1000).count()

### Jobs

* Job ID is sorted DESC (highest is the most recent). Focus on Submitted column for the grouping of Job IDs
* the :24 means the specific line of code this behaves like the Oracle SQL\_PLAN\_LINE\_ID

count at <console>:24

* Click on the link

A picture containing table

Description automatically generated

### DAGs or Stages

* Clicking on the link shows the DAG visualization
* Clicking on each DAG visualization or stages link will drill down to tasks

A picture containing diagram

Description automatically generated

### Tasks

* This page shows the detailed elapsed time breakdown of tasks
  + Here you’ll be able to find the straggling tasks or skewness of elapsed time to a particular task due to serialization or data skewness
* Tasks are spawned by Executor
* Tasks are like PX slaves in Oracle and Executors are like QC coordinator
* Event Timeline shows the breakdown of time by task

Table, timeline

Description automatically generated

### Stages Summary

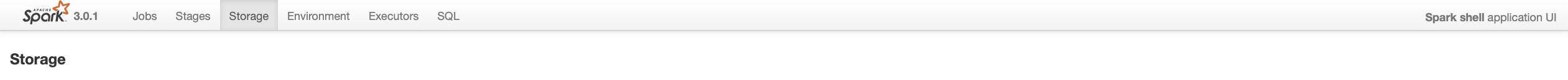
* Clicking the Stages tab shows the high-level Duration, Input, Output, Shuffle Read and Write
* This is just another view of the Stage ID breakdown, focus on Submitted column for the grouping on Job ID level

Table

Description automatically generated

### Storage

* This page shows the memory footprint of the objects



### Environment

* Runtime and Environment details

Graphical user interface, application, table

Description automatically generated

### Executors

* Executors summary and detailed statistics

Table

Description automatically generated with low confidence

### SQL

* Not related to RDDs, these are the spark SQL jobs executed

Graphical user interface, text, application

Description automatically generated

## DataFrame join and sum – Spark SQL

## create times5 dataframe

scala> val simpleNumbers = spark.range(1,1000)

**simpleNumbers**: **org.apache.spark.sql.Dataset[Long]** = [id: bigint]

scala> val times5 = simpleNumbers.selectExpr("id \* 5 as id")

**times5**: **org.apache.spark.sql.DataFrame** = [id: bigint]

## create moreNumbers dataframe

scala> val moreNumbers = spark.range(1,5000,2)

**moreNumbers**: **org.apache.spark.sql.Dataset[Long]** = [id: bigint]

## join the data frames

scala> val joined = times5.join(moreNumbers, "id")

**joined**: **org.apache.spark.sql.DataFrame** = [id: bigint]

scala> val sum\_two = joined.selectExpr("sum(id)")

**sum\_two**: **org.apache.spark.sql.DataFrame** = [sum(id): bigint]

## this will execute the job

scala> sum\_two.show()

+-------+

|sum(id)|

+-------+

|1250000|

+-------+

## generate only the physical plan

scala> sum\_two.explain()

== Physical Plan ==

\*(3) HashAggregate(keys=[], functions=[sum(id#2L)])

+- Exchange SinglePartition, true, [id=#194]

+- \*(2) HashAggregate(keys=[], functions=[partial\_sum(id#2L)])

+- \*(2) Project [id#2L]

+- \*(2) BroadcastHashJoin [id#2L], [id#6L], Inner, BuildLeft

:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false])), [id=#188]

: +- \*(1) Project [(id#0L \* 5) AS id#2L]

: +- \*(1) Range (1, 1000, step=1, splits=4)

+- \*(2) Range (1, 5000, step=2, splits=4)

### Jobs

* The action sum\_two.show() generated the Job ID 1 and 2 below (highlighted below)
* Job ID is sorted DESC (highest is the most recent). Focus on Submitted column for the grouping of Job IDs
* The Job ID 3 and 4 are re-execution of the variable sum\_two hence the same line id :26, also notice the faster duration (cached result)

A picture containing calendar

Description automatically generated

### DAGs or Stages

#### Job ID 1

A picture containing calendar

Description automatically generated

#### Job ID 2

A picture containing calendar

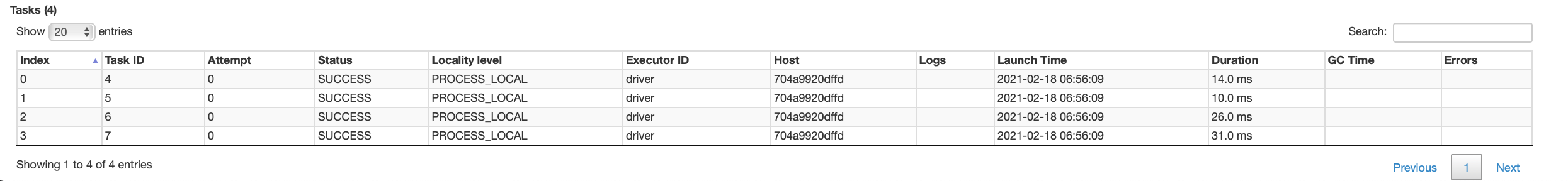
Description automatically generated

### Tasks

#### Job ID 1 - Stage 1

Graphical user interface, application, table, Word

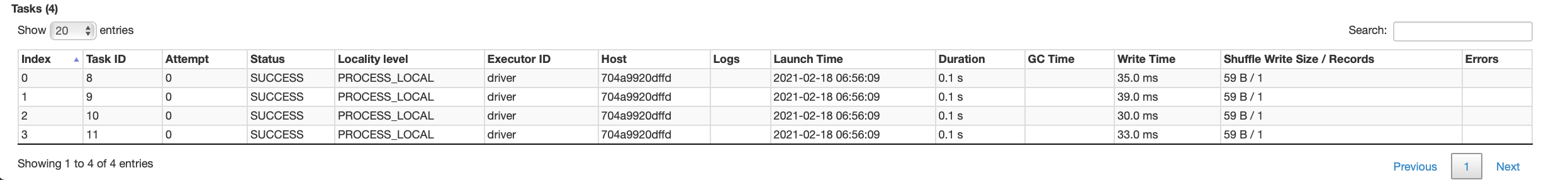
Description automatically generated



#### Job ID 2 - Stage 2

Timeline

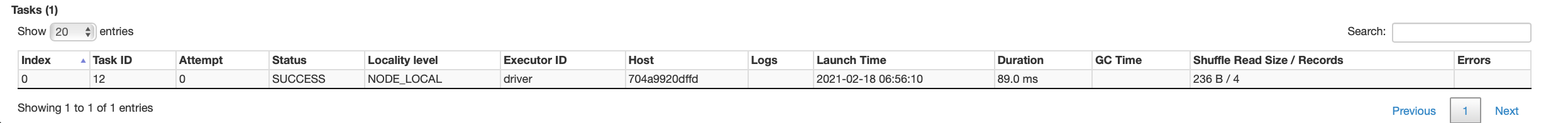
Description automatically generated



#### Job ID 2 - Stage 3

Graphical user interface, application, table, Excel

Description automatically generated



### Stages Summary

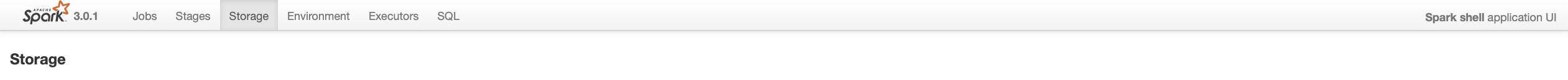
* Clicking the Stages tab shows the high-level Duration, Input, Output, Shuffle Read and Write
* This is just another view of the Stage ID breakdown, focus on Submitted column for the grouping on Job ID level

Table

Description automatically generated

### Storage

* This page shows the memory footprint of the objects



### Environment

* Runtime and Environment details

Graphical user interface, application, table

Description automatically generated

### Executors

* Executors summary and detailed statistics

Table

Description automatically generated with low confidence

### SQL

* Not related to RDDs, these are the spark SQL jobs executed

Graphical user interface, text, application

Description automatically generated

# Re-execution (cached result) comparison

* This shows the effect of cached result to the elapsed time – 2s vs 0.4s (cached)
* Stage 2 and 5 are the expensive operations
* Overall shorter
  + Task Deserialization Time
  + Executor Computing Time
  + Shuffle Write Time

## Summary

Graphical user interface, application

Description automatically generated

## Initial execution

Timeline

Description automatically generated

## 2nd execution

Timeline

Description automatically generated

…