阿里云 开发者社区 × 计算平台事业部 COMPUTING PLATFORM

BIG DATA DIGEST 大数据文摘

What's New in the Upcoming Apache Spark 3.0

Databricks 李潇

Meetup *S01

BigData+AI

About Me

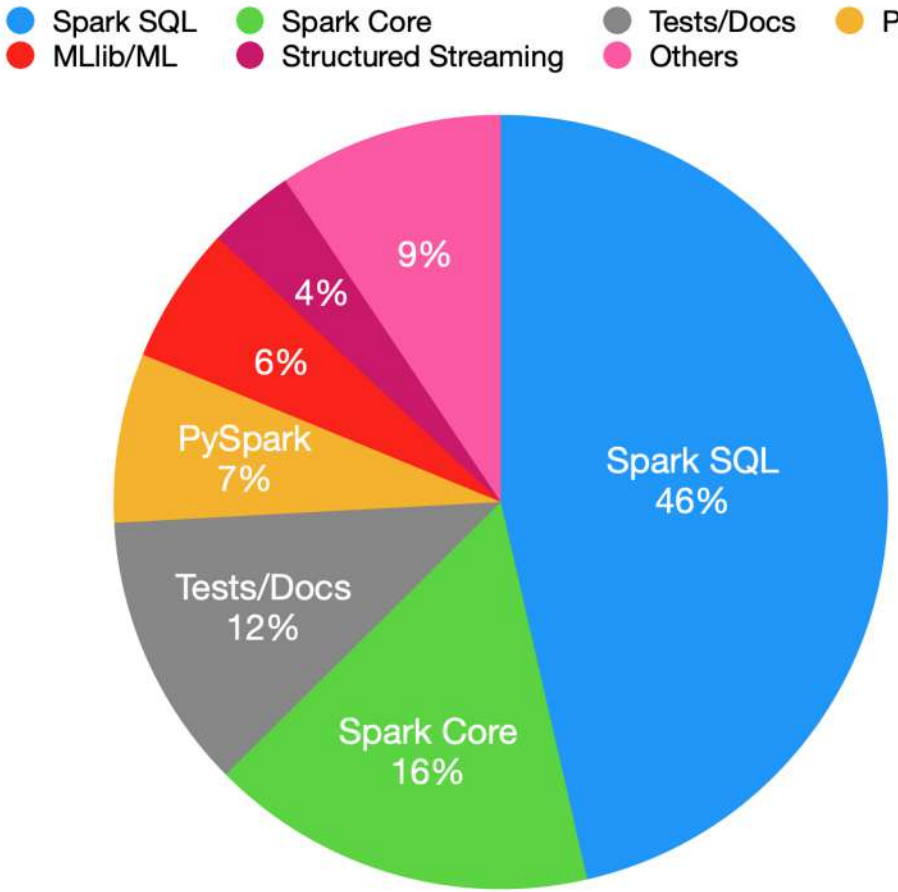
- Engineering Manager at Databricks
- Apache Spark Committer and PMC
- Previously, IBM Master Inventor
- Ph.D. in CISE at University of Florida
- GitHub: gatorsmile

databricks

Unified data analytics platform for accelerating innovation across data science, data engineering, and business analytics

Global company with 5,000 customers and 450+ partners

Original creators of popular data and machine learning open source projects



APACHE Spark™

3400+ Resolved JIRAs
in Spark 3.0 rc3

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Built-in Data Sources

Parquet/ORC Nested Column Pruning

CSV Filter Pushdown

Parquet: Nested Column Filter Pushdown

New Binary Data Source

Richer APIs

Accelerator-aware Scheduler

Built-in Functions

pandas UDF Enhancements

DELETE/UPDATE/ MERGE in Catalyst

SQL Compatibility

Overflow Checking

ANSI Store Assignment

Proleptic Gregorian Calendar

Reserved Keywords

Extensibility and Ecosystem

Data Source V2 API + Catalog Support

Hadoop 3 Support

Hive 3.x Metastore Hive 2.3 Execution

Java 11 Support

Monitoring and Debuggability

Structured Streaming UI

DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Achieve high performance for interactive, batch, streaming and ML workloads

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Achieve high performance for interactive, batch, streaming and ML workloads

Spark Catalyst Optimizer

Spark 1.x, Rule

Spark 2.x, Rule + Cost

Query Optimization in Spark 2.x

- Missing statistics
  - Expensive statistics collection
- Out-of-date statistics
  - Compute and storage separated
- Suboptimal Heuristics
  - Local
- Misestimated costs
  - Complex environments
  - User-defined functions

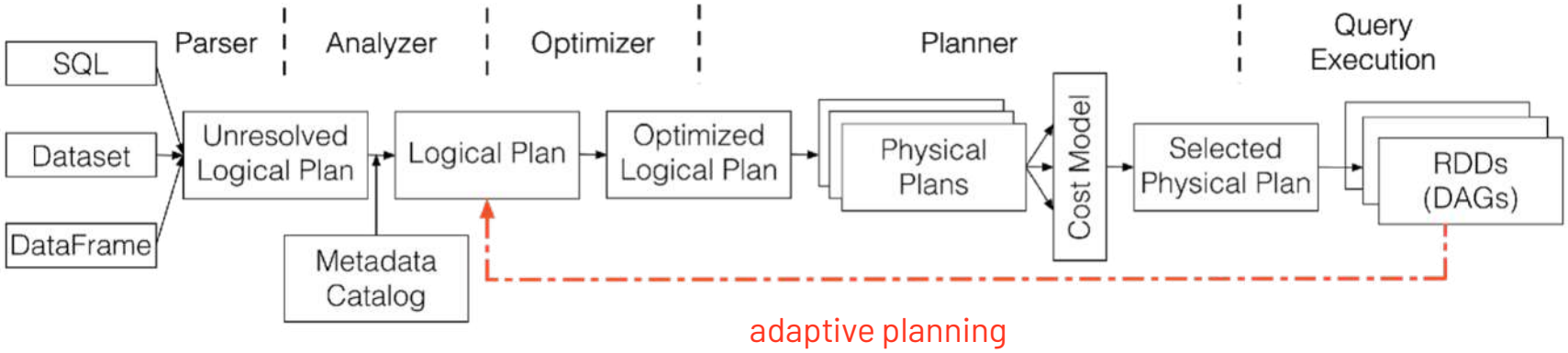
Spark Catalyst Optimizer

Spark 1.x, Rule

Spark 2.x, Rule + Cost

Spark 3.0, Rule + Cost + Runtime

Adaptive Query Execution [AQE]



Based on statistics of the finished plan nodes, re-optimize the execution plan of the remaining queries

Adaptive Query Execution

Based on statistics of the finished plan nodes, re-optimize the execution plan of the remaining queries

- Convert Sort Merge Join to Broadcast Hash Join
- Shrink the number of reducers
- Handle skew join

Blog post: https://databricks.com/blog/2020/05/29/adaptive-query-execution-speeding-up-spark-sql-at-runtime.html

One of the Most Popular Performance Tuning Tips

- Choose Broadcast Hash Join?
  - Increase “spark.sql.autoBroadcastJoinThreshold”?
  - Use “broadcast” hint?

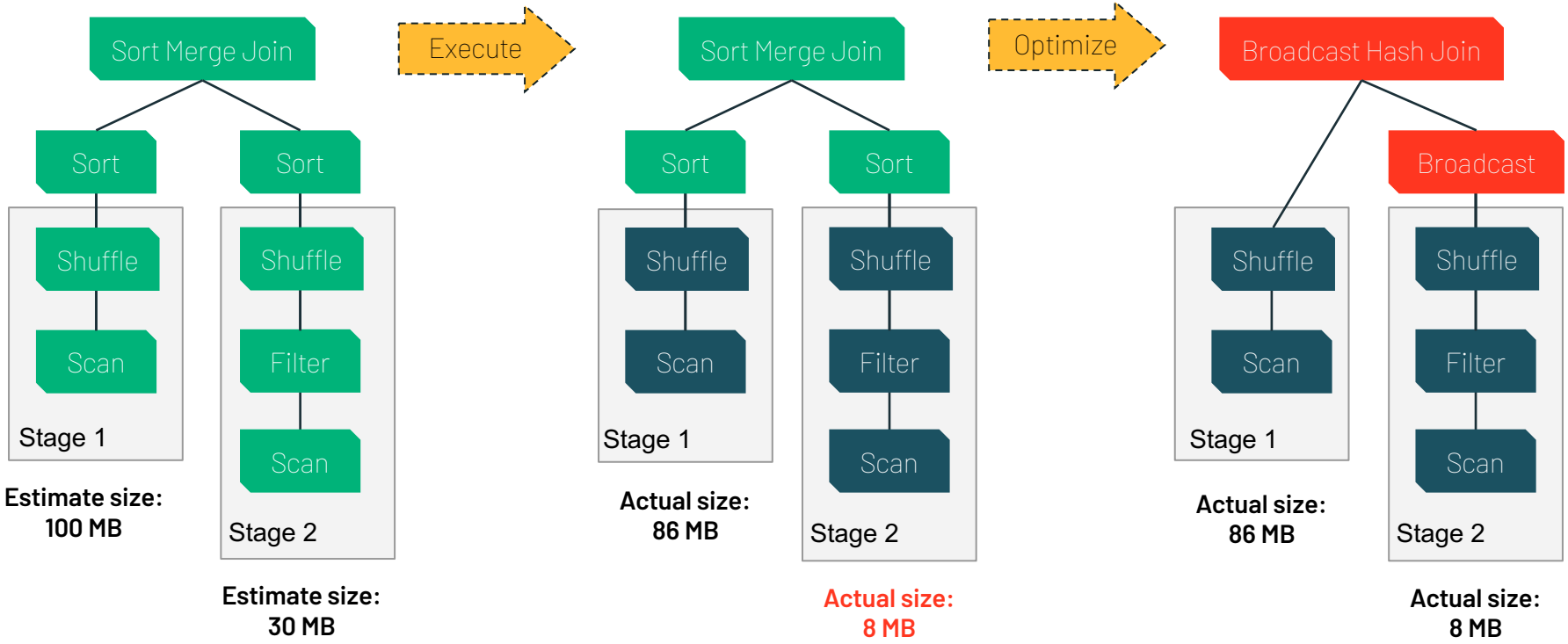
However

- Hard to tune
- Hard to maintain over time
- OOM…

Why Spark not Making the Best Choice Automatically?

- Inaccurate/missing statistics;
- File is compressed; columnar store;
- Complex filters; black-box UDFs;
- Complex query fragments…

Convert Sort Merge Join to Broadcast Hash Join



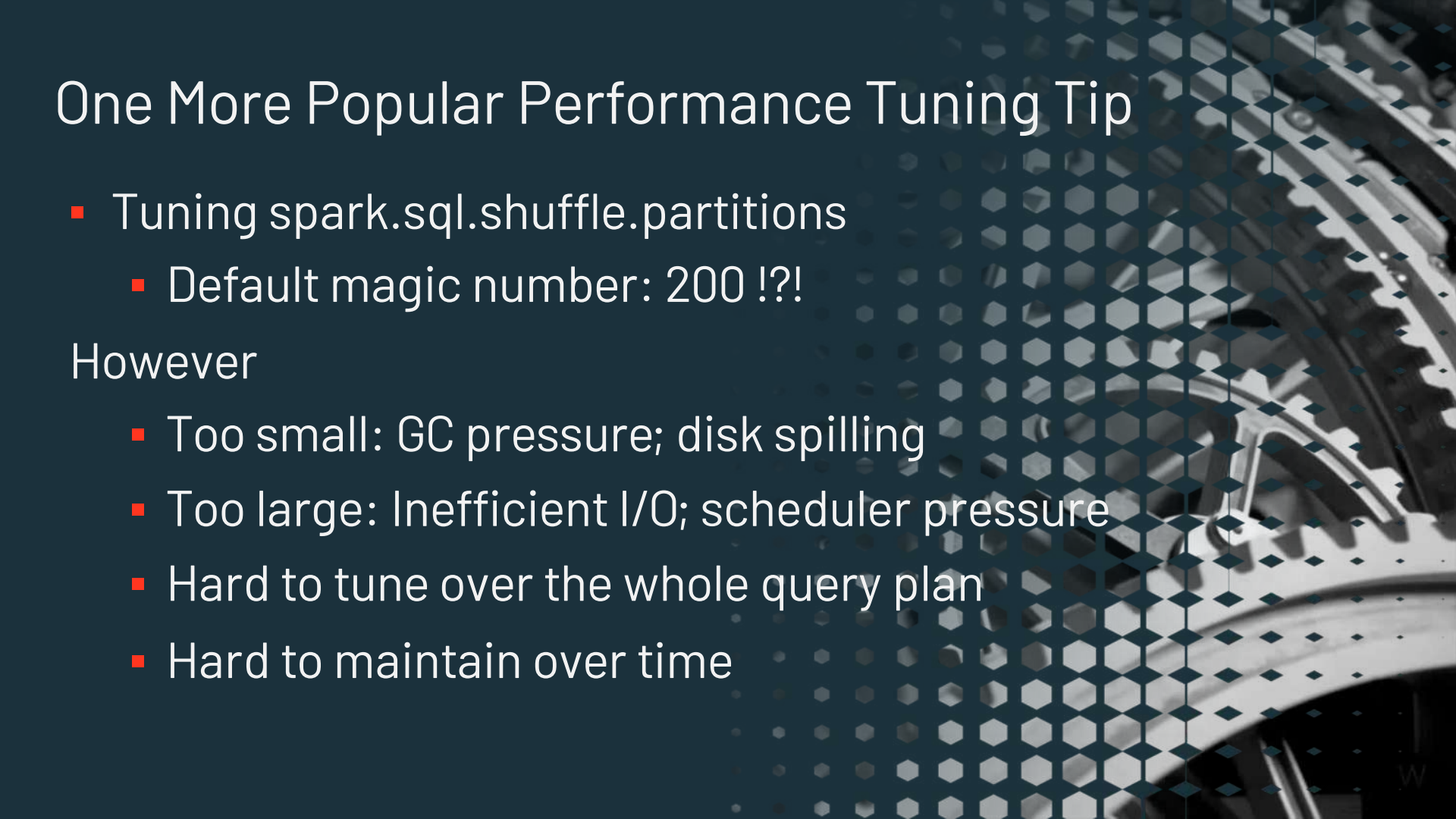
databricks

One More Popular Performance Tuning Tip

- Tuning spark.sql.shuffle.partitions
  - Default magic number: 200 !?!

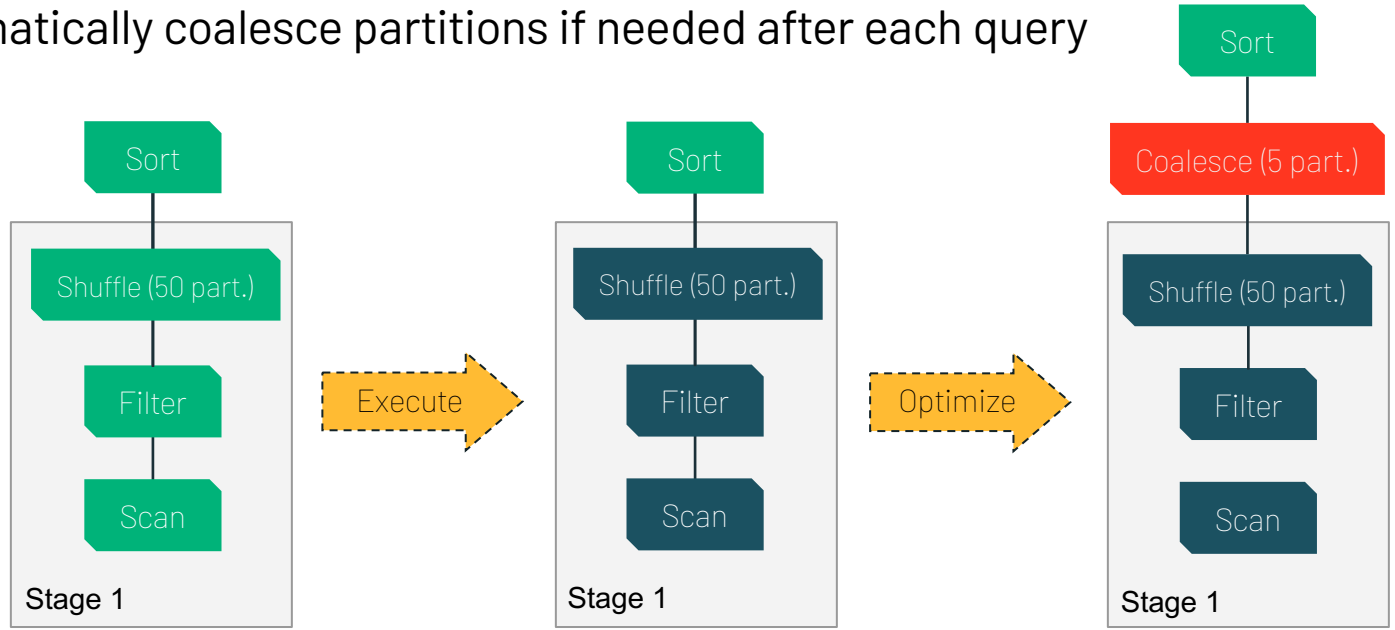
However

- Too small: GC pressure; disk spilling
- Too large: Inefficient I/O; scheduler pressure
- Hard to tune over the whole query plan
- Hard to maintain over time



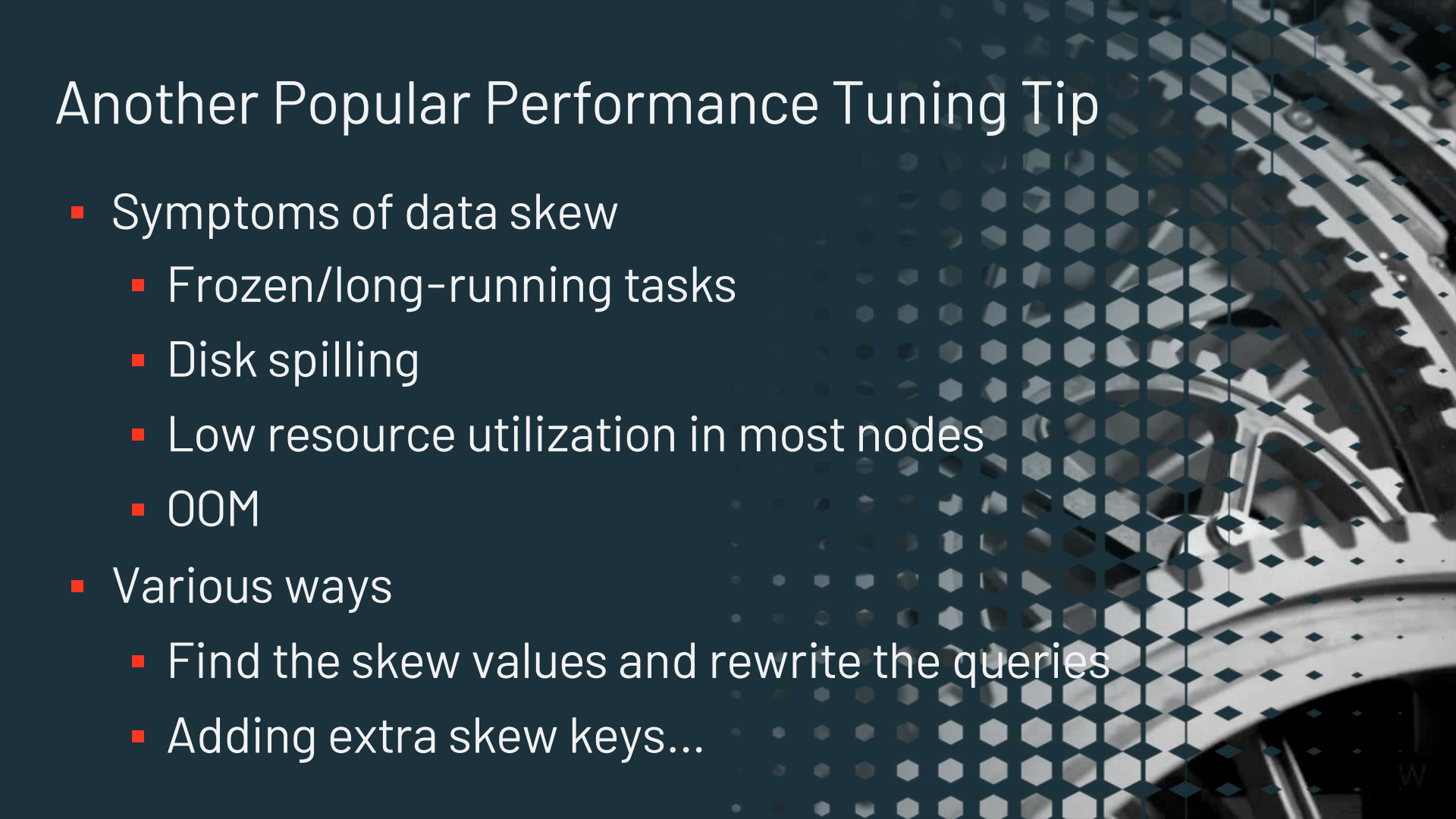
Dynamically Coalesce Shuffle Partitions

- Set the initial partition number high to accommodate the largest data size of the entire query execution
- Automatically coalesce partitions if needed after each query stage

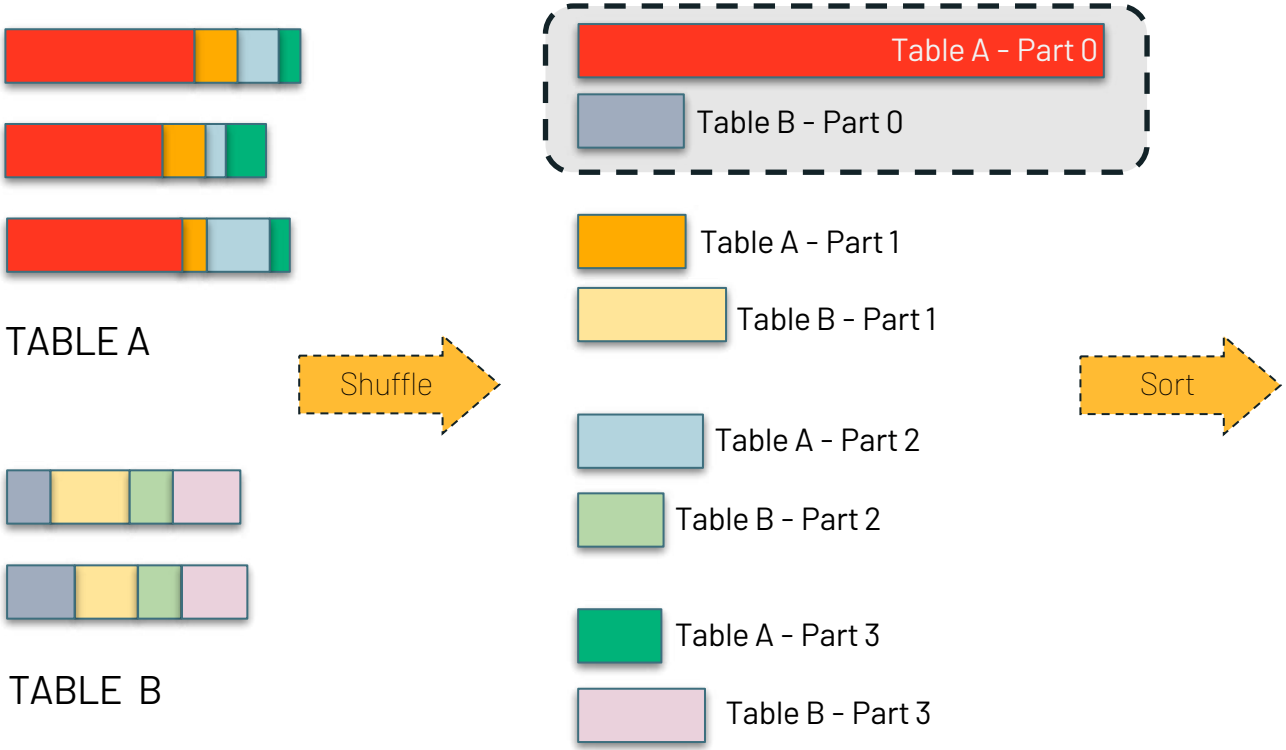


Another Popular Performance Tuning Tip

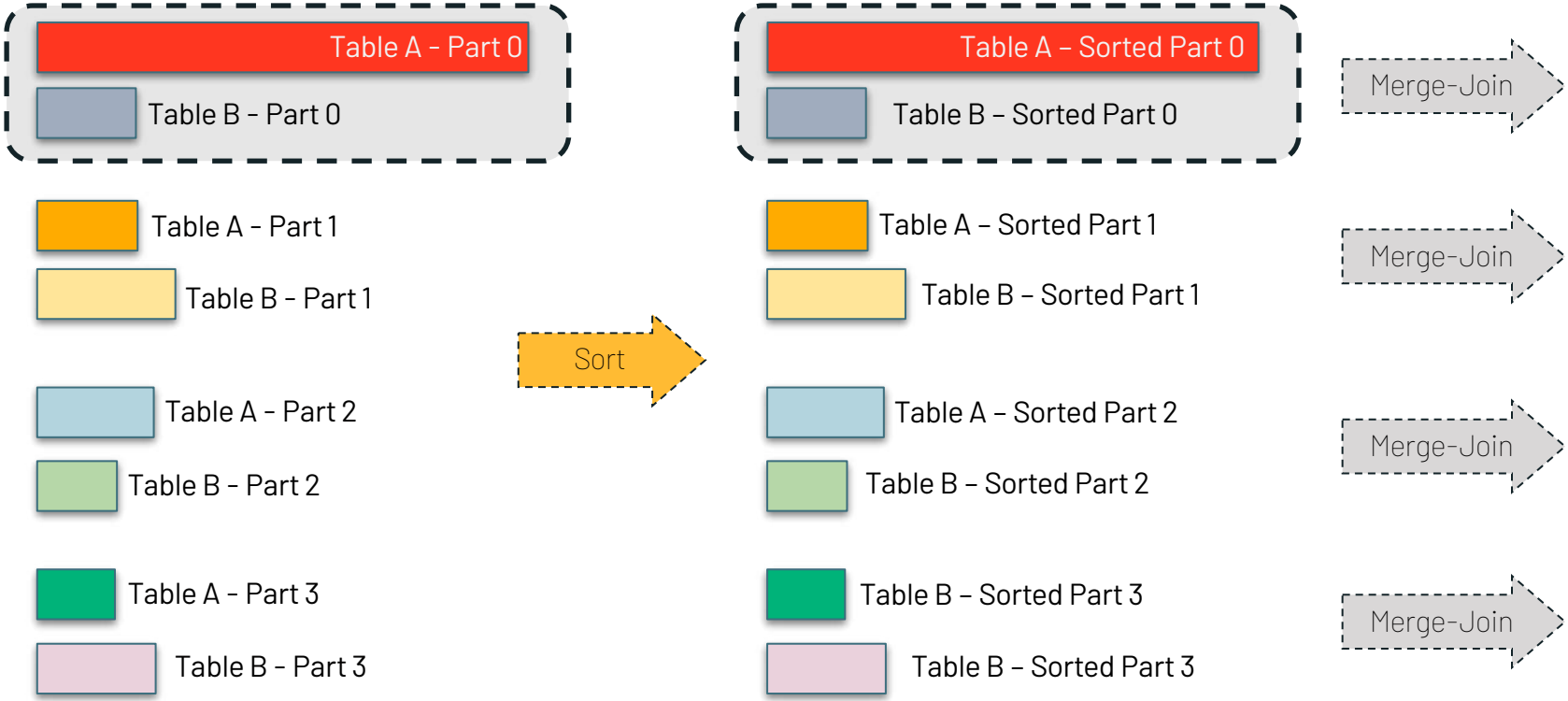
- Symptoms of data skew
  - Frozen/long-running tasks
  - Disk spilling
  - Low resource utilization in most nodes
  - OOM
- Various ways
  - Find the skew values and rewrite the queries
  - Adding extra skew keys...



Data Skew in Sort Merge Join

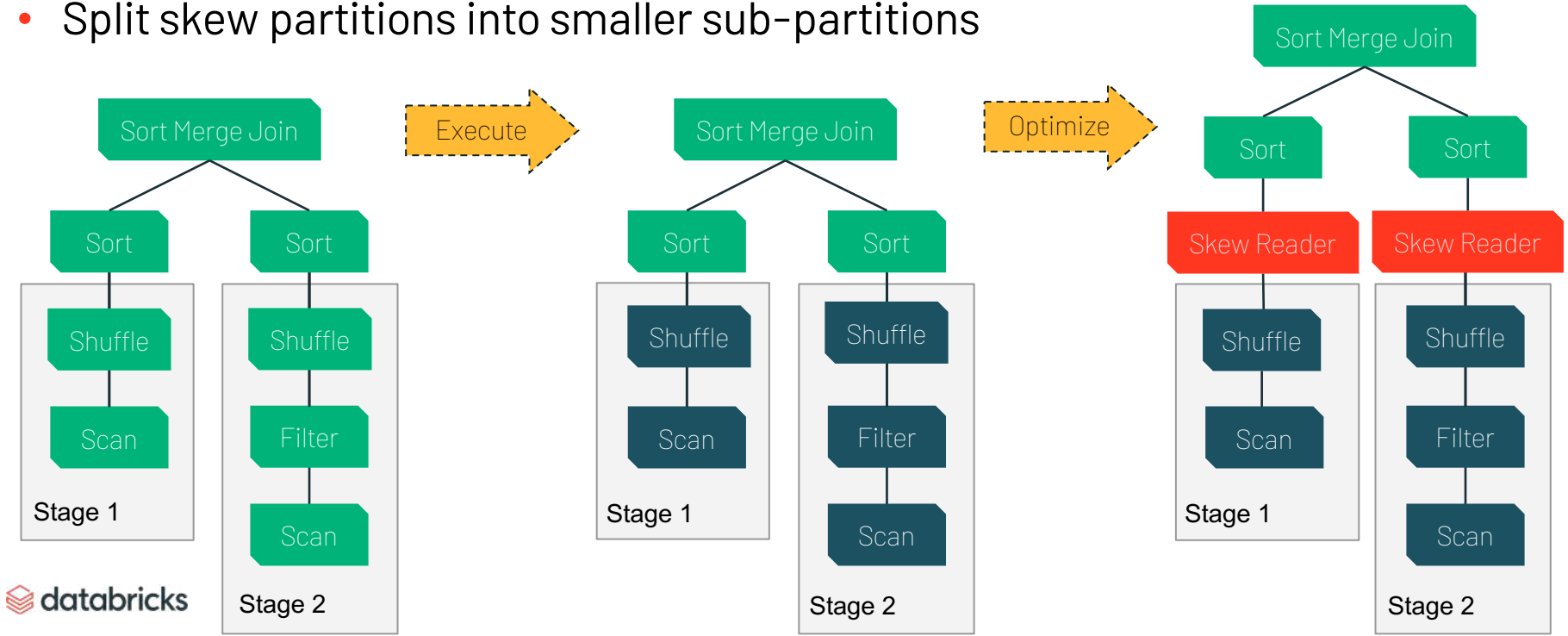


Data Skew in Sort Merge Join



Dynamically Optimize Skew Joins

- Detect skew from partition sizes using runtime statistics
- Split skew partitions into smaller sub-partitions



databricks

Dynamically Optimize Skew Joins

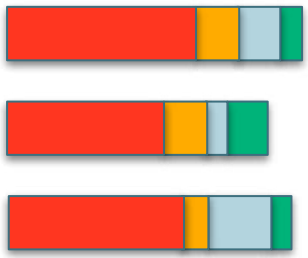


TABLE A

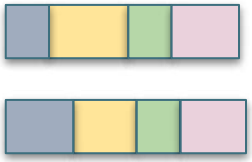


TABLE B

Shuffle

Table A - Part 0 – Split 0

Table B - Part 0

Table A - Part 0 – Split 1

Table B - Part 0

Table A - Part 0 – Split 2

Table B - Part 0

Sort

Table A - Part 1

Table B - Part 1

Table A - Part 2

Table B - Part 2

Table A - Part 3

Table B - Part 3

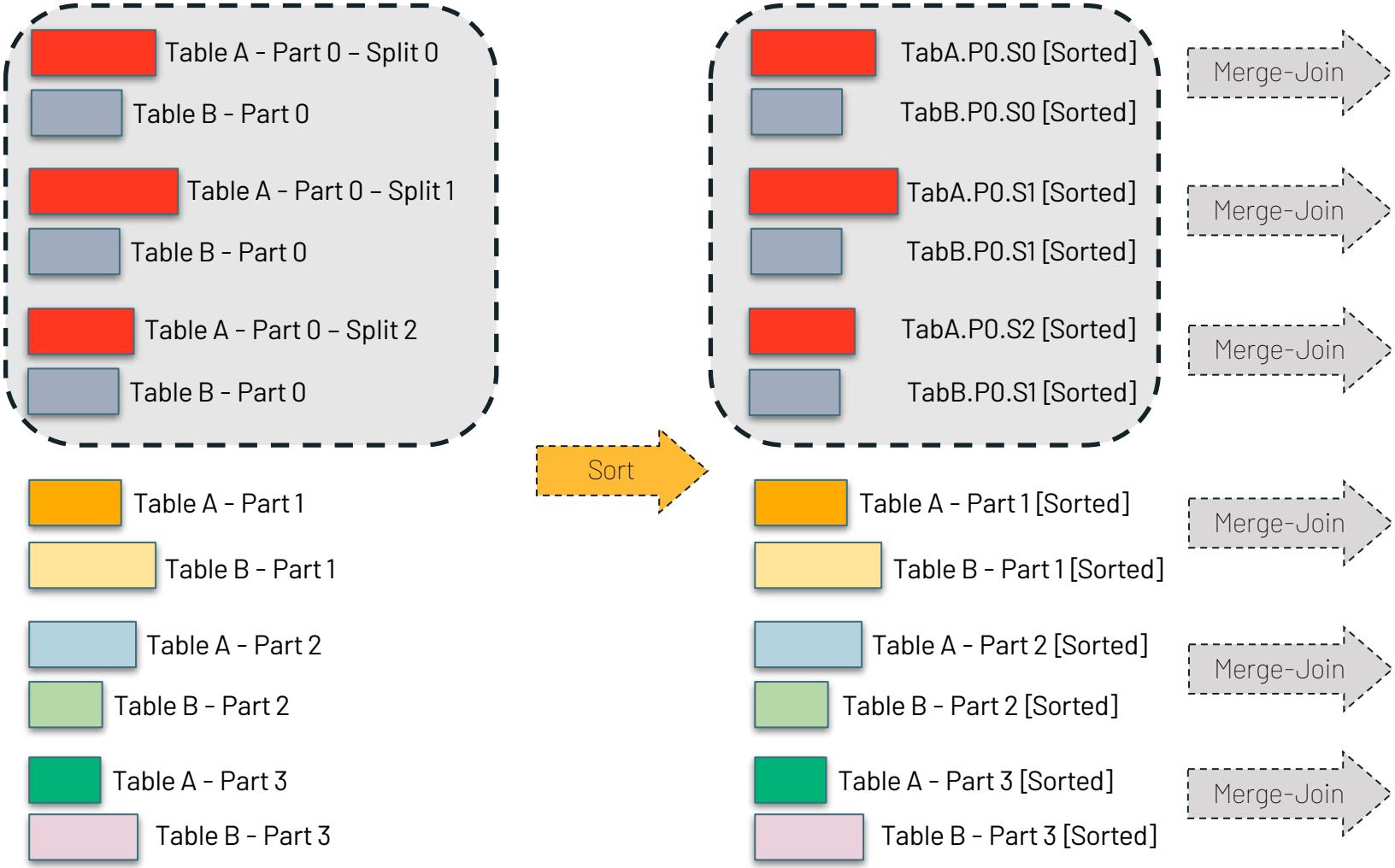
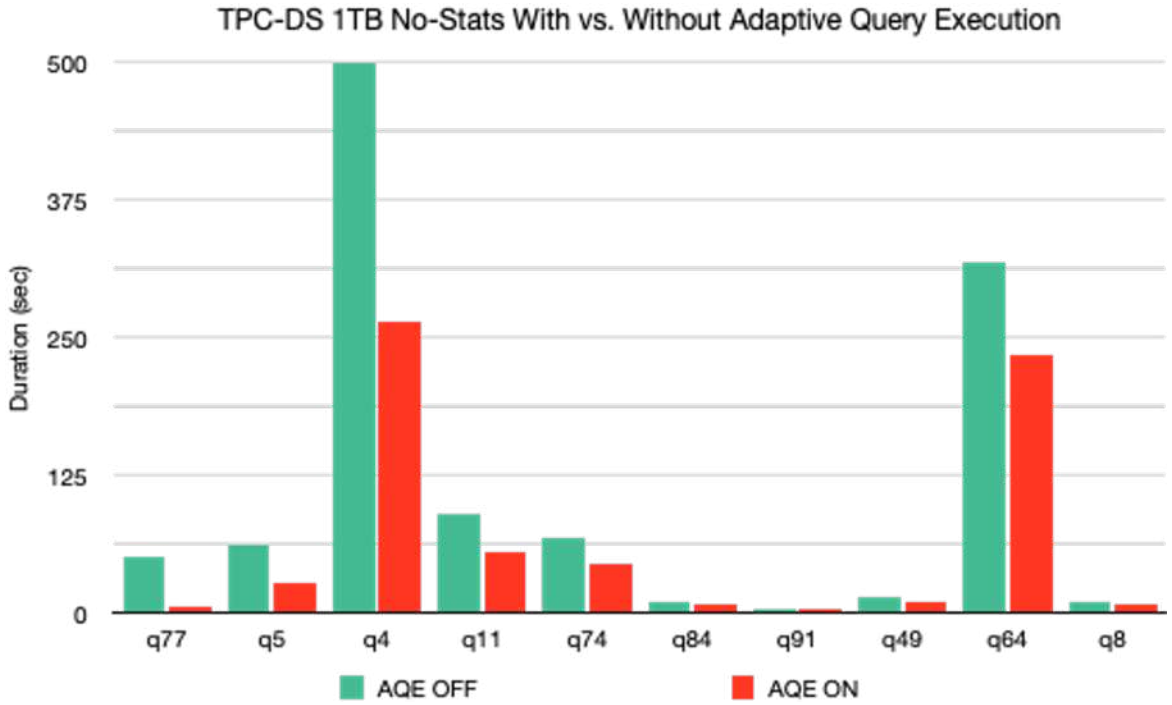


Table A - Part 0 – Split 0
Table B - Part 0
Table A - Part 0 – Split 1
Table B - Part 0
Table A - Part 0 – Split 2
Table B - Part 0
Table A - Part 1
Table B - Part 1
Table A - Part 2
Table B - Part 2
Table A - Part 3
Table B - Part 3
Sort
TabA.P0.S0 [Sorted]
TabB.P0.S0 [Sorted]
TabA.P0.S1 [Sorted]
TabB.P0.S1 [Sorted]
TabA.P0.S2 [Sorted]
TabB.P0.S1 [Sorted]
Table A - Part 1 [Sorted]
Table B - Part 1 [Sorted]
Table A - Part 2 [Sorted]
Table B - Part 2 [Sorted]
Table A - Part 3 [Sorted]
Table B - Part 3 [Sorted]
Merge-Join
Merge-Join
Merge-Join
Merge-Join
Merge-Join

Adaptive Query Execution



Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

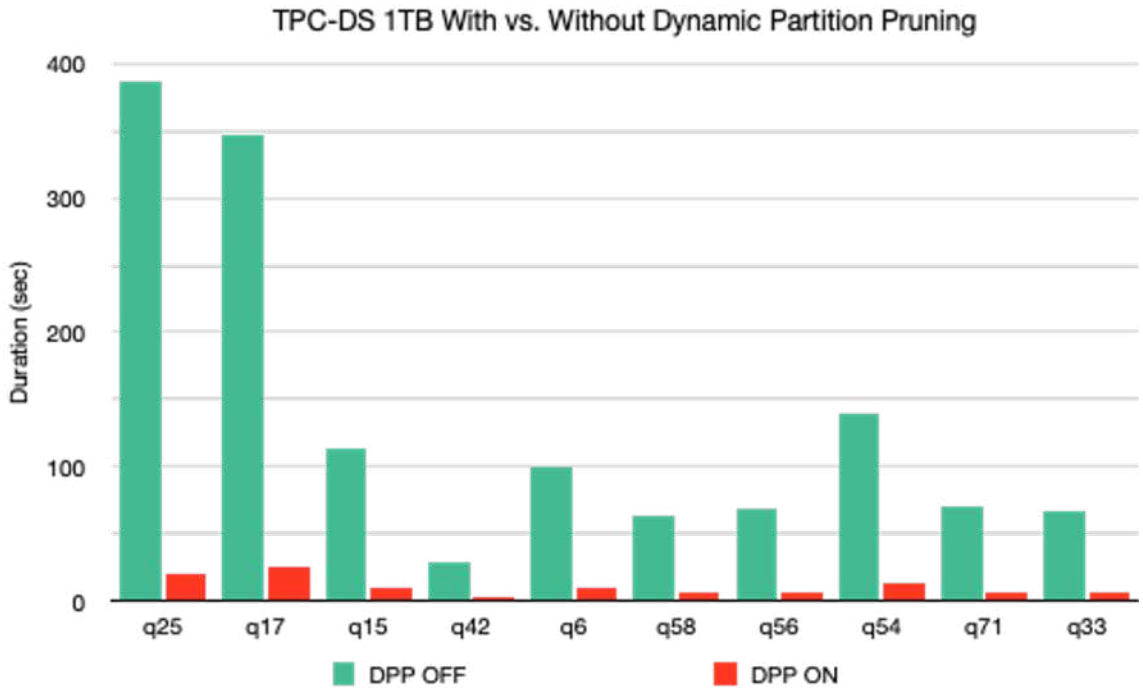
Join Hints

Achieve high performance for interactive, batch, streaming and ML workloads

Dynamic Partition Pruning

- Avoid partition scanning based on the query results of the other query fragments.
- Important for star-schema queries.
- Significant speedup in TPC-DS.

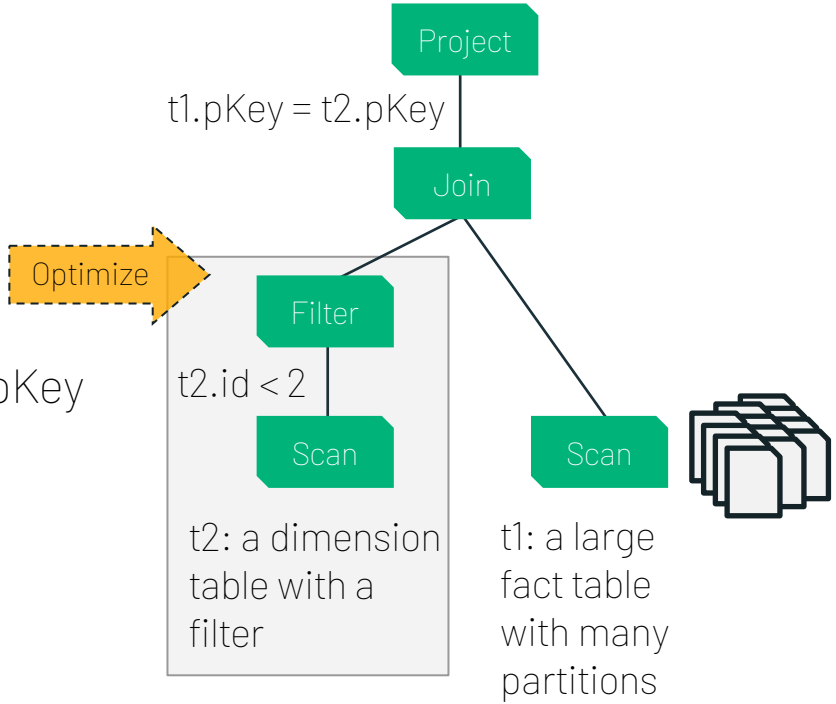
Dynamic Partition Pruning



60 / 102 TPC-DS queries: a speedup between 2x and 18x

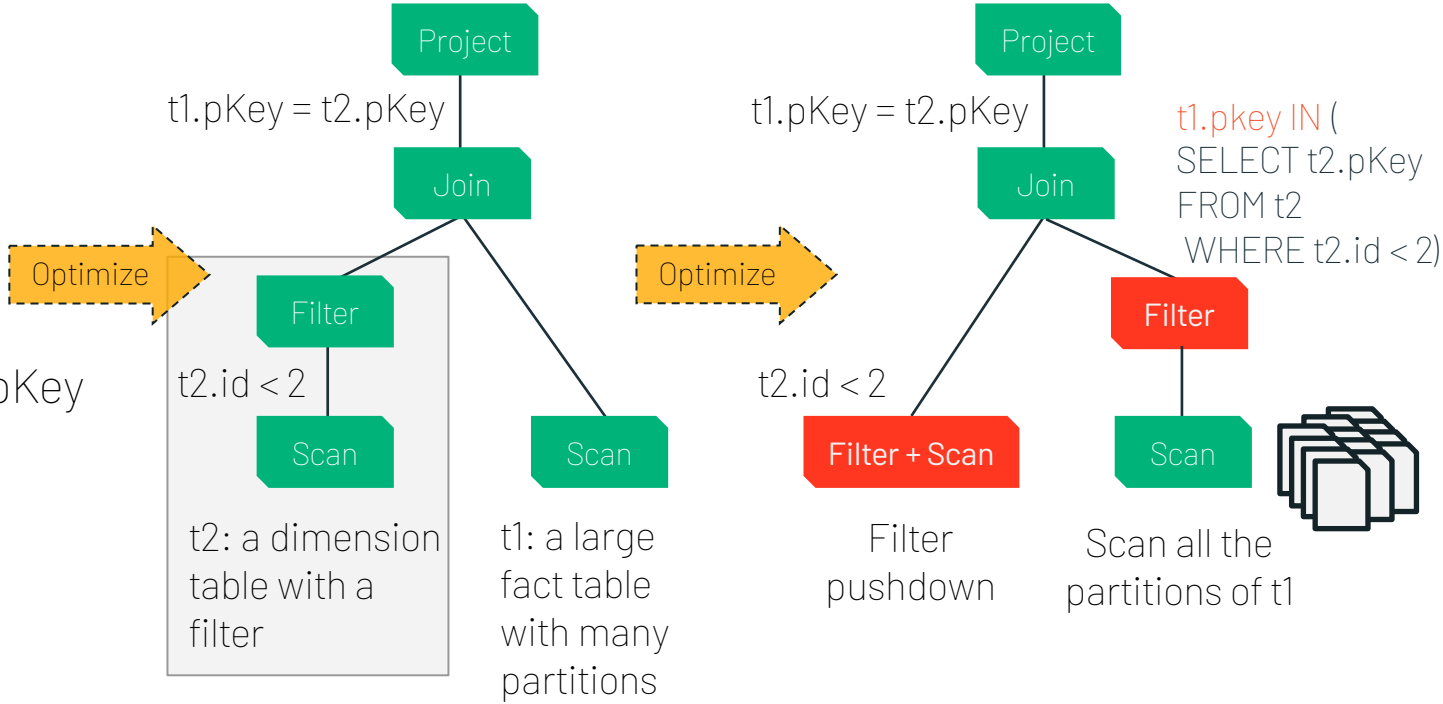
Dynamic Partition Pruning

```
SELECT t1.id, t2.pKey
  FROM t1
    JOIN t2
      ON t1.pKey = t2.pKey
    AND t2.id < 2
```

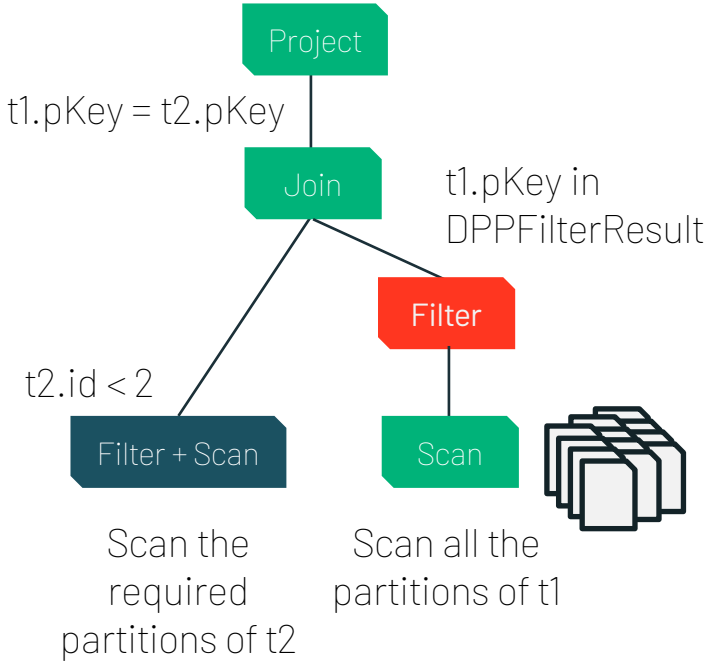


Dynamic Partition Pruning

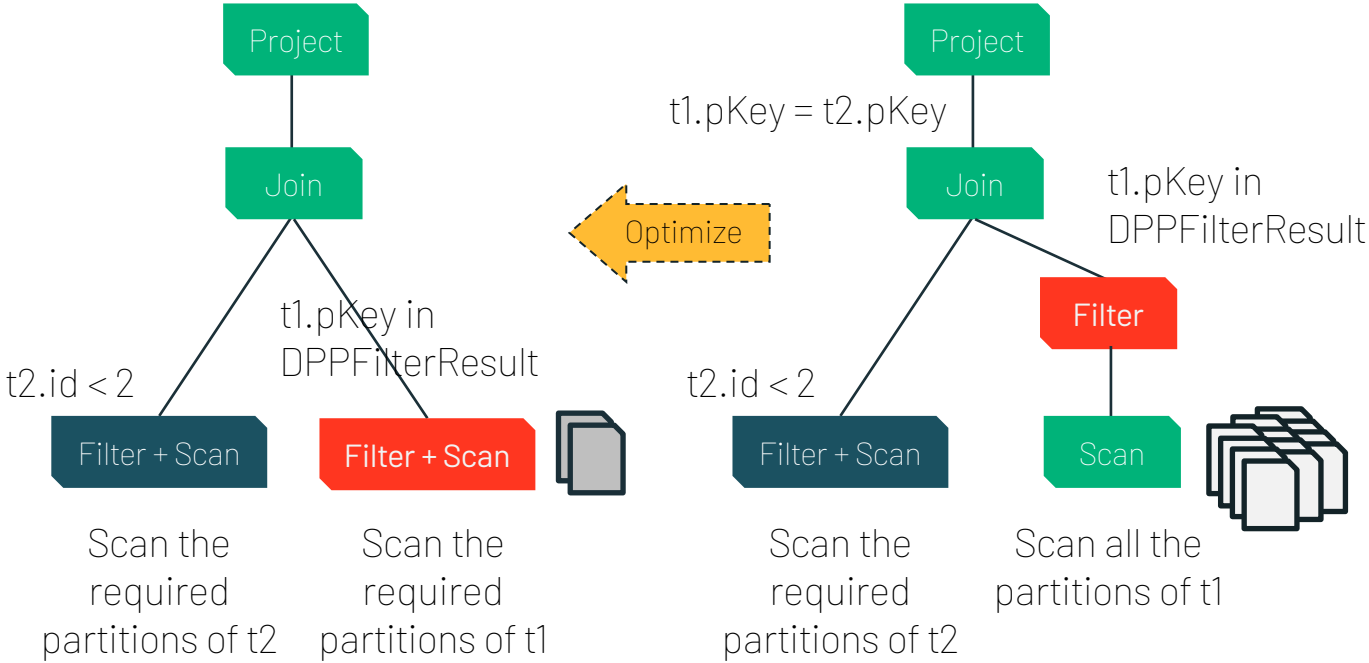
```
SELECT t1.id, t2.pKey
  FROM t1
    JOIN t2
      ON t1.pKey = t2.pKey
    AND t2.id < 2
```



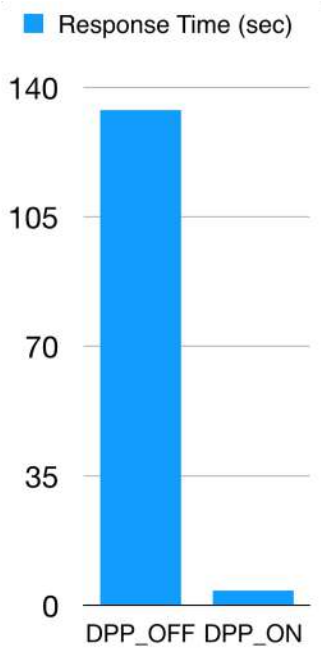
Dynamic Partition Pruning



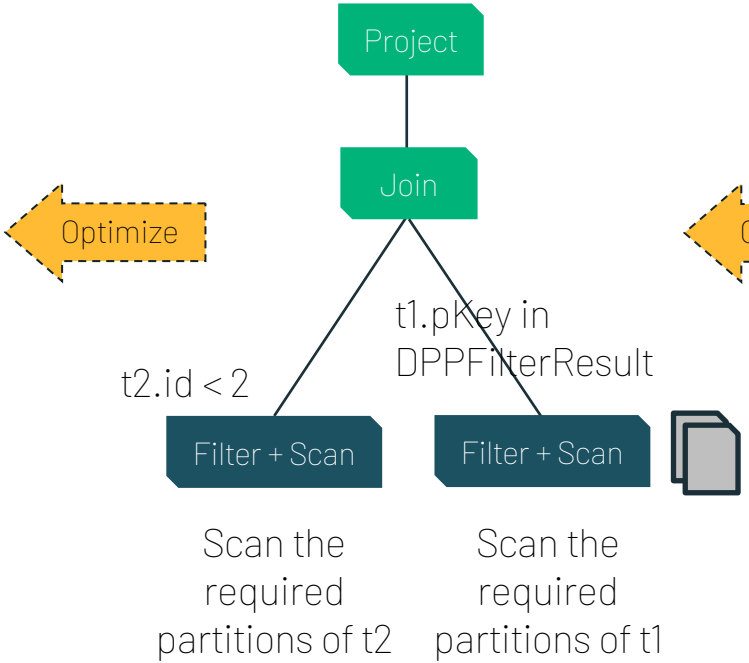
Dynamic Partition Pruning

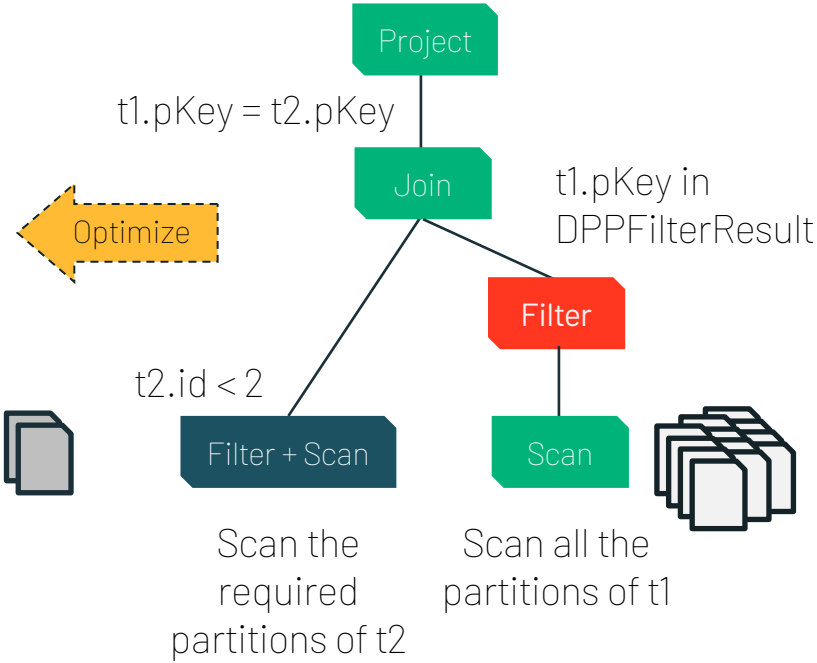


Dynamic Partition Pruning



33 X faster





90+% less file scan

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Achieve high performance for interactive, batch, streaming and ML workloads

Optimizer Hints

- Join hints influence optimizer to choose the join strategies
  - Broadcast hash join
  - Sort-merge join NEW
  - Shuffle hash join NEW
  - Shuffle nested loop join NEW
- Should be used with extreme caution.
  - Difficult to manage over time.

How to Use Join Hints?

- Broadcast Hash Join

  SELECT **/*+ BROADCAST(a) */** id FROM a JOIN b ON a.key = b.key

- Sort-Merge Join

  SELECT **/*+ MERGE(a, b) */** id FROM a JOIN b ON a.key = b.key

- Shuffle Hash Join

  SELECT **/*+ SHUFFLE_HASH(a, b) */** id FROM a JOIN b ON a.key = b.key

- Shuffle Nested Loop Join

  SELECT **/*+ SHUFFLE_REPLICATE_NL(a, b) */** id FROM a JOIN b

Broadcast Hash Join

Requires one side to be small. No shuffle, no sort, very fast.

Shuffle Hash Join

Needs to shuffle data but no sort. Can handle large tables, but will OOM too if data is skewed.

Sort-Merge Join

Robust. Can handle any data size. Needs to shuffle and sort data, slower in most cases when the table size is small.

Shuffle Nested Loop Join

Doesn't require join keys.

Richer APIs

Accelerator-aware Scheduler

Built-in Functions

pandas UDF enhancements

DELETE/UPDATE/ MERGE in Catalyst

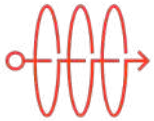
Enable new use cases and simplify the Spark application development

Richer APIs

Accelerator-aware Scheduler

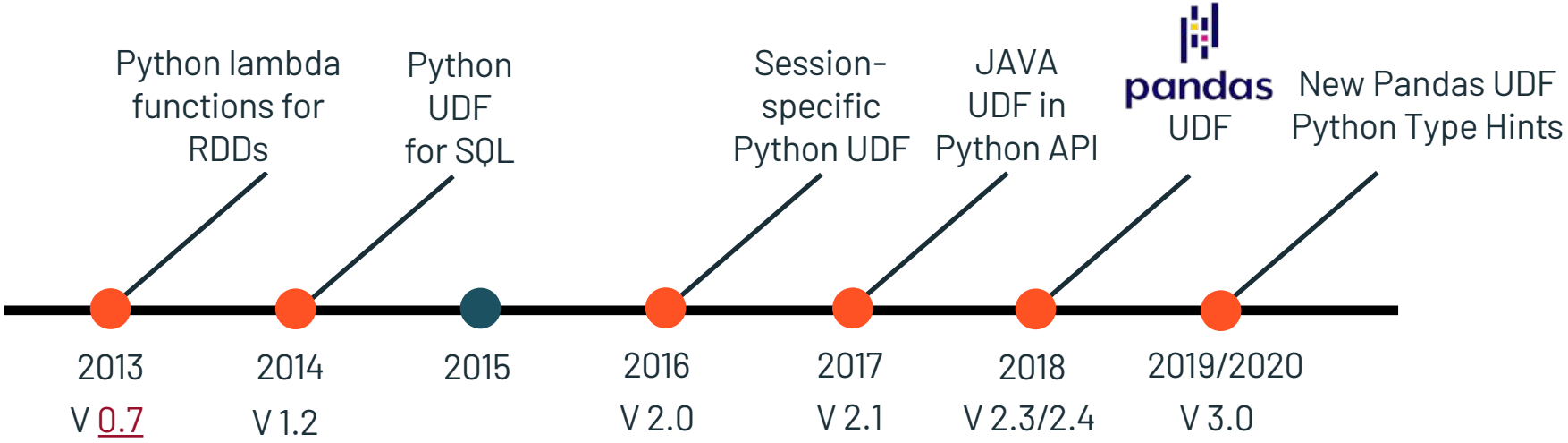
Built-in Functions

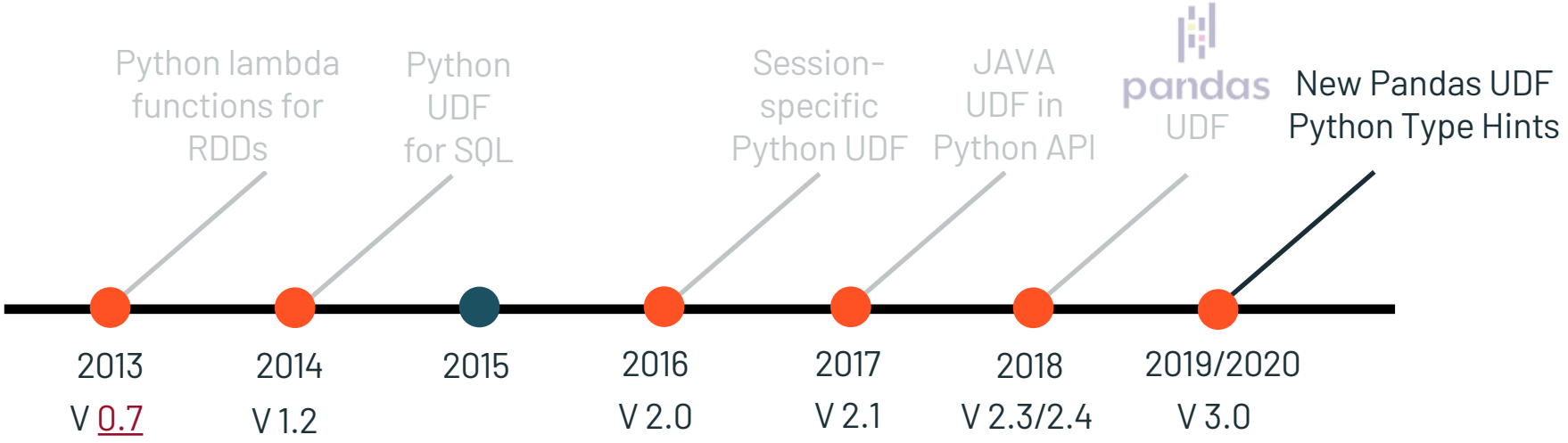
pandas UDF enhancements



DELETE/UPDATE/ MERGE in Catalyst

Enable new use cases and simplify the Spark application development


Python lambda functions for RDDs
Python UDF for SQL
Session-specific Python UDF
JAVA UDF in Python API
pandas UDF
New Pandas UDF Python Type Hints
2013
V 0.7
2014
V 1.2
2015
2016
V 2.0
2017
V 2.1
2018
V 2.3/2.4
2019/2020
V 3.0



Blog post: https://databricks.com/blog/2020/05/20/new-pandas-udfs-and-python-type-hints-in-the-upcoming-release-of-apache-spark-3-0.html

Scalar Pandas UDF
[pandas.Series to pandas.Series]

```
from pyspark.sql.functions import pandas_udf, PandasUDFType

@pandas_udf('long', PandasUDFType.SCALAR)
def pandas_plus_one(v):
    return v + 1

spark.range(10).select(pandas_plus_one("id")).show()
```

SPARK 2.3

```
import pandas as pd
from pyspark.sql.functions import pandas_udf

@pandas_udf('long')
def pandas_plus_one(s: pd.Series) -> pd.Series:
    return s + 1

spark.range(10).select(pandas_plus_one("id")).show()
```

SPARK 3.0

Python Type Hints

Grouped Map Pandas Function API
[pandas.DataFrame to pandas.DataFrame]

```python
import pandas as pd
from pyspark.sql.functions import pandas_udf, PandasUDFType

df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), (2, 3.0), (2, 5.0), (2, 10.0)], ("id", "v"))

@pandas_udf(df.schema, PandasUDFType.GROUPED_MAP)
def subtract_mean(pdf):
    v = pdf.v
    return pdf.assign(v=v - v.mean())

df.groupby("id").apply(subtract_mean).show()
```

SPARK 2.3

```python
import pandas as pd

df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), (2, 3.0), (2, 5.0), (2, 10.0)], ("id", "v"))

def subtract_mean(pdf: pd.DataFrame) -> pd.DataFrame:
    v = pdf.v
    return pdf.assign(v=v - v.mean())

df.groupby("id").applyInPandas(subtract_mean, schema=df.schema).show()
```

SPARK 3.0

Python Type Hints

Grouped Aggregate Pandas UDF
[pandas.Series to Scalar]

```python
import pandas as pd
from pyspark.sql.functions import pandas_udf, PandasUDFType
from pyspark.sql import Window

df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), (2, 3.0), (2, 5.0), (2, 10.0)], ("id", "v"))

@pandas_udf("double", PandasUDFType.GROUPED_AGG)
def pandas_mean(v):
    return v.sum()

df.select(pandas_mean(df['v'])).show()
df.groupby("id").agg(pandas_mean(df['v'])).show()
df.select(pandas_mean(df['v']).over(Window.partitionBy('id'))).show()
```

SPARK 2.4

```python
import pandas as pd
from pyspark.sql.functions import pandas_udf
from pyspark.sql import Window

df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), (2, 3.0), (2, 5.0), (2, 10.0)], ("id", "v"))

@pandas_udf("double")
def pandas_mean(v: pd.Series) -> float:
    return v.sum()

df.select(pandas_mean(df['v'])).show()
df.groupby("id").agg(pandas_mean(df['v'])).show()
df.select(pandas_mean(df['v']).over(Window.partitionBy('id'))).show()
```

SPARK 3.0

Python Type Hints

New Pandas UDF Types

```
@pandas_udf("long")
def calculate(iterator: Iterator[pd.Series]) -> Iterator[pd.Series]:
    # Do some expensive initialization with a state
    state = very_expensive_initialization()
    for x in iterator:
        # Use that state for the whole iterator.
        yield calculate_with_state(x, state)

df.select(calculate("value")).show()
```

```
from typing import Iterator, Tuple
import pandas as pd
from pyspark.sql.functions import pandas_udf

@pandas_udf("long")
def multiply_two(
        iterator: Iterator[Tuple[pd.Series, pd.Series]]) -> Iterator[pd.Series]:
    return (a * b for a, b in iterator)

spark.range(10).select(multiply_two("id", "id")).show()
```

New Pandas Function APIs

```
from typing import Iterator
import pandas as pd

df = spark.createDataFrame([(1, 21), (2, 30)], ("id", "age"))

def pandas_filter(iterator: Iterator[pd.DataFrame]) -> Iterator[pd.DataFrame]:
    for pdf in iterator:
        yield pdf[pdf.id == 1]

df.mapInPandas(pandas_filter, schema=df.schema).show()
```

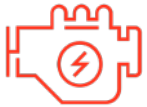
Map Pandas UDF

```
import pandas as pd

df1 = spark.createDataFrame(
    [(1201, 1, 1.0), (1201, 2, 2.0), (1202, 1, 3.0), (1202, 2, 4.0)],
    ("time", "id", "v1"))
df2 = spark.createDataFrame(
    [(1201, 1, "x"), (1201, 2, "y")], ("time", "id", "v2"))

def asof_join(left: pd.DataFrame, right: pd.DataFrame) -> pd.DataFrame:
    return pd.merge_asof(left, right, on="time", by="id")

df1.groupby("id").cogroup(
    df2.groupby("id")
).applyInPandas(asof_join, "time int, id int, v1 double, v2 string").show()
```

Cogrouped Map Pandas UDF

Richer APIs

Accelerator-aware Scheduler



Built-in Functions

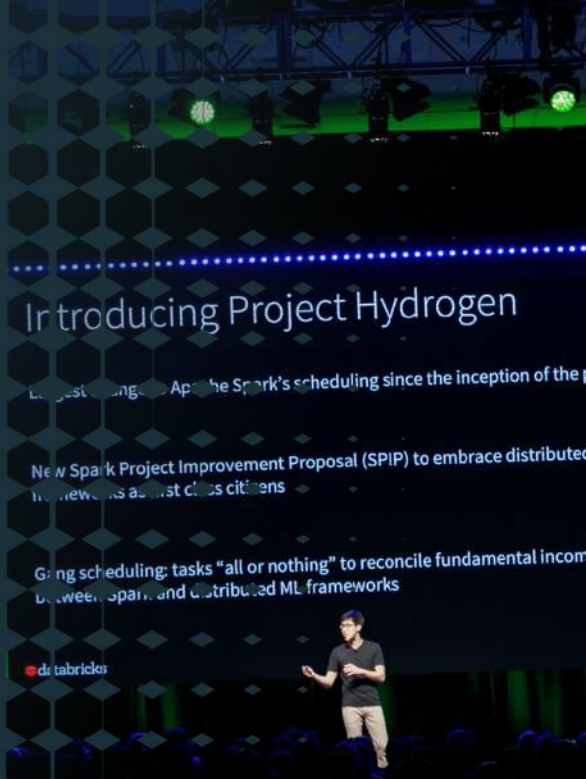
pandas UDF enhancements

DELETE/UPDATE/ MERGE in Catalyst

Enable new use cases and simplify the Spark application development

Accelerator-aware Scheduling

- Widely used for accelerating special workloads, e.g., deep learning and signal processing.
- Supports Standalone, YARN and K8S.
- Supports GPU now, FPGA, TPU, etc. in the future.
- Needs to specify required resources by configs
- Application level. Will support job/stage/task level in the future.



The workflow

User	Spark	Cluster Manager
		0. Auto-discover resources.
1. Submit an application with resource requests.	2. Pass resource requests to cluster manager. 4. Register executors.	3. Allocate executors with resource isolation.
5. Submit a Spark job.	6. Schedule tasks on available executors.	7. Dynamic allocation.
8. Retrieve assigned resources and use them in tasks.		9. Monitor and recover failed executors.

Discover and request accelerators

Admin can specify a script to auto-discover accelerators (SPARK-27024)

- spark.driver.resource.${resourceName}.discoveryScript
- spark.executor.resource.${resourceName}.discoveryScript
- e.g., `nvidia-smi --query-gpu=index ...`

User can request accelerators at application level (SPARK-27366)

- spark.executor.resource.${resourceName}.amount
- spark.driver.resource.${resourceName}.amount
- spark.task.resource.${resourceName}.amount

Retrieve assigned accelerators

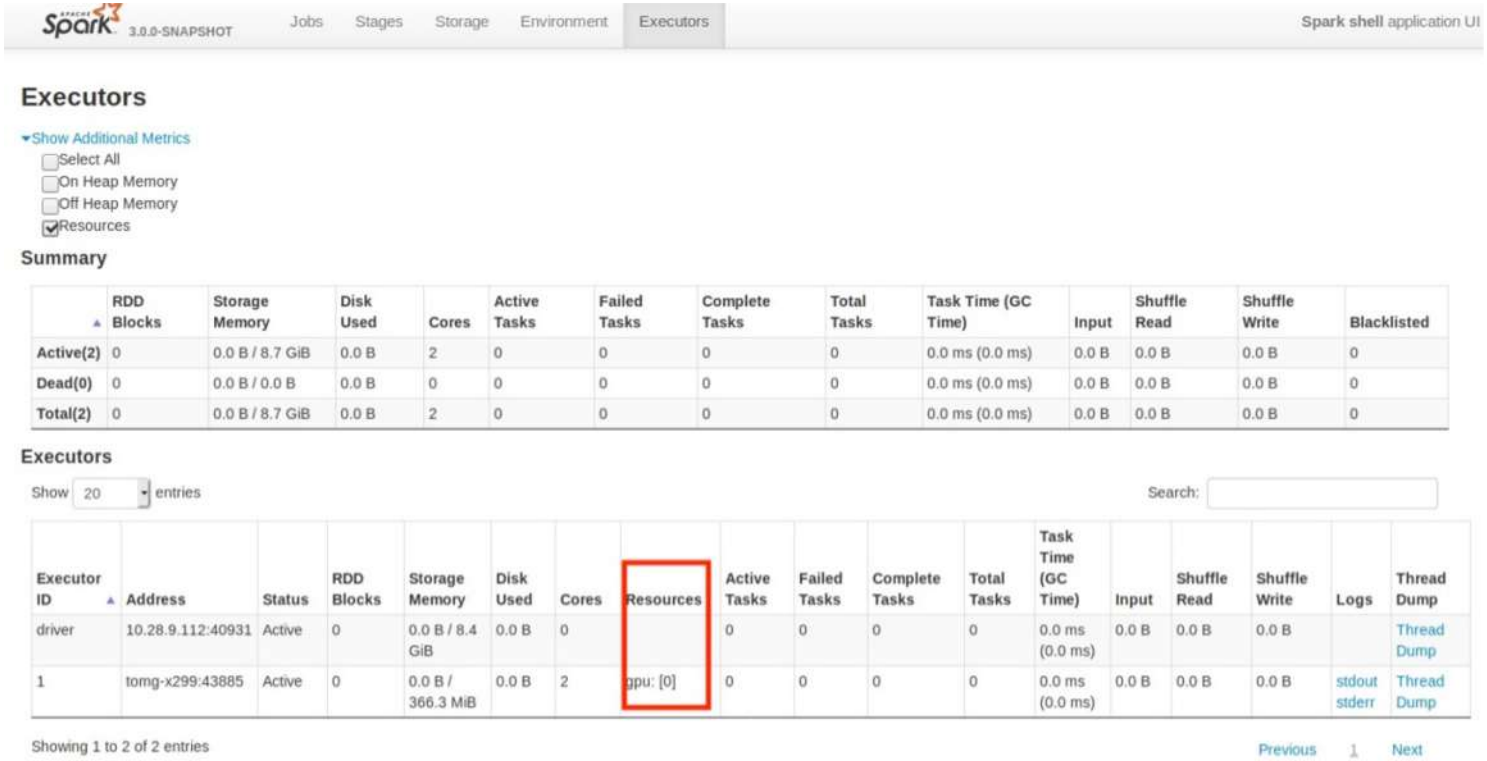
User can retrieve assigned accelerators from task context (SPARK-27366)

```
context = TaskContext.get()
assigned_gpu =
context.resources().get(“gpu”).get.addresses.head

with tf.device(assigned_gpu):
  # training code ...
```

Cluster manager support

Standalone SPARK-27360	YARN SPARK-27361
Kubernetes SPARK-27362	Mesos (not started) SPARK-27363

Web UI for accelerators



databricks

Richer APIs

Accelerator-aware Scheduler

Built-in Functions

pandas UDF enhancements

DELETE/UPDATE/ MERGE in Catalyst

Enable new use cases and simplify the Spark application development

32 New Built-in Functions

Cmd 1

```
1 SELECT map_keys(map(1, 'a', 2, 'b'))
```

▸ (1) Spark Jobs

map_keys(map(1, a, 2, b))

▾ array
- 0: 1
- 1: 2

Command took 0.13 seconds -- by wenchen@databricks.com at 5/29/2020, 1:52:49 PM on DBR 7.0 Shared Autoscaling

Cmd 2

```
1 SELECT map_values(map(1, 'a', 2, 'b'))
```

▸ (1) Spark Jobs

map_values(map(1, a, 2, b))

▾ array
- 0: a
- 1: b

Command took 0.06 seconds -- by wenchen@databricks.com at 5/29/2020, 1:53:01 PM on DBR 7.0 Shared Autoscaling

Cmd 3

```
1 SELECT transform_keys(map(1, 1, 2, 2, 3, 3), (k, v) -> k + 1)
```

▸ (1) Spark Jobs

transform_keys(map(1, 1, 2, 2, 3, 3), lambdafunction((namedlambdavariable() + 1), namedlambdavariable(), namedlambdavariable()))

▾ object
- 2: 1
- 3: 2
- 4: 3

Command took 0.14 seconds -- by wenchen@databricks.com at 5/29/2020, 1:53:11 PM on DBR 7.0 Shared Autoscaling

Cmd 4

```
1 SELECT transform_values(map(1, 1, 2, 2, 3, 3), (k, v) -> k + 1)
```

▸ (1) Spark Jobs

transform_values(map(1, 1, 2, 2, 3, 3), lambdafunction((namedlambdavariable() + 1), namedlambdavariable(), namedlambdavariable()))

▾ object
- 1: 2
- 2: 3
- 3: 4

Command took 0.15 seconds -- by wenchen@databricks.com at 5/29/2020, 1:53:26 PM on DBR 7.0 Shared Autoscaling

Cmd 10

```
1 display(sql("select map(1, 1, 2, 2, 3, 3) as m").select(transform_keys($"m", (key, value) => key + 1)))
```

▸ (1) Spark Jobs

transform_keys(m, lambdafunction((x + 1), x, y))

▾ object
- 2: 1
- 3: 2
- 4: 3

Command took 0.57 seconds -- by wenchen@databricks.com at 6/1/2020, 11:09:52 AM on DBR 7.0 Shared Autoscaling

Cmd 11

```
1 display(sql("select map(1, 1, 2, 2, 3, 3) as m").select(transform_values($"m", (key, value) => key + 1)))
```

▸ (1) Spark Jobs

transform_values(m, lambdafunction((x + 1), x, y))

▾ object
- 1: 2
- 2: 3
- 3: 4

Command took 0.54 seconds -- by wenchen@databricks.com at 6/1/2020, 11:10:23 AM on DBR 7.0 Shared Autoscaling

Monitoring and Debuggability

Structured Streaming UI

DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Make monitoring and debugging Spark applications more comprehensive and stable

Monitoring and Debuggability

Structured Streaming UI

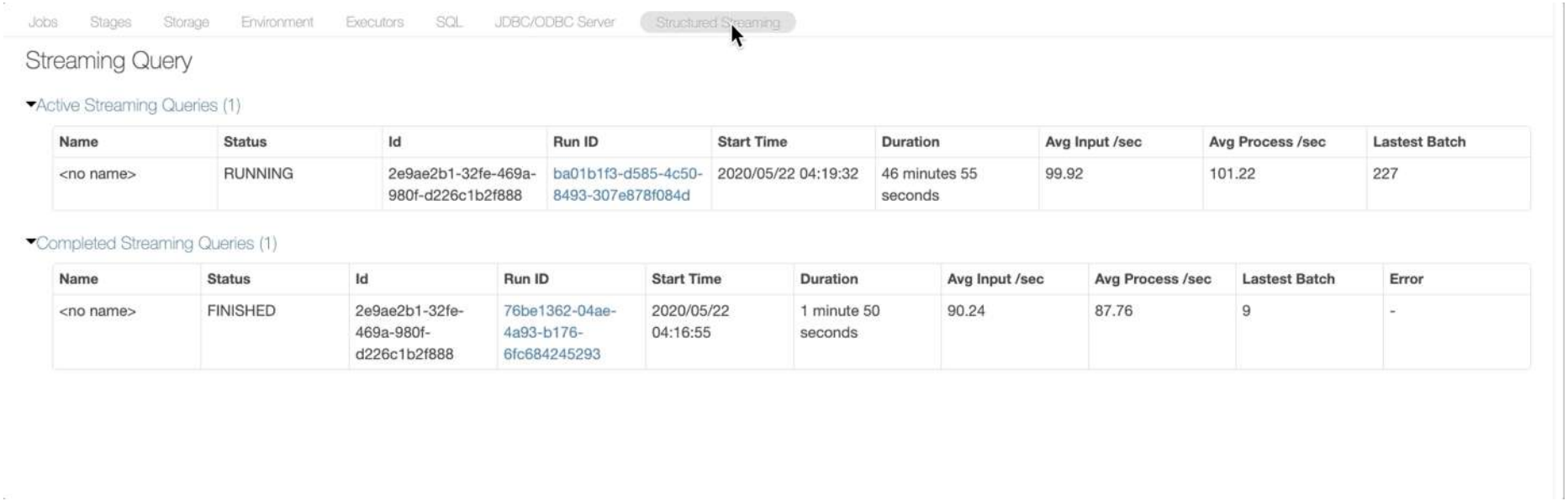
DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Make monitoring and debugging Spark applications more comprehensive and stable

Structured Streaming UI



Jobs Stages Storage Environment Executors SQL JDBC/ODBC Server Structured Streaming

Streaming Query

▾Active Streaming Queries (1)

Name	Status	Id	Run ID	Start Time	Duration	Avg Input /sec	Avg Process /sec	Lastest Batch
<no name>	RUNNING	2e9ae2b1-32fe-469a-980f-d226c1b2f888	ba01b1f3-d585-4c50-8493-307e878f084d	2020/05/22 04:19:32	46 minutes 55 seconds	99.92	101.22	227

▾Completed Streaming Queries (1)

Name	Status	Id	Run ID	Start Time	Duration	Avg Input /sec	Avg Process /sec	Lastest Batch	Error
<no name>	FINISHED	2e9ae2b1-32fe-469a-980f-d226c1b2f888	76be1362-04ae-4a93-b176-6fc684245293	2020/05/22 04:16:55	1 minute 50 seconds	90.24	87.76	9	-

Monitoring and Debuggability

Structured Streaming UI

DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Make monitoring and debugging Spark applications more comprehensive and stable

New Command EXPLAIN FORMATTED

```
*(1) Project [key#5, val#6]
+- *(1) Filter (isnotnull(key#5) AND (key#5 = Subquery scalar-subquery#15, [id=#113]))
   :  +- Subquery scalar-subquery#15, [id=#113]
   :     +- *(2) HashAggregate(keys=[], functions=[max(key#21)])
   :        +- Exchange SinglePartition, true, [id=#109]
   :           +- *(1) HashAggregate(keys=[], functions=[partial_max(key#21)])
   :              +- *(1) Project [key#21]
   :                 +- *(1) Filter (isnotnull(val#22) AND (val#22 > 5))
   :                    +- *(1) ColumnarToRow
   :                       +- FileScan parquet default.tab2[key#21,val#22] Batched: true, DataFilters: [isnotnull(val#22), (val#22 > 5)], Format: Parquet, Location: InMemoryFileIndex[file:/user/hive/warehouse/tab2], PartitionFilters: [], PushedFilters: [IsNotNull(val), GreaterThan(val,5)], ReadSchema: struct<key:int,val:int>
   +- *(1) ColumnarToRow
      +- FileScan parquet default.tab1[key#5,val#6] Batched: true, DataFilters: [isnotnull(key#5)], Format: Parquet, Location: InMemoryFileIndex[file:/user/hive/warehouse/tab1], PartitionFilters: [], PushedFilters: [IsNotNull(key)], ReadSchema: struct<key:int,val:int>
```

```
* Project (4)
+- * Filter (3)
   +- * ColumnarToRow (2)
      +- Scan parquet default.tab1 (1)

(1) Scan parquet default.tab1
Output [2]: [key#5, val#6]
Batched: true
Location: InMemoryFileIndex [file:/user/hive/warehouse/tab1]
PushedFilters: [IsNotNull(key)]
ReadSchema: struct<key:int,val:int>

(2) ColumnarToRow [codegen id : 1]
Input [2]: [key#5, val#6]

(3) Filter [codegen id : 1]
Input [2]: [key#5, val#6]
Condition : (isnotnull(key#5) AND (key#5 = Subquery scalar-subquery#27, [id=#164]))

(4) Project [codegen id : 1]
Output [2]: [key#5, val#6]
Input [2]: [key#5, val#6]
```

```
EXPLAIN FORMATTED
 SELECT *
 FROM  tab1
 WHERE  key = (SELECT max(key)
          FROM  tab2
          WHERE  val > 5
```

===== Subqueries =====

Subquery:1 Hosting operator id = 3 Hosting Expression = Subquery **scalar-subquery#27**, [id=#164]

```
* HashAggregate (11)
+- Exchange (10)
  +- * HashAggregate (9)
    +- * Project (8)
      +- * Filter (7)
        +- * ColumnarToRow (6)
          +- Scan parquet default.tab2 (5)
```

(5) Scan parquet default.tab2
Output [2]: [key#21, val#22]
Batched: true
Location: InMemoryFileIndex [file:/user/hive/warehouse/tab2]
PushedFilters: [IsNotNull(val), GreaterThan(val,5)]
ReadSchema: struct<key:int,val:int>

(6) ColumnarToRow [codegen id : 1]
Input [2]: [key#21, val#22]

(7) Filter [codegen id : 1]
Input [2]: [key#21, val#22]
Condition : (isnotnull(val#22) AND (val#22 > 5))

(8) Project [codegen id : 1]
Output [1]: [key#21]
Input [2]: [key#21, val#22]

(9) HashAggregate [codegen id : 1]
Input [1]: [key#21]
Keys: []
Functions [1]: [partial_max(key#21)]
Aggregate Attributes [1]: [max#35]
Results [1]: [max#36]

(10) Exchange
Input [1]: [max#36]
Arguments: SinglePartition, true, [id=#160]

(11) HashAggregate [codegen id : 2]
Input [1]: [max#36]
Keys: []
Functions [1]: [max(key#21)]
Aggregate Attributes [1]: [max(key#21)#33]
Results [1]: [max(key#21)#33 AS max(key)#34]

Monitoring and Debuggability

Structured Streaming UI

DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Make monitoring and debugging Spark applications more comprehensive and stable

Observable Metrics

A flexible way to monitor data quality.

```scala
val stream = spark.readStream...

stream.observe("data_quality", count($"error") / count(lit(1))).writeStream...

spark.streams.addListener(new StreamingQueryListener() {
  override def onQueryProgress(event: QueryProgressEvent): Unit = {
    event.progress.observedMetrics.get("data_quality").foreach {
      case Row(pct_parse_errors: Double) if pct_parse_errors > 0.05 => // Trigger alert
      case _ => // OK
    }
  }
})
```

SQL Compatibility

ANSI Store Assignment

Overflow Checking

Reserved Keywords in Parser

Proleptic Gregorian Calendar

Reduce the time and complexity of enabling applications that were written for other relational database products to run in Spark SQL.

SQL Compatibility

ANSI Store Assignment

Overflow Checking

Reserved Keywords in Parser

Proleptic Gregorian Calendar

Reduce the time and complexity of enabling applications that were written for other relational database products to run in Spark SQL.

ANSI store assignment + overflow check

A safer way to do table insertion and avoid bad data.

Cmd 7

```
1  CREATE TABLE ansi_tbl(i INT, j STRING) USING parquet
```

OK

Command took 0.27 seconds -- by wenchen@databricks.com at 5/29/2020, 2:14:10 PM on DBR 7.0 Shared Autoscaling

Cmd 8

```
1  -- write int value to string column is safe
2  INSERT INTO ansi_tbl VALUES (1, 1)
```

▸ (1) Spark Jobs

OK

Command took 2.75 seconds -- by wenchen@databricks.com at 5/29/2020, 2:16:42 PM on DBR 7.0 Shared Autoscaling

Cmd 9

```
1  -- write string value to int column is not safe
2  INSERT INTO ansi_tbl VALUES ("1", "1")
```

⊞Error in SQL statement: AnalysisException: Cannot write incompatible data to table ``default`.`ansi_tbl``:
- Cannot safely cast 'i': StringType to IntegerType;

Command took 0.31 seconds -- by wenchen@databricks.com at 5/29/2020, 2:16:50 PM on DBR 7.0 Shared Autoscaling

ANSI store assignment + overflow check

A safer way to do table insertion and avoid bad data.

Cmd 10

```
1  -- write long value to int column is OK if no overflow
2  INSERT INTO ansi_tbl VALUES (1L, "1")
```

▸ (1) Spark Jobs

OK

Command took 2.21 seconds -- by wenchen@databricks.com at 5/29/2020, 2:16:58 PM on DBR 7.0 Shared Autoscaling

Cmd 11

```
1  -- fail at runtime if overflow happens
2  INSERT INTO ansi_tbl VALUES (12345678912345L, "1")
```

⊞Error in SQL statement: ArithmeticException: Casting 12345678912345 to int causes overflow

Command took 0.33 seconds -- by wenchen@databricks.com at 5/29/2020, 2:17:08 PM on DBR 7.0 Shared Autoscaling

Built-in Data Sources

Parquet/ORC Nested Column Pruning

CSV Filter Pushdown

Parquet: Nested Column Filter Pushdown

New Binary Data Source

Enhance the performance and functionalities of the built-in data sources

Built-in Data Sources

Parquet/ORC Nested Column Pruning

CSV Filter Pushdown

Parquet: Nested Column Filter Pushdown

New Binary Data Source

Enhance the performance and functionalities of the built-in data sources

Better performance for nested fields

- Skip reading useless data blocks when only a few inner fields are selected.

```
1  spark.read.table("nested").select("col.a").explain()
```

```
== Physical Plan ==
*(1) Project [col#33245196.a AS a#33253077]
+- *(1) ColumnarToRow
   +- FileScan parquet default.nested[col#33245196] Batched: true, DataFilters: [],
s: [], ReadSchema: struct<col:struct<a:int>>
```

Better performance for nested fields

- Skip reading useless data blocks when there are predicates with inner fields.

```
1  spark.read.table("nested").filter($"col.a" > 0).explain()
```

```
== Physical Plan ==
*(1) Project [col#33245196]
+- *(1) Filter (isnotnull(col#33245196) AND (col#33245196.a > 0))
   +- *(1) ColumnarToRow
      +- FileScan parquet default.nested[col#33245196] Batched: true, DataFilters: [isnotnu
ouse/nested], PartitionFilters: [], PushedFilters: [IsNotNull(col), GreaterThan(col.a,0)],
```

Extensibility and Ecosystem

Data Source V2 API + Catalog Support

Java 11 Support

Hadoop 3 Support

Hive 3.x Metastore
Hive 2.3 Execution

Improve the plug-in interface and extend the deployment environments

Extensibility and Ecosystem

Data Source V2 API + Catalog Support

Java 11 Support

Hadoop 3 Support

Hive 3.x Metastore
Hive 2.3 Execution

Improve the plug-in interface and extend the deployment environments

Catalog plugin API

Users can register customized catalogs and use Spark to access/manipulate table metadata directly.

```
-- Assume a MySQL connector is registered as catalog named "mysql"
SELECT * FROM mysql.db1.t1;
INSERT INTO mysql.db2.t2 SELECT * FROM temp_view;
CREATE TABLE mysql.db1.t3(i INT, j STRING);
ALTER TABLE mysql.db1.t3 ADD COLUMN k INT;
```

JDBC data source v2 is coming in Spark 3.1

To developers: When to support Data Source V2?

- Pick V2 if you want to provide catalog functionalities, as V1 doesn't have such ability.
- Pick V2 If you want to support both batch and streaming, as V1 uses separated APIs for batch and streaming which makes it hard to reuse code.
- Pick V2 if you are sensitive to the scan performance, as V2 allows you to report data partitioning to skip shuffle, and implement vectorized reader for better performance.

Data source v2 API is not as stable as V1!

Extensibility and Ecosystem

Data Source V2 API + Catalog Support

Java 11 Support

Hadoop 3 Support

Hive 3.x Metastore
Hive 2.3 Execution

Improve the plug-in interface and extend the deployment environments

Spark 3.0 Builds

- Only builds with Scala 2.12
- Deprecates Python 2 (already EOL)
- Can build with various Hadoop/Hive versions
  - Hadoop 2.7 + Hive 1.2
  - Hadoop 2.7 + Hive 2.3 (supports Java 11) [Default]
  - Hadoop 3.2 + Hive 2.3 (supports Java 11)
- Supports the following Hive metastore versions:
  - "0.12", "0.13", "0.14", "1.0", "1.1", "1.2", "2.0", "2.1", "2.2", "2.3", "3.0", "3.1"

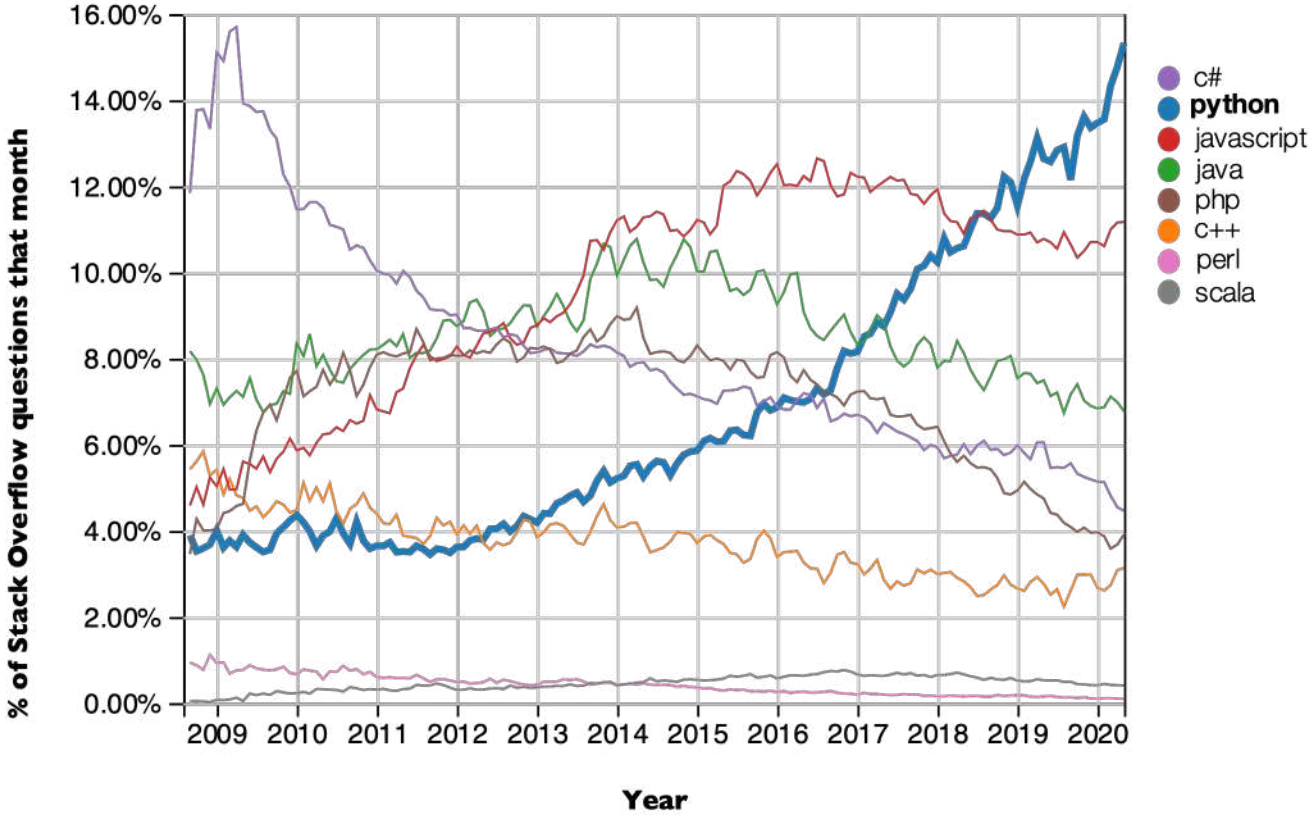
Extensibility and Ecosystem

Koalas

潘石屹

我刚开始学Python的时候，朋友就问我，你是要做码农了吗？其实并不是做码农才要学编程，不管干什么都要有科学意识，同时因为世界变化太快，掌握计算机的语言已经变成了一个必须具备的能力。#潘石屹学Python# 潘石屹的微博视频

05月24日 09:50 来自 微博 weibo.com

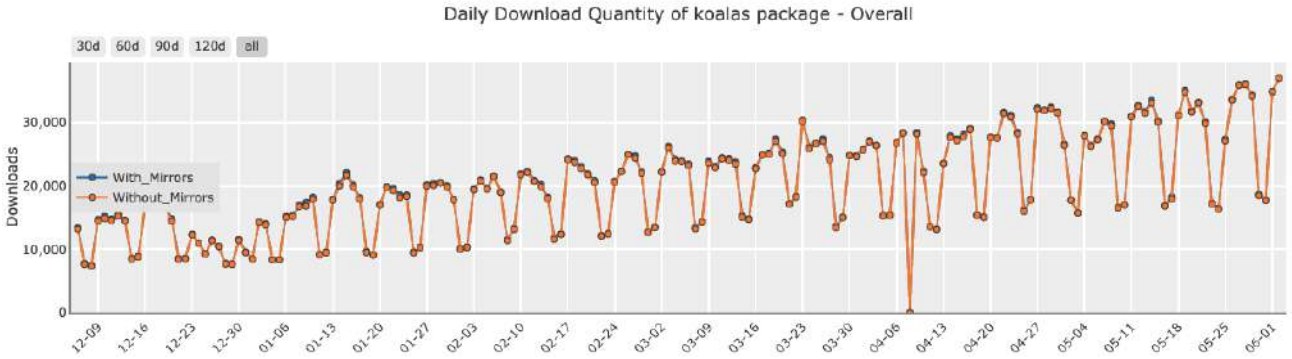

16.00%
14.00%
12.00%
10.00%
8.00%
6.00%
4.00%
2.00%
0.00%
% of Stack Overflow questions that month
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Year
c#
python
javascript
java
php
c++
perl
scala

Koalas

- Announced April 24, 2019
- Pure Python library
- Aims at providing the pandas API on top of Spark.
- Seamless transition between small and large data

Koalas

- Unifies the two ecosystems with a familiar API
- pandas users:
  - scale out the pandas code using Koalas
  - make learning PySpark much easier
- Spark users:
  - more productive by pandas-like functions



- pip install koalas
- conda install koalas
- Docs and updates on github.com/databricks/koalas
- Project docs are published on koalas.readthedocs.io

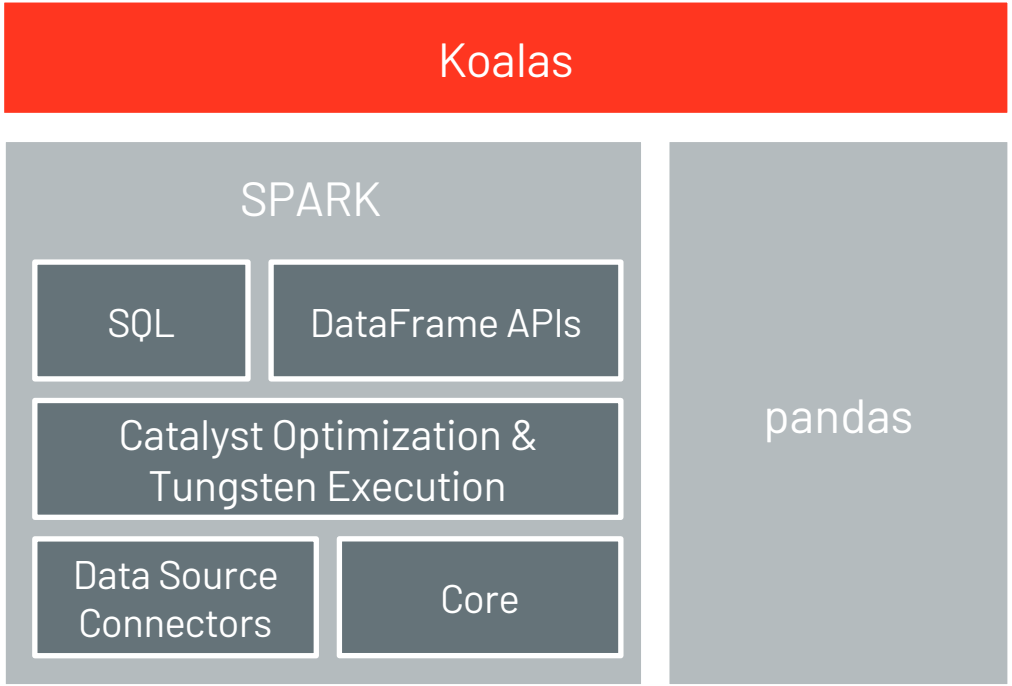
37,000+
Downloads per day

845,223
Downloads this May

2100+
GitHub Stars

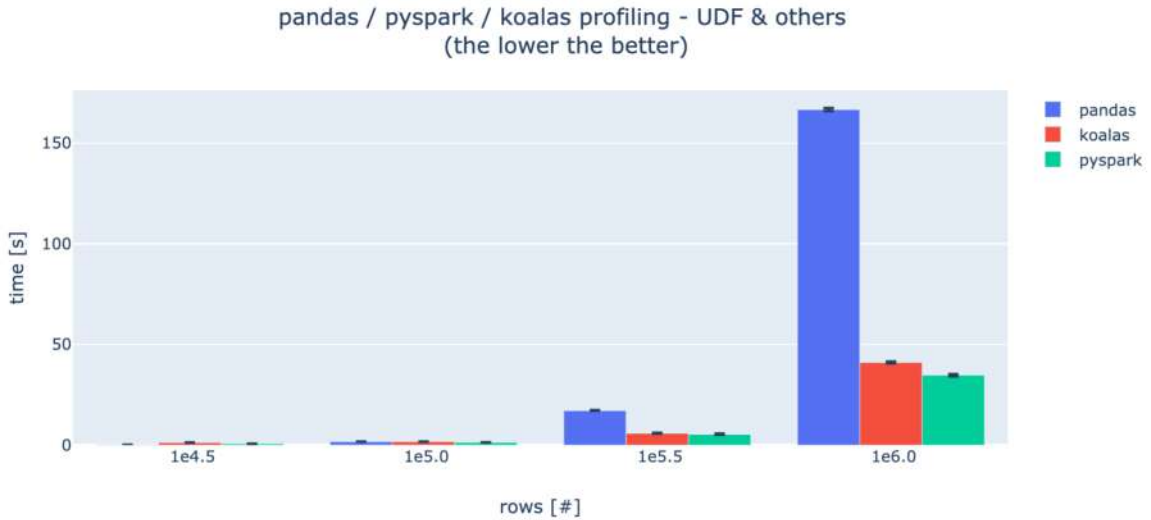
33
Bi-weekly releases

Architecture



A lean API layer

How Virgin Hyperloop One reduced processing time from hours to minutes with Koalas



Blog post : https://t.co/2cZ4m5ymGo
Webinar: https://youtu.be/hYvHg2PwUlc

Challenge: increasing scale and complexity of data operations

Struggling with the "Spark switch" from pandas

More than 10X faster with less than 1% code changes

Documentation

- Web UI
- SQL reference
- Migration guide
- Semantic versioning guidelines

Spark Overview

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

Security

Security in Spark is OFF by default. This could mean you are vulnerable to attack by default. Please see Spark Security before downloading and running Spark.

Downloading

Get Spark from the downloads page of the project website. This documentation is for Spark version 3.0.0. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and run Spark with any Hadoop version by augmenting Spark's classpath. Scala and Java users can include Spark in their projects using its Maven coordinates and Python users can install Spark from PyPI.

If you'd like to build Spark from source, visit Building Spark.

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS), and it should run on any platform that runs a supported version of Java. This should include JVMs on x86_64 and ARM64. It's easy to run locally on one machine — all you need is to have `java` installed on your system `PATH`, or the `JAVA_HOME` environment variable pointing to a Java installation.

Spark runs on Java 8/11, Scala 2.12, Python 2.7+/3.4+ and R 3.1+. Java 8 prior to version 8u92 support is deprecated as of Spark 3.0.0. Python 2 and Python 3 prior to version 3.6 support is deprecated as of Spark 3.0.0. R prior to version 3.4 support is deprecated as of Spark 3.0.0. For the Scala API, Spark 3.0.0 uses Scala 2.12. You will need to use a compatible Scala version (2.12.x).

For Java 11, `-Dio.netty.tryReflectionSetAccessible=true` is required additionally for Apache Arrow library. This prevents `java.lang.UnsupportedOperationException: sun.misc.Unsafe or java.nio.DirectByteBuffer.(long, int) not available` when Apache Arrow uses Netty internally.

Running the Examples and Shell

Spark Overview

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

Security

Security in Spark is OFF by default. This could mean you are vulnerable to attack by default. Please see Spark Security before downloading and running Spark.

Downloading

Get Spark from the downloads page of the project website. This documentation is for Spark version 3.0.1-SNAPSHOT. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and run Spark with any Hadoop version by augmenting Spark's classpath. Scala and Java users can include Spark in their projects using its Maven coordinates and Python users can install Spark from PyPI.

If you'd like to build Spark from source, visit Building Spark.

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS), and it should run on any platform that runs a supported version of Java. This should include JVMs on x86_64 and ARM64. It's easy to run locally on one machine — all you need is to have `java` installed on your system `PATH`, or the `JAVA_HOME` environment variable pointing to a Java installation.

Spark runs on Java 8/11, Scala 2.12, Python 2.7+/3.4+ and R 3.1+. Java 8 prior to version 8u92 support is deprecated as of Spark 3.0.0. Python 2 and Python 3 prior to version 3.6 support is deprecated as of Spark 3.0.0. R prior to version 3.4 support is deprecated as of Spark 3.0.0. For the Scala API, Spark 3.0.1-SNAPSHOT uses Scala 2.12. You will need to use a compatible Scala version (2.12.x).

For Java 11, `-Dio.netty.tryReflectionSetAccessible=true` is required additionally for Apache Arrow library. This prevents `java.lang.UnsupportedOperationException: sun.misc.Unsafe or java.nio.DirectByteBuffer.(long, int) not available` when Apache Arrow uses Netty internally.

Running the Examples and Shell

SQL Reference

Spark SQL is Apache Spark's module for working with structured data. This guide is a reference for Structured Query Language (SQL) and includes syntax, semantics, keywords, and examples for common SQL usage. It contains information for the following topics:

Spark Overview

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

Security

Security in Spark is OFF by default. This could mean you are vulnerable to attack by default. Please see Spark Security before downloading and running Spark.

Downloading

Get Spark from the downloads page of the project website. This documentation is for Spark version 3.0.0. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and run Spark with any Hadoop version by augmenting Spark's classpath. Scala and Java users can include Spark in their projects using its Maven coordinates and Python users can install Spark from PyPI.

If you'd like to build Spark from source, visit Building Spark.

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS), and it should run on any platform that runs a supported version of Java. This should include JVMs on x86_64 and ARM64. It's easy to run locally on one machine — all you need is to have `java` installed on your system `PATH`, or the `JAVA_HOME` environment variable pointing to a Java installation.

Spark runs on Java 8/11, Scala 2.12, Python 2.7+/3.4+ and R 3.1+. Java 8 prior to version 8u92 support is deprecated as of Spark 3.0.0. Python 2 and Python 3 prior to version 3.6 support is deprecated as of Spark 3.0.0. R prior to version 3.4 support is deprecated as of Spark 3.0.0. For the Scala API, Spark 3.0.0 uses Scala 2.12. You will need to use a compatible Scala version (2.12.x).

For Java 11, `-Dio.netty.tryReflectionSetAccessible=true` is required additionally for Apache Arrow library. This prevents `java.lang.UnsupportedOperationException: sun.misc.Unsafe or java.nio.DirectByteBuffer.(long, int) not available` when Apache Arrow uses Netty internally.

Running the Examples and Shell

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Built-in Data Sources

Parquet/ORC Nested Column Pruning

CSV Filter Pushdown

Parquet: Nested Column Filter Pushdown

New Binary Data Source

Richer APIs

Accelerator-aware Scheduler

Built-in Functions

pandas UDF Enhancements

DELETE/UPDATE/ MERGE in Catalyst

SQL Compatibility

Overflow Checking

ANSI Store Assignment

Proleptic Gregorian Calendar

Reserved Keywords

Extensibility and Ecosystem

Data Source V2 API + Catalog Support

Hadoop 3 Support

Hive 3.x Metastore Hive 2.3 Execution

Java 11 Support

Monitoring and Debuggability

Structured Streaming UI

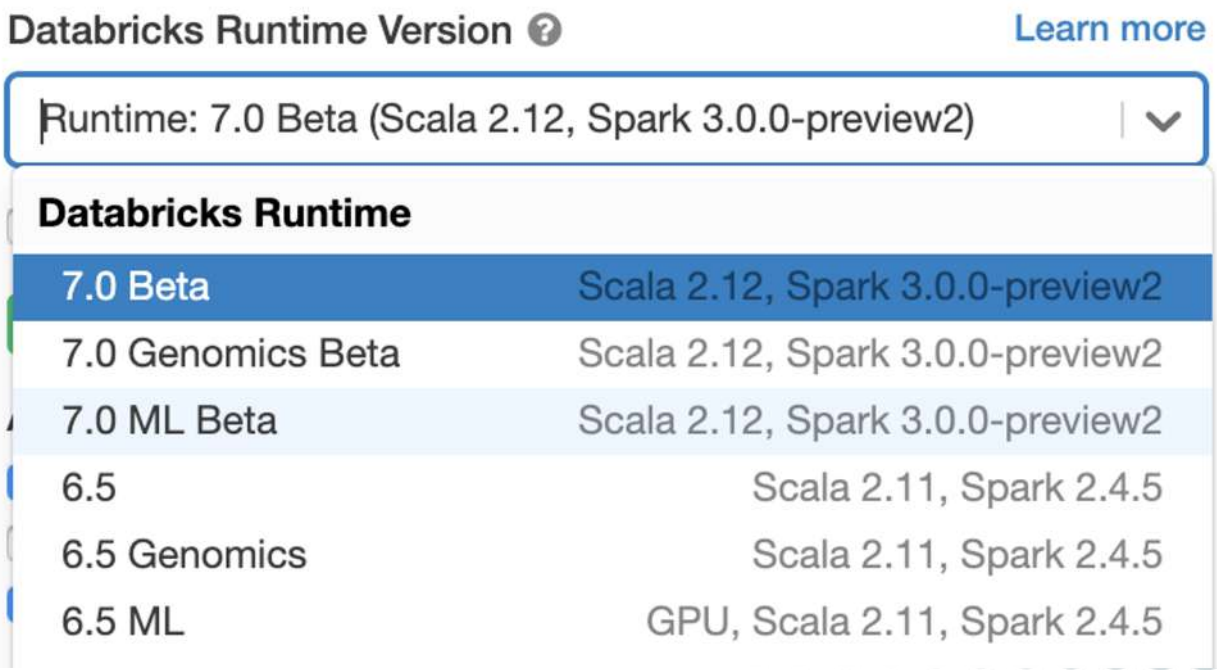
DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Try Databricks Runtime 7.0 Beta For Free

https://databricks.com/try-databricks



Thank you for your contributions!


SPARK+AI SUMMIT
June 22-26 | Organized by databricks
THE VIRTUAL EVENT FOR DATA TEAMS
• Extended to 5 days with over 200 sessions
• 4x the pre-conference training
• Keynotes by visionaries and thought leaders
NOW FREE

阿里云 × intel × DAMO ALIBABA DAMO ACADEMY

Spark AI

Challenge Round

"数字人体"挑战赛 _ 脊柱疾病智能诊断大赛

主办单位 × 阿里巴巴云计算有限公司
协办单位 × 英特尔（中国）有限公司
合作单位 × 唯医骨科
指导单位 × 湘雅医院；浙江大学附属第二附属医院；解放军301医院；香港大学骨科学系；

Meetup *S01

BigData+AI

微信公众号
Apache Spark技术交流社区