Monitoring and Optimizing Spark Job

Dano Lee

Catalyst Optimizer

Catalyst Optimizer

- This is a rule based engine.
- It takes the logical plan which we expressed in a Dataframe API and then re-writes it into an Optimized Physical Plan. Physical Plan is developed before the query is executed.
- The catalyst optimizer is at the core of the Spark SQL's power and speed. It automatically finds the most efficient plan for applying your transformations and actions.

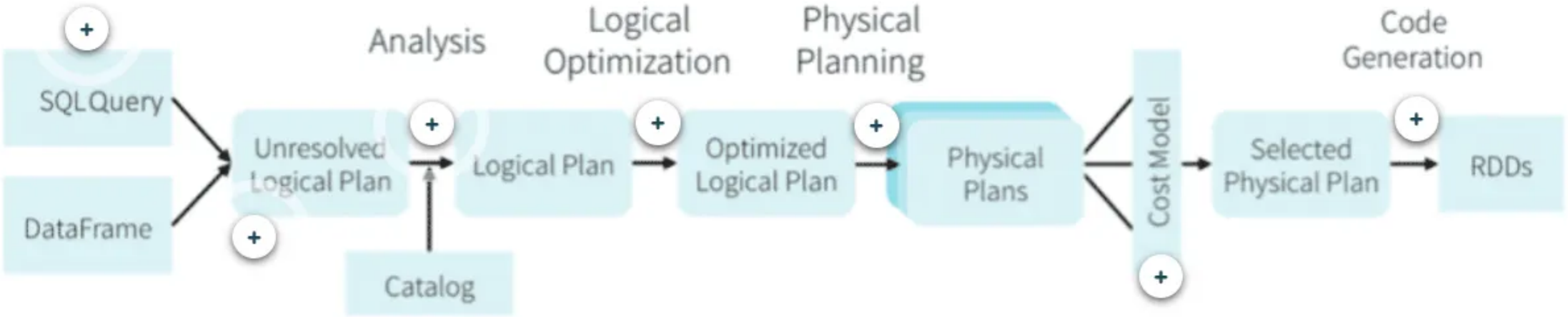
Benefits of Catalyst Optimizer

- **Optimize Query Plans**
  - It analyzes and transforms the logical execution plans of SQL queries to generate optimized physical plans that can be executed efficiently.
- **Enable Advanced Features**
  - It supports various advanced features such as predicate pushdown, join reordering, constant folding, and more.
- **Code Generation**
  - Catalyst also works closely with Tungsten, Spark's execution engine, to generate optimized code for the execution of these plans.

Catalyst Optimizer

- Let's suppose you write the code in Java, Scala, Python and R in the primary language supported by Apache Spark. When we use this language we use Dataframe or Dataset API. **These are declarative API** that is telling Apache Spark **what we intend to do and NOT how to do**.
- As Apache Spark SQL is a declarative API is another way of expressing what we want the Apache Spark to do for us and NOT how to do.

Catalyst Optimizer



How Catalyst Optimizer Works Internally (1)

- **Parsing**:
  - The SQL query or DataFrame operation is parsed into an Abstract Syntax Tree (AST).
- **Logical Plan Generation**:
  - The AST is converted into a Logical Plan. The logical plan is a tree of relational operators (e.g., Filter, Project, Join) that describes the computation required by the query.
- **Logical Plan Optimization**:
  - The logical plan undergoes several optimization phases where Catalyst applies various optimization rules. These include:
    - **Constant Folding**: Simplifies expressions by evaluating constant expressions at compile time.
    - **Predicate Pushdown**: Moves filters as close to the data source as possible, reducing the amount of data that needs to be processed.
    - **Join Reordering**: Reorders joins to optimize the execution by minimizing the size of intermediate results.
    - **Projection Pruning**: Removes unnecessary columns from the query plan, reducing the amount of data read and processed.

How Catalyst Optimizer Works Internally (2)

- **Physical Plan Generation**:
  - The optimized logical plan is then converted into one or more Physical Plans. Each physical plan corresponds to a specific way to execute the query, considering different strategies like different join algorithms (e.g., broadcast join, sort-merge join).
- **Physical Plan Optimization**:
  - Catalyst compares different physical plans and chooses the most efficient one based on a cost model, which estimates the computational resources needed (e.g., CPU, I/O).
- **Code Generation**:
  - Finally, the selected physical plan is compiled into optimized Java bytecode using **Tungsten**'s code generation capabilities, which is then executed by Spark's runtime engine.

Dynamic Partition Pruning

- Dynamic Partition Pruning = **Predicate Push Down** + **Broadcast Hash Join**

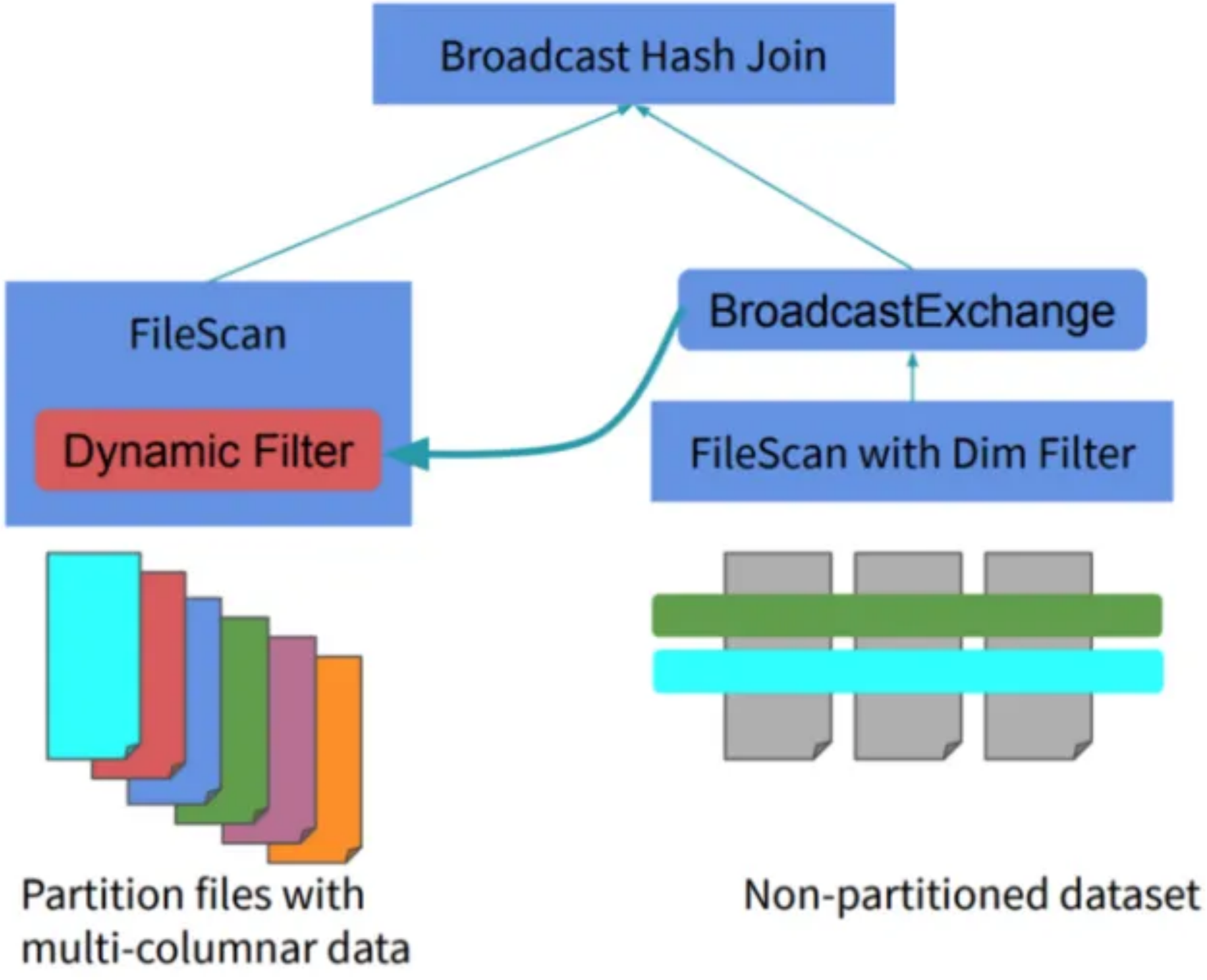
- **spark.sql.optimizer.dynamicPartitionPruning.enabled=true**

Example of DPP

- Imagine you have a **fact table with sales records (Sales)** and a **dimension table with dates (Date)**.
- If you **filter the Date table to only include January 2024 dates**, Spark will use DPP to **only scan partitions in the Sales table that contain records from January 2024**, instead of scanning the entire Sales table.

Dynamic Partition Pruning

- The smaller(**dimension**) table is queried and filtered. **A hash table is built as part of the filter query**.
- Spark uses the result of this query (and hash table) to create a broadcast variable
- Then, it will broadcast the filter to each executor
- At runtime, Spark's physical plan is changed so that the dynamic filter is applied to the bigger(**fact**) table. This dynamic filter is created as an internal subquery built from the filter applied to the smaller table.
- The dynamic filter is essentially an internal subquery that Spark creates based on the filter applied to the smaller table. When Spark scans the larger table, it applies this dynamic filter, **skipping partitions that do not meet the criteria**, thereby significantly reducing the amount of data that needs to be processed.


Broadcast Hash Join
FileScan
Dynamic Filter
BroadcastExchange
FileScan with Dim Filter
Partition files with
multi-columnar data
Non-partitioned dataset

source

Constraints of Dynamic Partition Pruning

- Tables that need to be pruned, **must be partitioned with any one of the join key columns.**
- It works only with Equi-join (joins with the '=' condition).
- DPP won't apply to the correlated subqueries
- DPP is useful for queries that follow the Star-schema architectural model.

Spark Web UI

Spark Web UI

- Spark Driver Web UI
  - The Spark Driver by default exposes the Spark Web UI using **information from the live application**.
  - This is the most comprehensive interface, as you can **see live data of your applications in terms of disk and memory utilization**.
  - You can easily locate and connect to this interface by **connecting on the YARN Resource Manger and then opening the Application Master URL available for the running Spark application**.

Spark History Server

- This is a service that Spark provides to review completed or running applications on the cluster.
- It's important to note that this interface **uses the Spark Event Logs** to populate the information that are available only at runtime (e.g. caching). It may miss out on some details that were only accessible while the application was actively running.

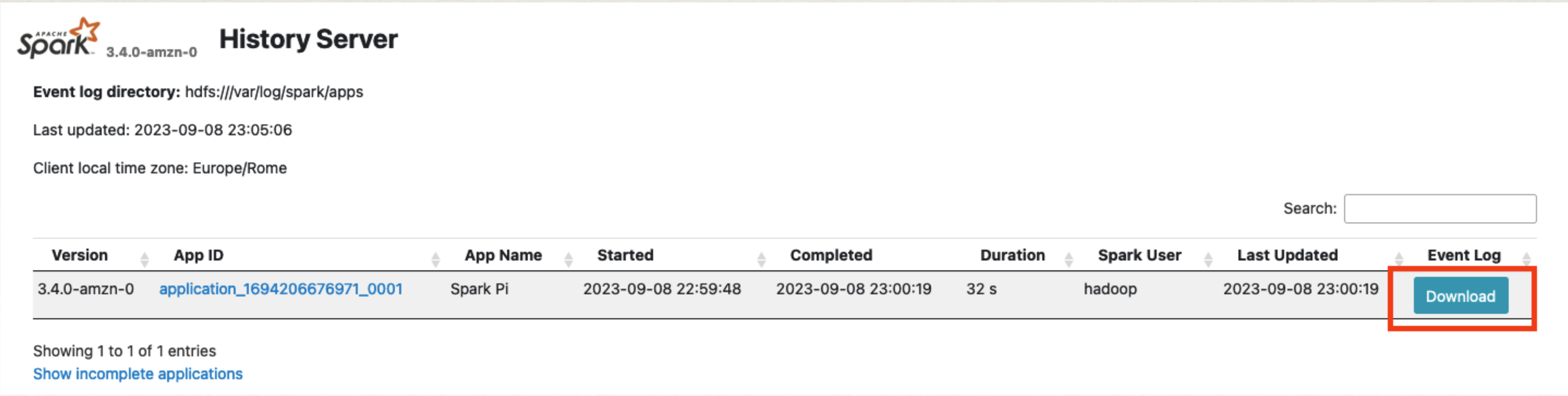
Spark History Server

- **./sbin/start-history-server.sh**

- The spark jobs themselves must be configured to log events, and to log them to the same shared, writable directory. For example, if the server was configured with a log directory of hdfs://namenode/shared/spark-logs, then the client-side options would be:
  - **spark.eventLog.enabled true**
  - **spark.eventLog.dir hdfs://namenode/shared/spark-logs**

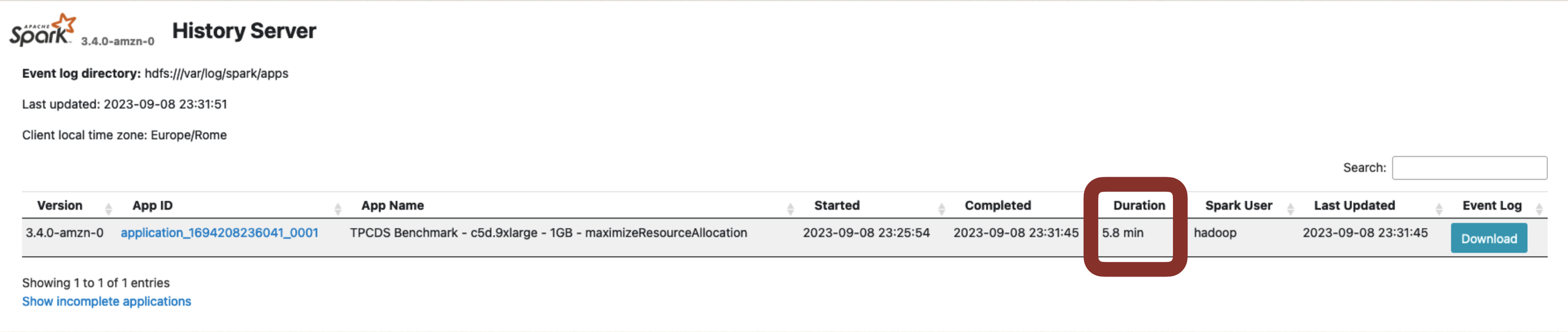
Retrieve Spark Event Logs

- When using Amazon EMR, **the Spark Event logs are enabled by default** and are automatically stored on the HDFS of the cluster where the job was running under the HDFS path /var/log/spark/apps/



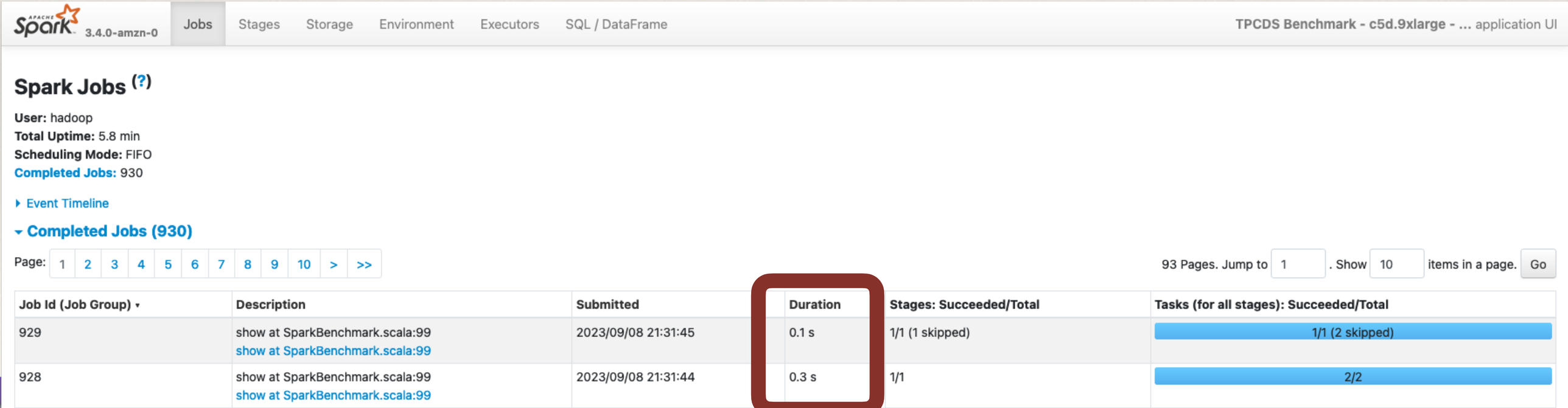
Applications in Spark Web UII

- Specifically the **duration** field is a useful field as it provides a metrics for the duration of the application since the Spark driver was launched and terminated, excluding additional submission details that are specific for different deployment models, so it can be useful to compare the Spark runtime across different versions or providers.



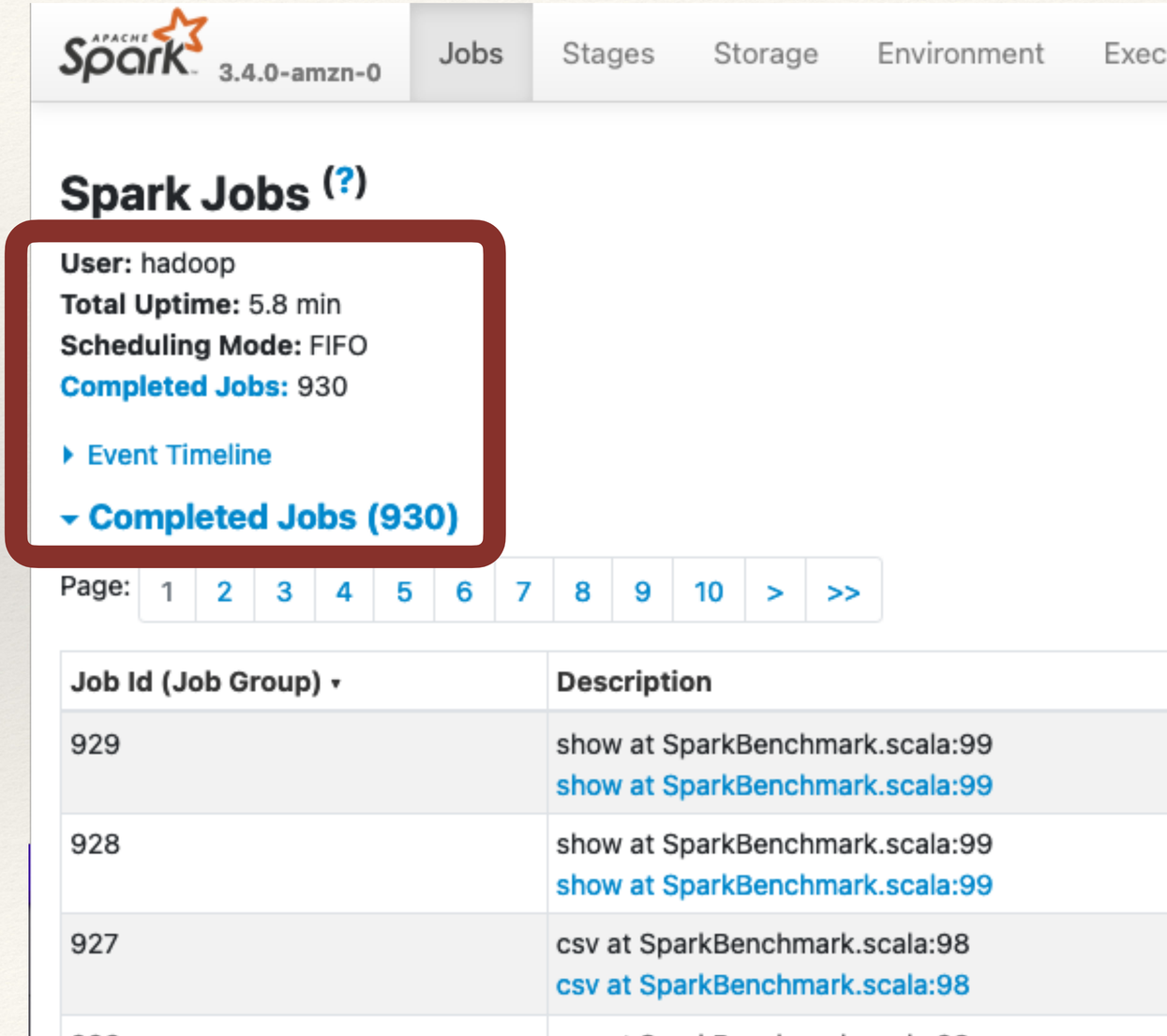
Jobs in Spark Web UI (1)

- it can give you a good indication of which portions of your code took more time to execute and you can also use it to compare two jobs executed in different environments or with different configurations to spot differences in terms of time processing. In those last cases is useful to sort your Jobs by Duration using the interface.



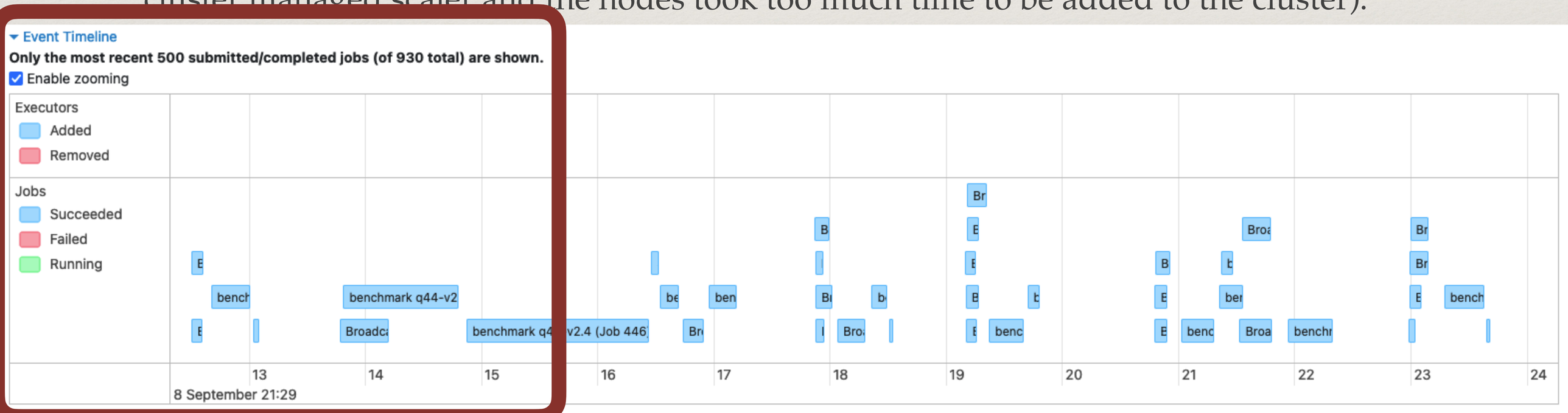
Jobs in Spark Web UI (2)

- Additionally, this page provides additional information on the top left page:
- **User*** The user who launched the application. In Amazon EMR this typically match the hadoop user unless you're using the Hadoop impersonation.
- **Total Uptime*** Time since the Spark application started till the completion of the last Job **Scheduling Mode*** The internal scheduler used within the Spark Application to execute the Spark Jobs. By default Spark uses a FIFO (First In First Out) scheduler.*



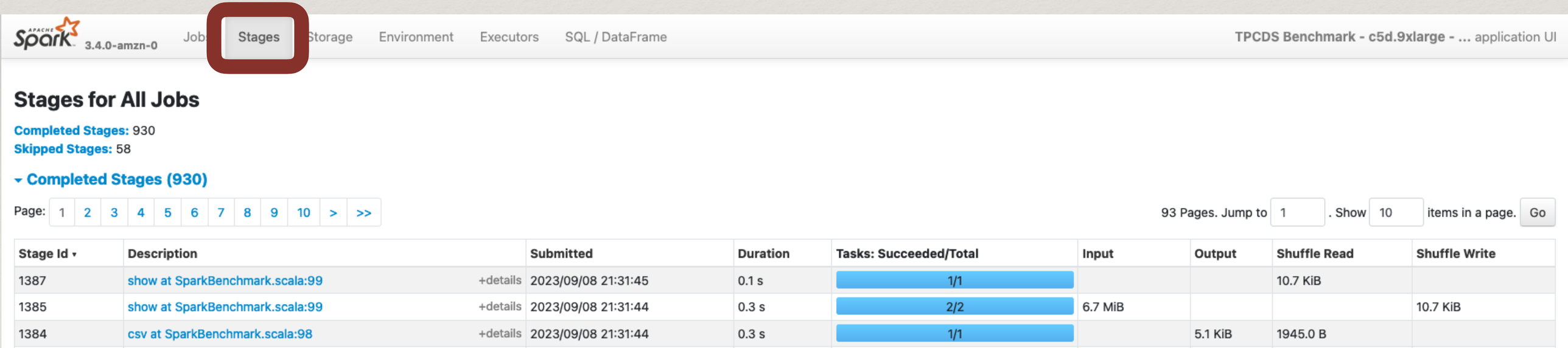
Jobs in Spark Web UI (3)

- Lastly, this page allows you you to review the lifecycle of your application, by expanding the **Event Timeline**** section
  - you can review how the different Spark Jobs were executed during the time, as also Spark Executors launch and termination, that can give you useful information to detect slow-downs due to the lack of resources (e.g. you're using the Spark Dynamic allocation along with a cluster managed scaler and the nodes took too much time to be added to the cluster).



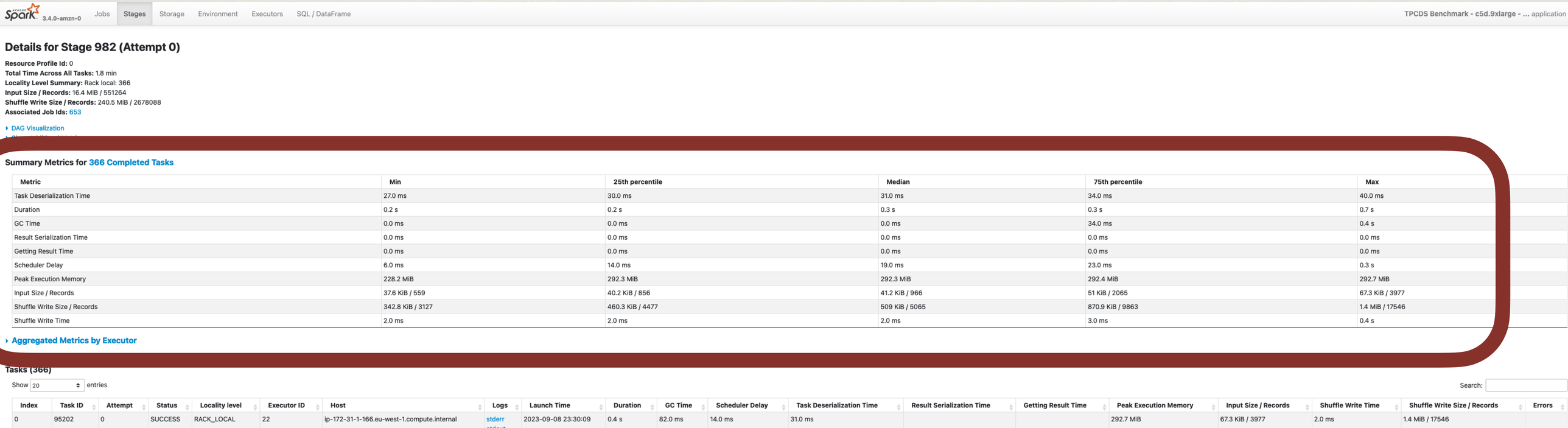
Stages in Spark Web UI (1)

- As for the Jobs page, you can review also all the Stages that have been processed in your application. This pages can be reached directly from the Web UI, and in this case it will display all the Stages of the application, or you can select a single Spark Job in the Jobs** section if you're only interested to the Stages processed in an individual Spark Job.



Stages in Spark Web UI (2)

- If you expand an individual Stage, you'll be redirected on a more detailed page where you can examine aggregated metrics of the Tasks processed in the stage.



Details for Stage 982 (Attempt 0)

Resource Profile Id: 0
Total Time Across All Tasks: 1.8 min
Locality Level Summary: Rack local: 366
Input Size / Records: 16.4 MiB / 551264
Shuffle Write Size / Records: 240.5 MiB / 2678088
Associated Job Ids: 653

▸ DAG Visualization
▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 366 Completed Tasks

Metric	Min	25th percentile
Task Deserialization Time	27.0 ms	30.0 ms
Duration	0.2 s	0.2 s
GC Time	0.0 ms	0.0 ms
Result Serialization Time	0.0 ms	0.0 ms
Getting Result Time	0.0 ms	0.0 ms
Scheduler Delay	6.0 ms	14.0 ms
Peak Execution Memory	228.2 MiB	292.3 MiB
Input Size / Records	37.6 KiB / 559	40.2 KiB / 856
Shuffle Write Size / Records	342.8 KiB / 3127	460.3 KiB / 4477
Shuffle Write Time	2.0 ms	2.0 ms

▸ Aggregated Metrics by Executor

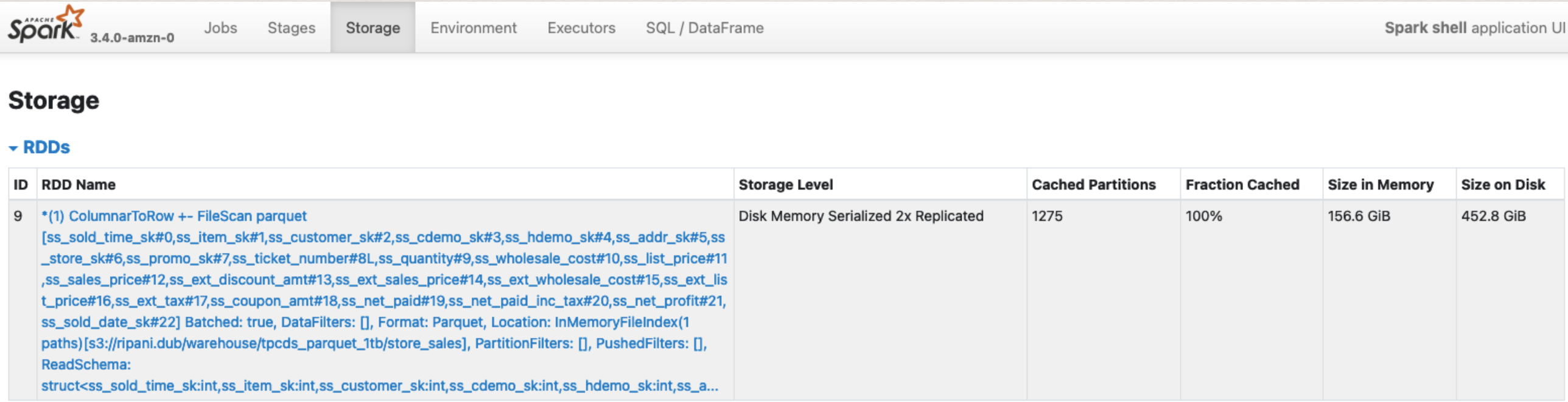
Tasks (366)

Show 20 entries

Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Tim
0	95202	0	SUCCESS	RACK_LOCAL	22	ip-172-31-1-166.eu-west-1.compute.internal	stderr stdout	2023-09-08 23:30:09	0.4 s	82.0 ms

Storage in Spark Web UI

- TheStorage page** contains **information about RDD blocks that have been cached or persisted in memory or on the local disks of the cluster**. This page will show some details only if you explicitly invoke a persist or cache operation against a Spark Dataframe. More in detail, the page shows for each RDD:



RDD Storage Info for *(1) ColumnarToRow +- FileScan parquet [ss_sold_time_sk#0,ss_item_sk#1,ss_customer_sk#2,ss_cdemo_sk#...

Storage Level: Disk Memory Serialized 2x Replicated
Cached Partitions: 1278
Total Partitions: 1278
Memory Size: 156.6 GiB
Disk Size: 459.9 GiB

Data Distribution on 6 Executors

Host	On Heap Memory Usage	Off Heap Memory Usage	Disk Usage
ip-172-31-3-222.eu-west-1.compute.internal:44907	26.2 GiB (336.4 MiB Remaining)	0.0 B (0.0 B Remaining)	89.1 GiB
ip-172-31-3-244.eu-west-1.compute.internal:43799	26.3 GiB (267.1 MiB Remaining)	0.0 B (0.0 B Remaining)	44.3 GiB
ip-172-31-3-170.eu-west-1.compute.internal:46697	25.8 GiB (751.3 MiB Remaining)	0.0 B (0.0 B Remaining)	91.4 GiB
ip-172-31-3-36.eu-west-1.compute.internal:46023	26.1 GiB (476.3 MiB Remaining)	0.0 B (0.0 B Remaining)	94.3 GiB
ip-172-31-3-39.eu-west-1.compute.internal:43111	26.0 GiB (594.8 MiB Remaining)	0.0 B (0.0 B Remaining)	49.5 GiB
ip-172-31-3-196.eu-west-1.compute.internal:39167	26.2 GiB (344.6 MiB Remaining)	0.0 B (0.0 B Remaining)	91.4 GiB

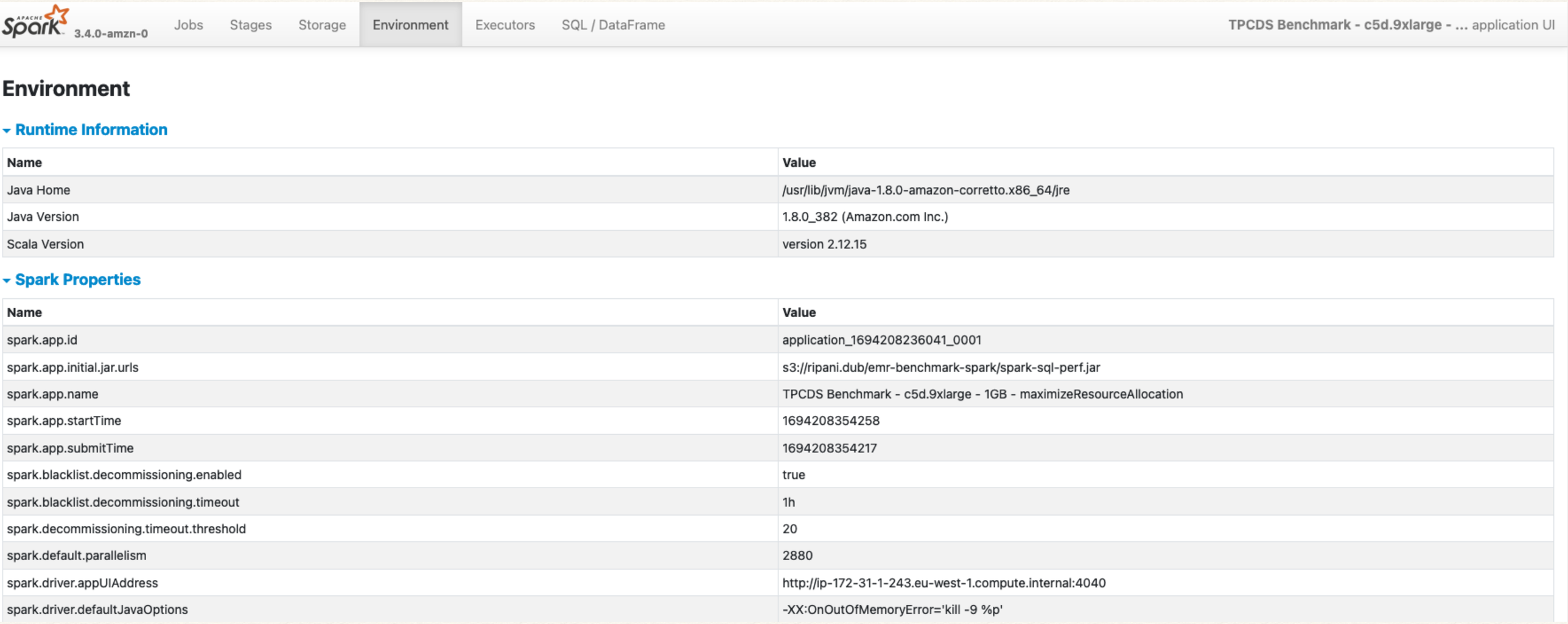
1278 Partitions

Page: < 1 2 3 4 5 6 7 8 9 10 > >>

128 Pages. Jump to 5 . Show 10 items in a page. Go

Block Name ▾	Storage Level	Size in Memory	Size on Disk	Executors
rdd_9_962	Disk Serialized 2x Replicated	0.0 B	388.9 MiB	ip-172-31-3-170.eu-west-1.compute.internal:46697 ip-172-31-3-39.eu-west-1.compute.internal:43111
rdd_9_961	Disk Serialized 2x Replicated	0.0 B	389.1 MiB	ip-172-31-3-196.eu-west-1.compute.internal:39167 ip-172-31-3-36.eu-west-1.compute.internal:46023
rdd_9_960	Disk Serialized 2x Replicated	0.0 B	389.1 MiB	ip-172-31-3-170.eu-west-1.compute.internal:46697 ip-172-31-3-39.eu-west-1.compute.internal:43111
rdd_9_96	Memory Serialized 2x Replicated	613.0 MiB	0.0 B	ip-172-31-3-170.eu-west-1.compute.internal:46697 ip-172-31-3-36.eu-west-1.compute.internal:46023
rdd_9_959	Disk Serialized 2x Replicated	0.0 B	389.5 MiB	ip-172-31-3-170.eu-west-1.compute.internal:46697 ip-172-31-3-36.eu-west-1.compute.internal:46023
rdd_9_958	Disk Serialized 2x Replicated	0.0 B	389.7 MiB	ip-172-31-3-222.eu-west-1.compute.internal:44907 ip-172-31-3-39.eu-west-1.compute.internal:43111
rdd_9_957	Disk Serialized 2x Replicated	0.0 B	388.8 MiB	ip-172-31-3-170.eu-west-1.compute.internal:46697 ip-172-31-3-222.eu-west-1.compute.internal:44907
rdd_9_956	Disk Serialized 2x Replicated	0.0 B	389.9 MiB	ip-172-31-3-222.eu-west-1.compute.internal:44907 ip-172-31-3-39.eu-west-1.compute.internal:43111

Environment in Spark Web UI



Executors in Spark Web UI

Executors

Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	On Heap Storage Memory	Off Heap Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(7)	2556	156.6 GiB / 169.8 GiB	156.6 GiB / 169.8 GiB	0.0 B / 0.0 B	459.9 GiB	96	0	0	3104	3104	11.4 h (46 min)	127.6 GiB	62.4 KiB	62.4 KiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B / 0.0 B	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(7)	2556	156.6 GiB / 169.8 GiB	156.6 GiB / 169.8 GiB	0.0 B / 0.0 B	459.9 GiB	96	0	0	3104	3104	11.4 h (46 min)	127.6 GiB	62.4 KiB	62.4 KiB	0

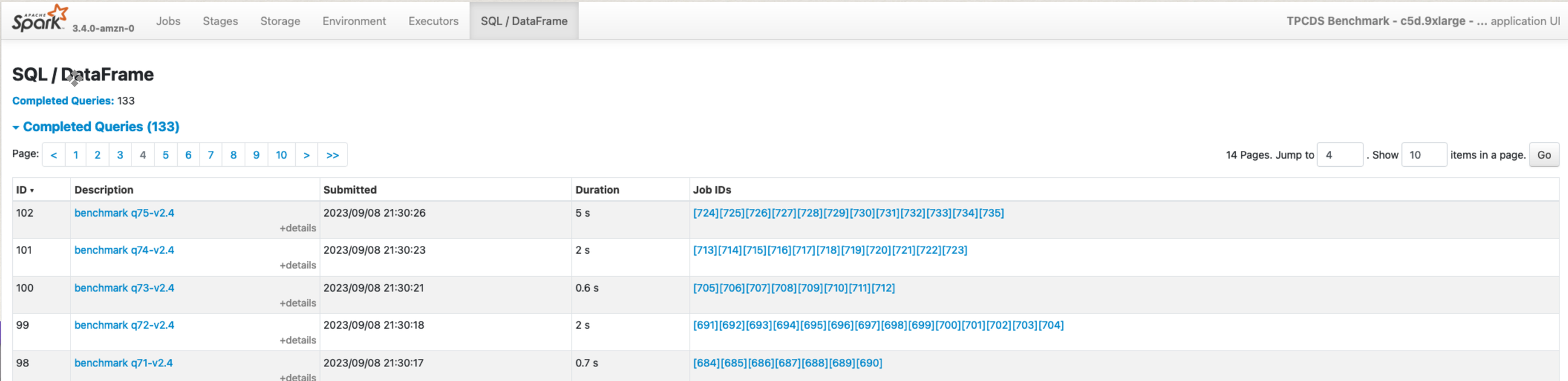
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	On Heap Storage Memory	Off Heap Storage Memory	Peak JVM Memory OnHeap / OffHeap	Peak Execution Memory OnHeap / OffHeap	Peak Storage Memory OnHeap / OffHeap	Peak Pool Memory Direct / Mapped	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	ip-172-31-3-68.eu-west-1.compute.internal:41281	Active	0	66.9 KiB / 10.5 GiB	66.9 KiB / 10.5 GiB	0.0 B / 0.0 B	920.2 MiB / 274.8 MiB	0.0 B / 0.0 B	1012.2 KiB / 0.0 B	6.5 MiB / 0.0 B	0.0 B	0	0	0	0	0	18 min (0.8 s)	0.0 B	0.0 B	0.0 B		Thread Dump
1	ip-172-31-3-170.eu-west-1.compute.internal:46697	Active	473	25.8 GiB / 26.6 GiB	25.8 GiB / 26.6 GiB	0.0 B / 0.0 B	43.8 GiB / 142.3 MiB	0.0 B / 0.0 B	26.6 GiB / 0.0 B	2.6 GiB / 1.7 GiB	91.4 GiB	16	0	0	679	679	2.1 h (7.6 min)	22.5 GiB	0.0 B	10.6 KiB	stdout stderr	Thread Dump
2	ip-172-31-3-222.eu-west-1.compute.internal:44907	Active	466	26.2 GiB / 26.6 GiB	26.2 GiB / 26.6 GiB	0.0 B / 0.0 B	40.6 GiB / 141.5 MiB	0.0 B / 0.0 B	26.6 GiB / 0.0 B	2.3 GiB / 1.8 GiB	89.1 GiB	16	0	0	701	701	2.1 h (7.3 min)	23.9 GiB	0.0 B	11.4 KiB	stdout stderr	Thread Dump
3	ip-172-31-3-196.eu-west-1.compute.internal:39167	Active	473	26.2 GiB / 26.6 GiB	26.2 GiB / 26.6 GiB	0.0 B / 0.0 B	41.6 GiB / 141.5 MiB	0.0 B / 0.0 B	26.5 GiB / 0.0 B	2.2 GiB / 1.1 GiB	91.4 GiB	16	0	0	680	680	2.1 h (8.0 min)	22.7 GiB	0.0 B	10.9 KiB	stdout stderr	Thread Dump
4	ip-172-31-3-36.eu-west-1.compute.internal:46023	Active	482	26.1 GiB / 26.6 GiB	26.1 GiB / 26.6 GiB	0.0 B / 0.0 B	42.6 GiB / 144.3 MiB	0.0 B / 0.0 B	26.6 GiB / 0.0 B	2.2 GiB / 1.7 GiB	94.3 GiB	16	0	0	674	674	2.1 h (9.2 min)	23.9 GiB	62.4 KiB	11.4 KiB	stdout stderr	Thread Dump
5	ip-172-31-3-244.eu-west-1.compute.internal:43799	Active	317	26.3 GiB / 26.6 GiB	26.3 GiB / 26.6 GiB	0.0 B / 0.0 B	41.8 GiB / 131.7 MiB	0.0 B / 0.0 B	26.6 GiB / 0.0 B	1.7 GiB / 754 MiB	44.3 GiB	16	0	0	178	178	1.4 h (7.7 min)	16.8 GiB	0.0 B	8.7 KiB	stdout stderr	Thread Dump
6	ip-172-31-3-39.eu-west-1.compute.internal:43111	Active	345	26 GiB / 26.6 GiB	26 GiB / 26.6 GiB	0.0 B / 0.0 B	43.6 GiB / 130.3 MiB	0.0 B / 0.0 B	26.5 GiB / 0.0 B	1.9 GiB / 1.1 GiB	49.5 GiB	16	0	0	192	192	1.4 h (6.2 min)	17.8 GiB	0.0 B	9.4 KiB	stdout stderr	Thread Dump

SQL/DataFrame in Spark Web UI

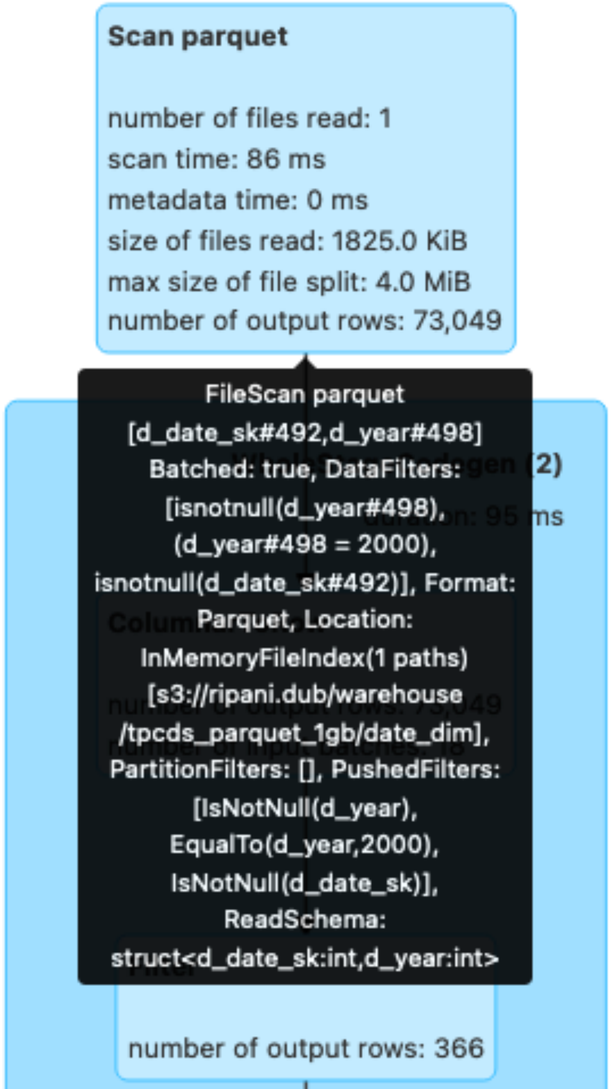
- SQL/DataFrame page summarizes **all Spark Queries executed in a Spark Application.**
- This page is only visible in the UI if your application is using Dataset or DataFrame Spark APIs. (Not when using RDD APIs)



Details for Query 30

Submitted Time: 2023/09/08 21:27:31
Duration: 1 s
Succeeded Jobs: 90 91 92 93 94 95

☐ Show the Stage ID and Task ID that corresponds to the max metric



Details

```
== Parsed Logical Plan ==
+- 'LocalLimit 100
   +- 'Sort ['i_item_id ASC NULLS FIRST], true
      +- 'Aggregate ['i_item_id], ['i_item_id, 'avg('ss_quantity) AS agg1#27406, 'avg('ss_list_price) AS agg2#27407, 'avg('s
         +- 'Filter (((((('ss_sold_date_sk = 'd_date_sk) AND ('ss_item_sk = 'i_item_sk)) AND ('ss_cdemo_sk = 'cd_demo_sk)) AN
            +- 'Join Inner
               :- 'Join Inner
               :  :- 'Join Inner
               :  :  :- 'Join Inner
               :  :  :  :- 'UnresolvedRelation [store_sales], [], false
               :  :  :  +- 'UnresolvedRelation [customer_demographics], [], false
               :  :  +- 'UnresolvedRelation [date_dim], [], false
               :  +- 'UnresolvedRelation [item], [], false
               +- 'UnresolvedRelation [promotion], [], false

== Analyzed Logical Plan ==
i_item_id: string, agg1: double, agg2: decimal(11,6), agg3: decimal(11,6), agg4: decimal(11,6)
GlobalLimit 100
+- LocalLimit 100
   +- Sort [i_item_id#565 ASC NULLS FIRST], true
      +- Aggregate [i_item_id#565], [i_item_id#565, avg(ss_quantity#139) AS agg1#27406, avg(ss_list_price#141) AS agg2#27407
         +- Filter (((((ss_sold_date_sk#152 = d_date_sk#492) AND (ss_item_sk#131 = i_item_sk#564)) AND (ss_cdemo_sk#133 = cd
            +- Join Inner
               :- Join Inner
               :  :- Join Inner
               :  :  :- Join Inner
```

```
:  :  :- Join Inner
:  :  :  :- SubqueryAlias store_sales
:  :  :  :  +- View (`store_sales`,
[ss_sold_time_sk#130,ss_item_sk#131,ss_customer_sk#132,ss_cdemo_sk#133,ss_hdemo_sk#134,ss_addr_sk#135,ss_store_sk#136,ss_pro
:  :  :  :     +- Relation
[ss_sold_time_sk#130,ss_item_sk#131,ss_customer_sk#132,ss_cdemo_sk#133,ss_hdemo_sk#134,ss_addr_sk#135,ss_store_sk#136,ss_pro
:  :  :  +- SubqueryAlias customer_demographics
:  :  :     +- View (`customer_demographics`, [cd_demo_sk#474,cd_gender#475,cd_marital_status#476,cd_educatio
:  :  :        +- Relation [cd_demo_sk#474,cd_gender#475,cd_marital_status#476,cd_education_status#477,cd_pur
:  :  +- SubqueryAlias date_dim
:  :     +- View (`date_dim`,
[d_date_sk#492,d_date_id#493,d_date#494,d_month_seq#495,d_week_seq#496,d_quarter_seq#497,d_year#498,d_dow#499,d_moy#500,d_do
:  :        +- Relation [d_date_sk#492,d_date_id#493,d_date#494,d_month_seq#495,d_week_seq#496,d_quarter_seq#
:  +- SubqueryAlias item
:     +- View (`item`, [i_item_sk#564,i_item_id#565,i_rec_start_date#566,i_rec_end_date#567,i_item_desc#568,i
:        +- Relation [i_item_sk#564,i_item_id#565,i_rec_start_date#566,i_rec_end_date#567,i_item_desc#568,i_cu
+- SubqueryAlias promotion
   +- View (`promotion`, [p_promo_sk#608,p_promo_id#609,p_start_date_sk#610,p_end_date_sk#611,p_item_sk#612,p_
      +- Relation [p_promo_sk#608,p_promo_id#609,p_start_date_sk#610,p_end_date_sk#611,p_item_sk#612,p_cost#61

== Optimized Logical Plan ==
GlobalLimit 100
+- LocalLimit 100
   +- Sort [i_item_id#565 ASC NULLS FIRST], true
      +- Aggregate [i_item_id#565], [i_item_id#565, avg(ss_quantity#139) AS agg1#27406, cast((avg(UnscaledValue(ss_list_price
         +- Project [ss_quantity#139, ss_list_price#141, ss_sales_price#142, ss_coupon_amt#148, i_item_id#565]
            +- Join Inner, (ss_promo_sk#137 = p_promo_sk#608)
               :- Project [ss_promo_sk#137, ss_quantity#139, ss_list_price#141, ss_sales_price#142, ss_coupon_amt#148, i_iten
               :  +- Join Inner, (ss_item_sk#131 = i_item_sk#564)
```

Spark Web UI

https://spark.apache.org/docs/latest/web-ui.html

AWS EMR Best Practices

https://aws.github.io/aws-emr-best-practices/docs/bestpractices/