GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

What’s new with Spark 3

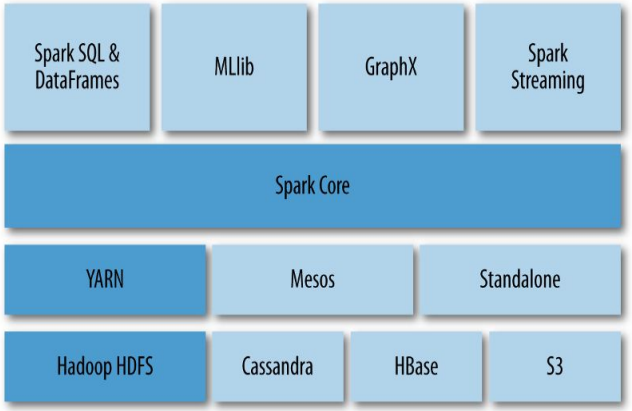
Abdul Rafay
MS in Computer Science Student

Outline

1. **Introduction**
2. Spark Deconstructed
3. Case Studies
4. Spark 3 Features
5. Summary

Apache Spark

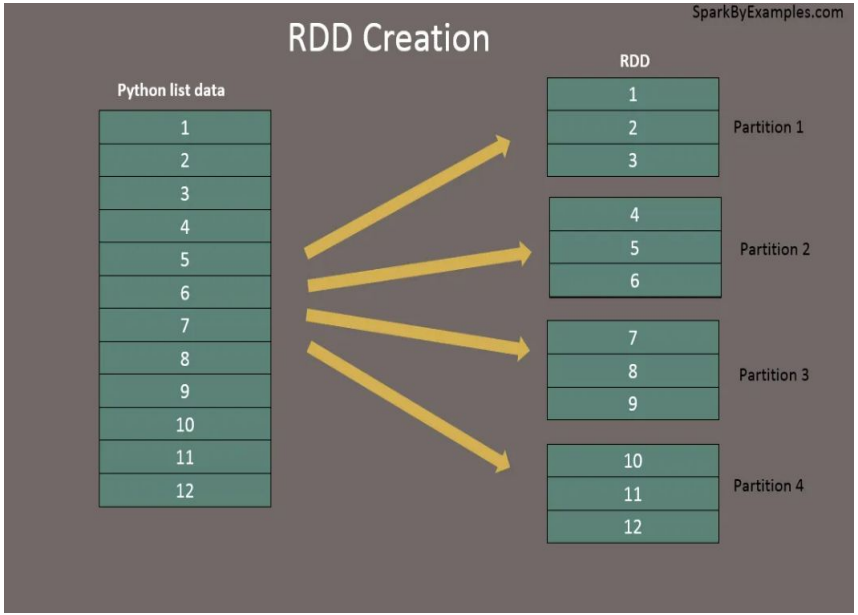
- ★ Apache Spark is a general purpose cluster computing platform.
- ★ Apache Spark is In-memory lightening fast processing engine.
  - It complements Hadoop.
  - Data Source can be any SQL/ NoSQL Database, Kafka etc.
  - Resource management via Mesos, Yarn, Kubernetes or own cluster.
- ★ In-memory data storage for fast iterative processing.
  - At least **10x** faster than Hadoop.
- ★ Compatible with Hadoop's storage APIs.
  - Can read/write to any Hadoop-supported system, including HBase, HDFS.
- ★ Spark contains 4 modules on top of spark core and serves as a unified data analytics engine.
- ★ Execution of the code/pipeline can be in client or cluster mode.



Source: "kdnuggets" [2]

Apache Spark Data Model

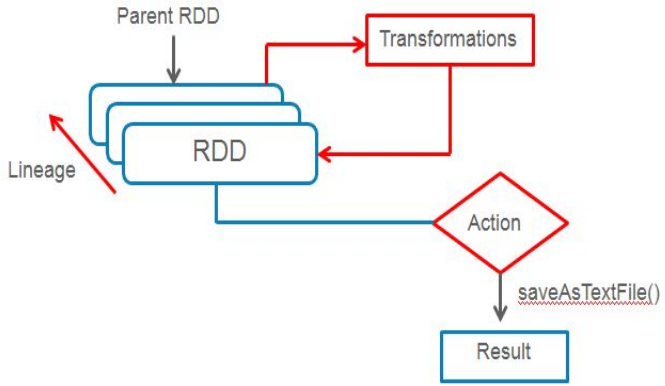
- ★ Apache Spark process data in RDD.
  - Resilient: Fault-tolerant
  - Distributed: Computed across multiple nodes
  - Dataset: Collection of partitioned data
- ★ RDDs are Immutable once constructed.
- ★ Track lineage information.
- ★ RDDs Lifecycle
  - RDDs are destroyed upon SparkContext terminates.
  - Can be cache in memory or persist to any data storage.
- ★ RDDs reside under the Spark APIs (Dataframe, Dataset and SQL)



Source: “sparkbyexamples” [1]

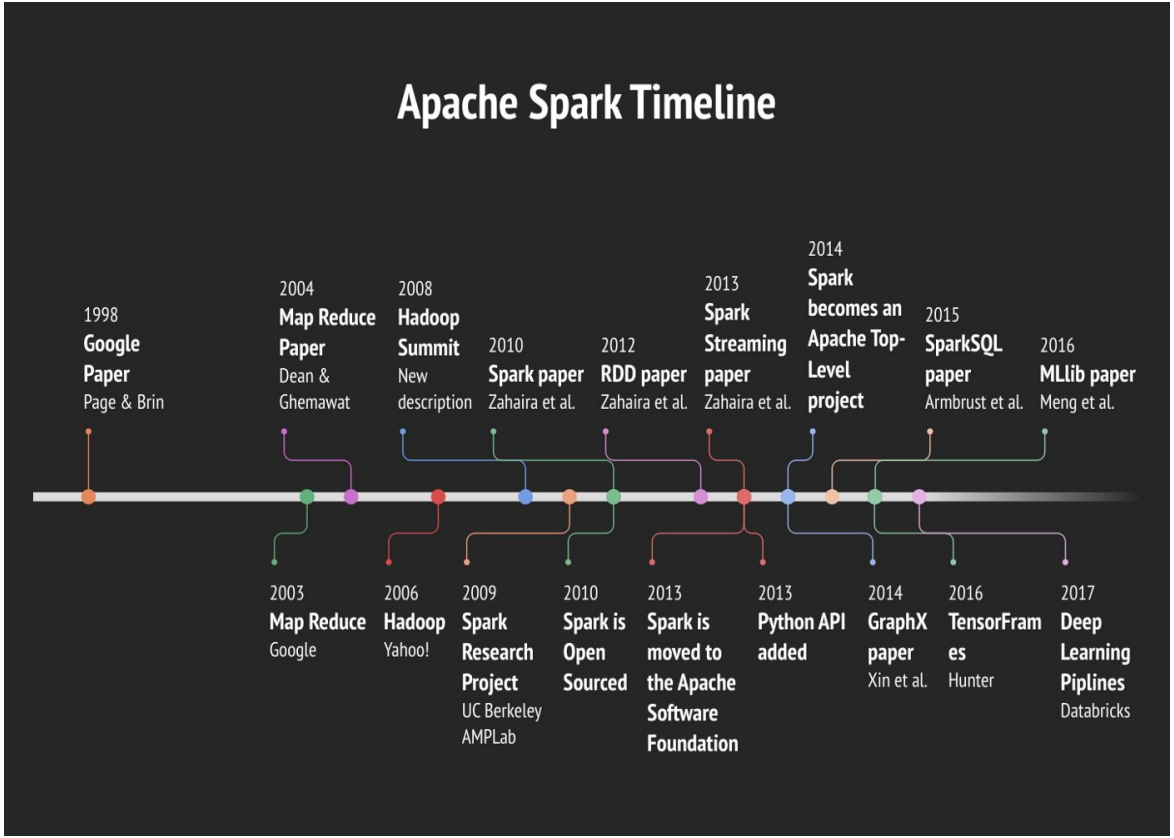
Apache Spark Internals

- ★ Operates data in RDDs.
  - Unlike Hadoop's **MapReduce** key-value pairs.
- ★ Two main operations in Spark.
  - Transformations (Filter, Group, Map, reading etc)
  - Actions (display, writing data, count etc) which triggers job.
- ★ Spark is **fault-tolerant**. It is handled by a data lineage.
  - Automatically rebuilt on failure.
- ★ **Lazy Evaluation:** Data inside RDDs are executed when action is called.
  - Benefit: Query plan optimisation.
- ★ **Location-Stickiness:** task is close to data as much as possible.



Apache Spark History

- ★ Created at the AMP Lab at Berkeley.
  - The original creators of Spark later found a company named Databricks.
- ★ Spark was created to address bringing data and machine learning together.
- ★ Donated to the Apache Foundation to make it open source project.



Source: “kdnuggets” [2]

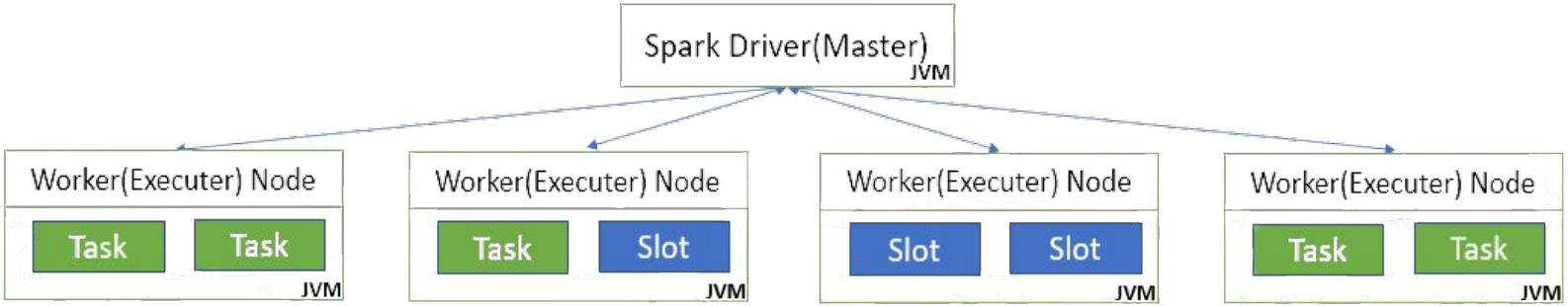
Outline

1. Introduction
2. **Spark Deconstructed**
3. Case Studies
4. Spark 3 Features
5. Summary

What is Cores/Slots/Threads

★ Spark parallelizes at two levels
- One is the splitting the work among **executors**.
- The other is the **slot** (Thread).

★ **Tasks:** Driver assign units of work to Slots on each Executor for parallel execution.

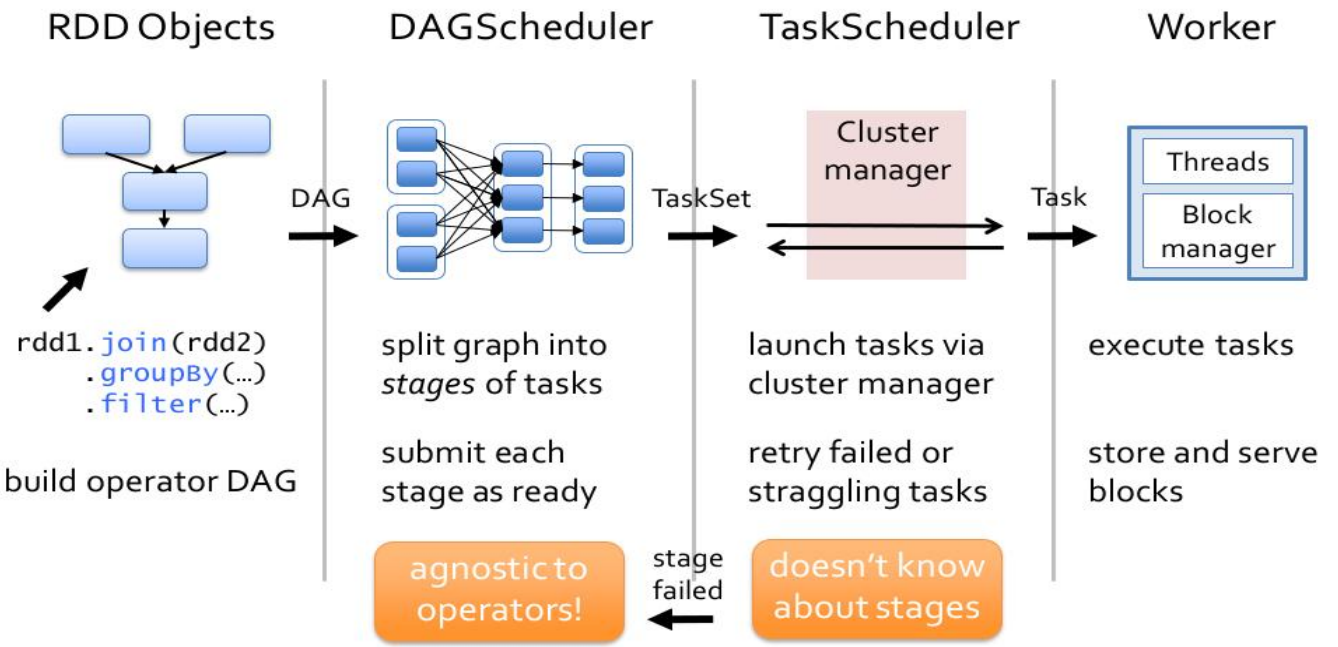


Source: “getinfoget” [3]

What is a DAG

- ★ Directed Acyclic Graph (DAG) in Spark is a set of Vertices and Edges.
  - vertices represent the RDDs.
  - edges represent the Operation to be applied on RDDs.
- ★ **DAGScheduler:** scheduling layer that implements stage-oriented scheduling.

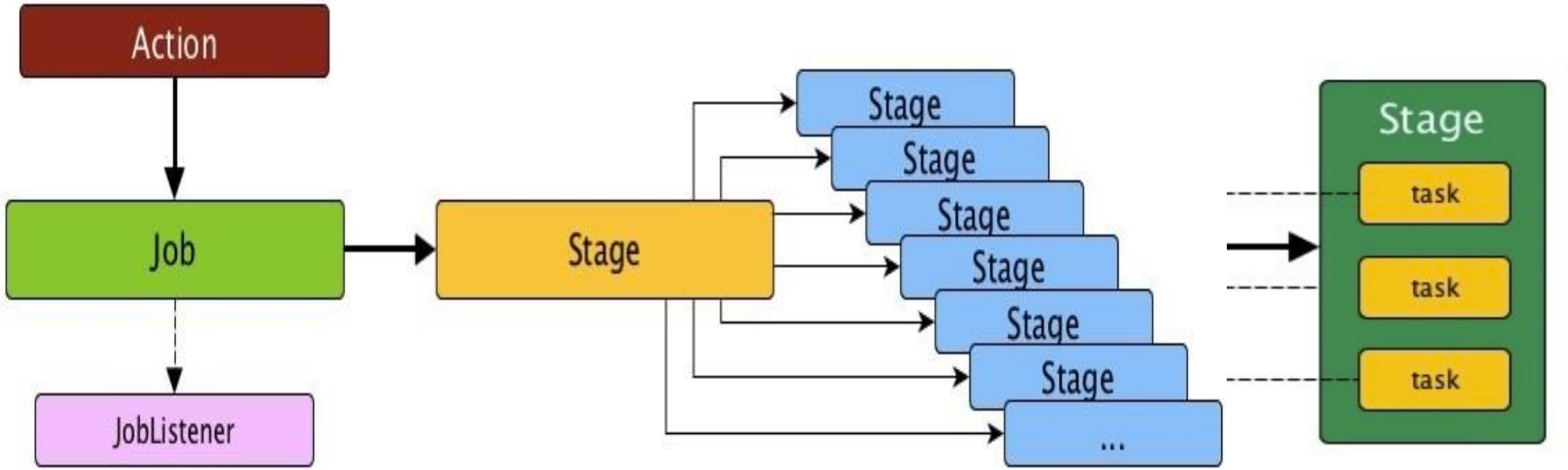
Source: “stackoverflow” [4]



What is a Job and Stage

★ **Job** is a sequence of stages, triggered by an action.
- Each parallelized action is referred to as a Job.

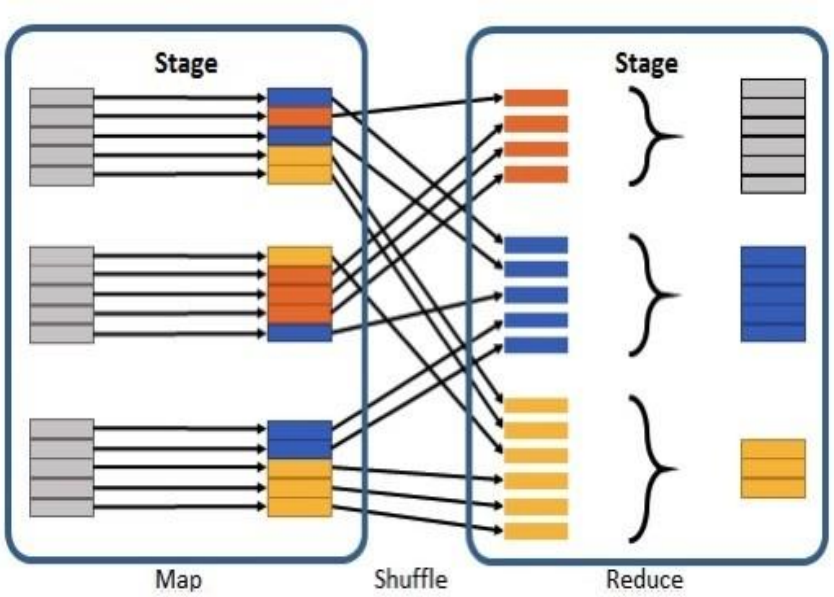
★ Each job that gets divided into smaller sets of tasks is a **stage**.
- A Stage is a sequence of **Tasks** that can be performed in one stage without **data shuffle**.

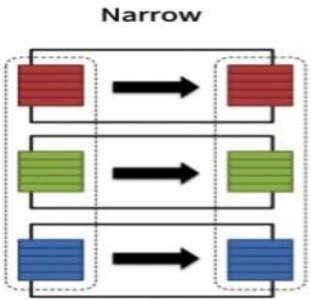


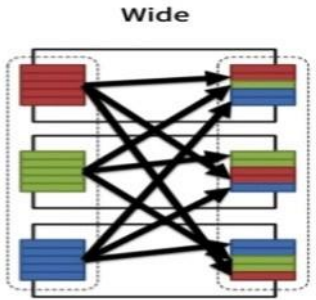
Source: “mallikarjuna_g” [5]

What is Caching and Shuffling

- ★ **Caching** will place a DataFrame or table into temporary storage across executors to make reads faster.
- ★ **Shuffle** refers to an operation where data is re-partitioned across a Cluster - i.e. when data needs to move between executors.
- ★ Types of Transformations (Narrow and Wide).
  - **Narrow**: Does not require shuffle.
  - **Wide**: Requires shuffle. Data resides in many partitions.
    - E.g, transformations (Join, GroupByKey etc).







Source: “stackoverflow” [6]

Outline

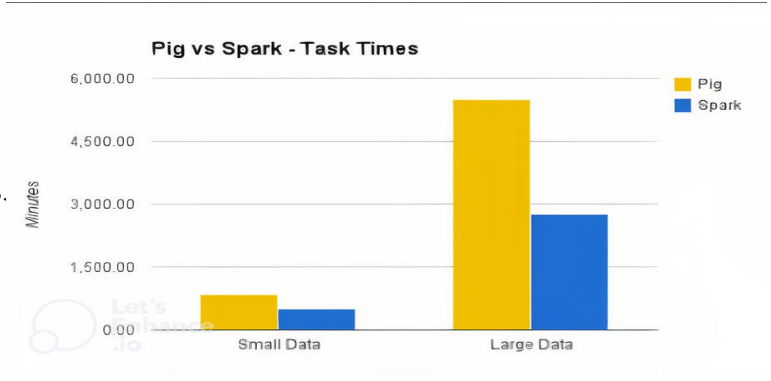
1. Introduction
2. Spark Deconstructed
3. **Case Studies**
4. Spark 3 Features
5. Summary

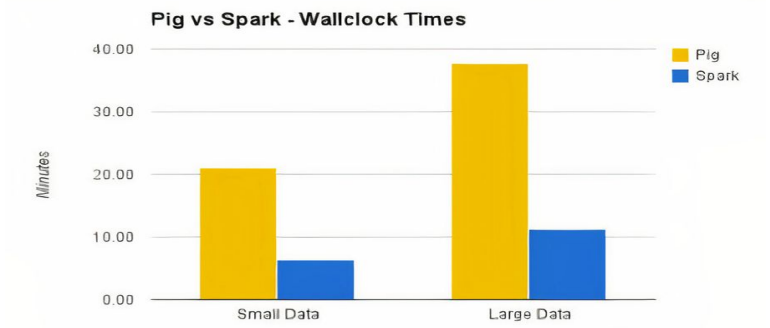
Motivation

1. How Apache Spark helps companies.
2. How they store and process data with Spark.
3. What problems they have solved with Spark.

Spark at Twitter: Evaluation & Lessons Learnt

- ★ Twitter's data
  - 241M+ monthly active users
  - ~500M tweets a day
  - 100TB+ of compressed data every day
- ★ Analytics Infrastructure at Twitter
  - Several more than 1K+ node Hadoop clusters for different purposes.
- ★ Goals: They want to achieve
  - Improve cluster utilization and efficiency.
  - Iterative algorithms, data sets caching etc.
- ★ Twitter Setup for benchmarking.
  - YARN cluster with 35 nodes (24GB RAM, 32 cores) each node.
  - Spark worker memory with 8GB.
- ★ Observations and Performance
  - Spark performed 4x faster than MapReduce in wall clock time.
  - Fewer I/O synchronization barriers

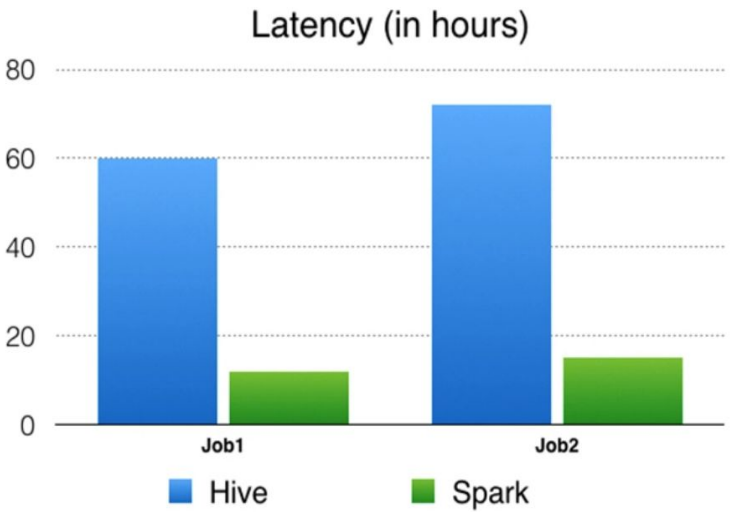




Spark at Meta: Feature preparation for entity ranking

Meta

- ★ Facebook often uses analytics for data-driven decision making.
- ★ Bottlenecks
  - Hive-based infrastructure: computationally resource intensive and challenging to maintain.
  - 10s of Terabytes of data generated every day.
- ★ Analytics Infrastructure at Meta
  - Data in TBs stored in HDFS and processed through Spark.
- ★ Goals: They want to achieve
  - Dealing with frequent node reboots for long-running jobs.
  - Processing massive amount of data in restrict time.
- ★ Observations and Performance
  - 4.5-6x CPU, 3-4x resource reservation, and ~5x latency faster than Hive.
  - Better maintainability and flexibility.



Outline

1. Introduction
2. Spark Deconstructed
3. Case Studies
4. **Spark 3 Features**
5. Summary

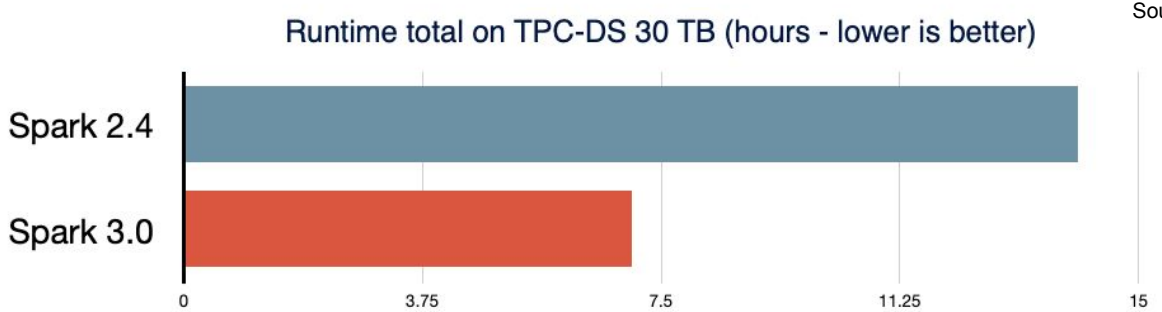
Motivation

1. Describe major improvements to performance in Spark 3.0
2. Identify major usability improvements in Spark 3.0
3. Performance improvement factors

Overview

- ★ Each new release of Spark is to make Spark faster, easier, and smarter.
  - Spark 3.0 a major release extends its scope with more than 3000 resolved JIRAs. 46% tickets resolved for Spark SQL. These enhancements benefit all the higher-level libraries.
- ★ Spark 3.x release delivered many new capabilities and performance gains.
- ★ Major performance-related features:
  - **Adaptive Query Execution (AQE)**. A progression to Catalyst Optimizer.
  - **Dynamic Partition Pruning (DPP)** speed up between 2 times and 18 times.
  - **Richer APIs and New Pandas UDF:** new features and simplify development.
  - New User Interface for Structured Streaming in the Spark UI.
  - Better Kubernetes Integration.
  - DDL, DML enhancements and more than 30 new built-in functions.

Source: “Databricks” [7]

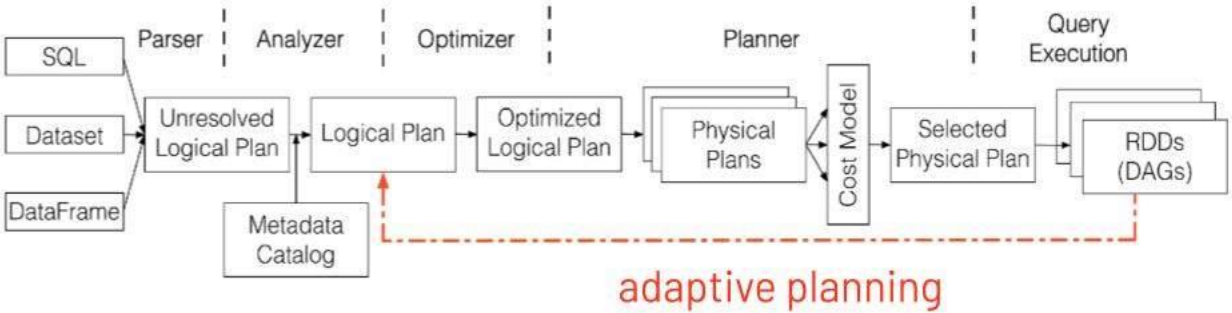


Adaptive Query Execution

★ Catalyst optimizer is at the core of Spark SQL.
- **Spark 2.x** applies optimizations throughout logical and physical planning stages.

★ Runtime statistics collected during query execution.
- Spark will feedback statistics about the size of the data in the shuffle files. That performs:
  - **switch join strategies**
  - **coalesce the number of shuffle partitions.**
  - **optimize skew joins**

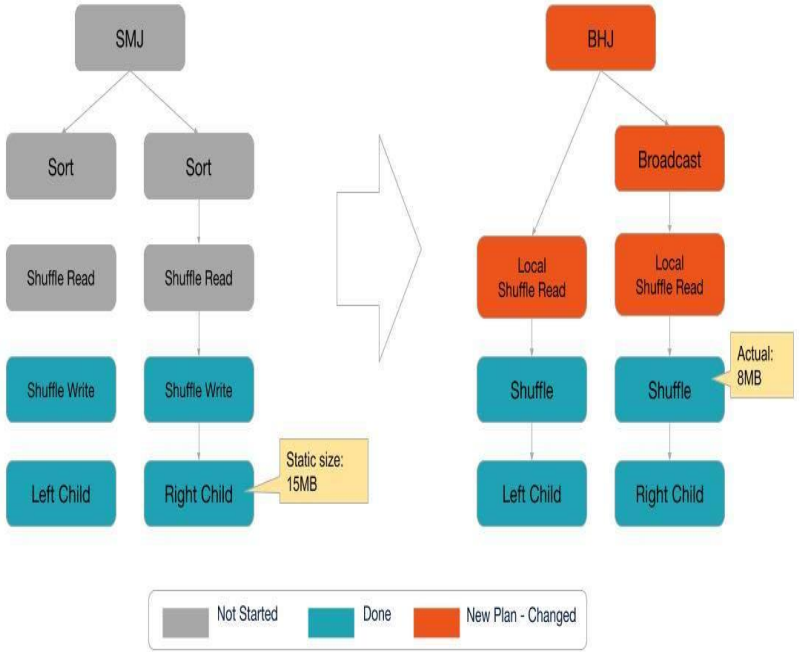
★ AQE is turned off by default.
- Can be enable by setting `spark.sql.adaptive.enabled` to `true` in the configuration file.



Source: “Databricks” [7]

Adaptive Query Execution | switch join strategies

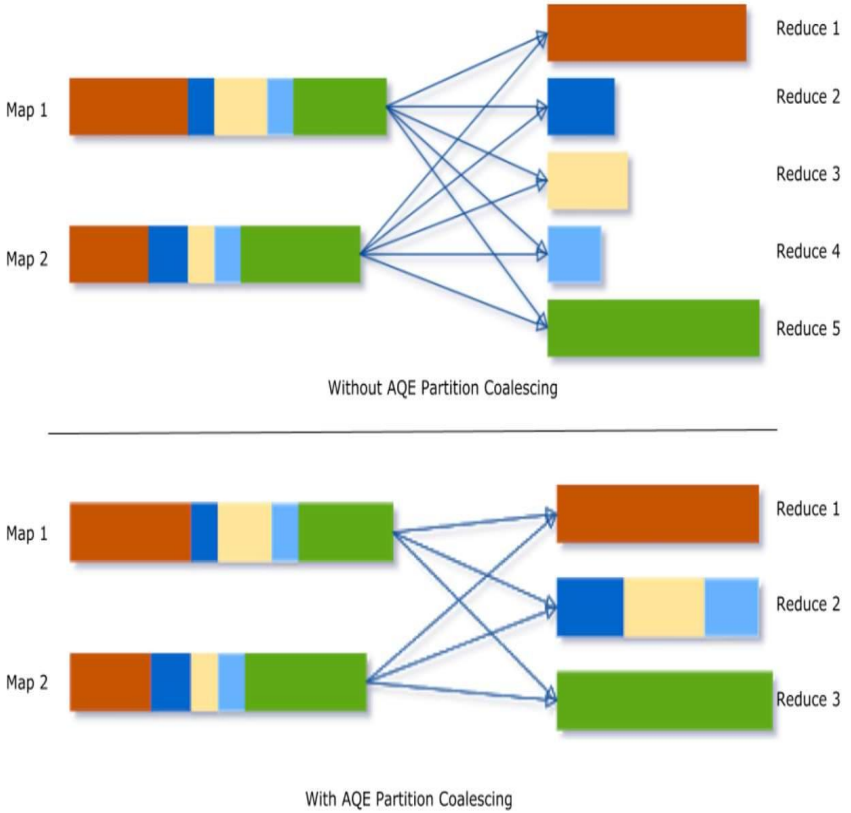
- ★ Broadcast Hash Join is one of the most performant join strategies supported by Spark.
- ★ Without AQE: Existing rule-based optimization.
  - Default broadcast-size threshold is 10MB.
- ★ AQE now replans the join strategy at runtime.
  - It is based on the most accurate join relation size.
- ★ With AQE, Spark is able to dynamically switch join strategies to use the more performant broadcast-hash join.



Source: “Databricks” [8]

Adaptive Query Execution | coalesce the number of shuffle partitions

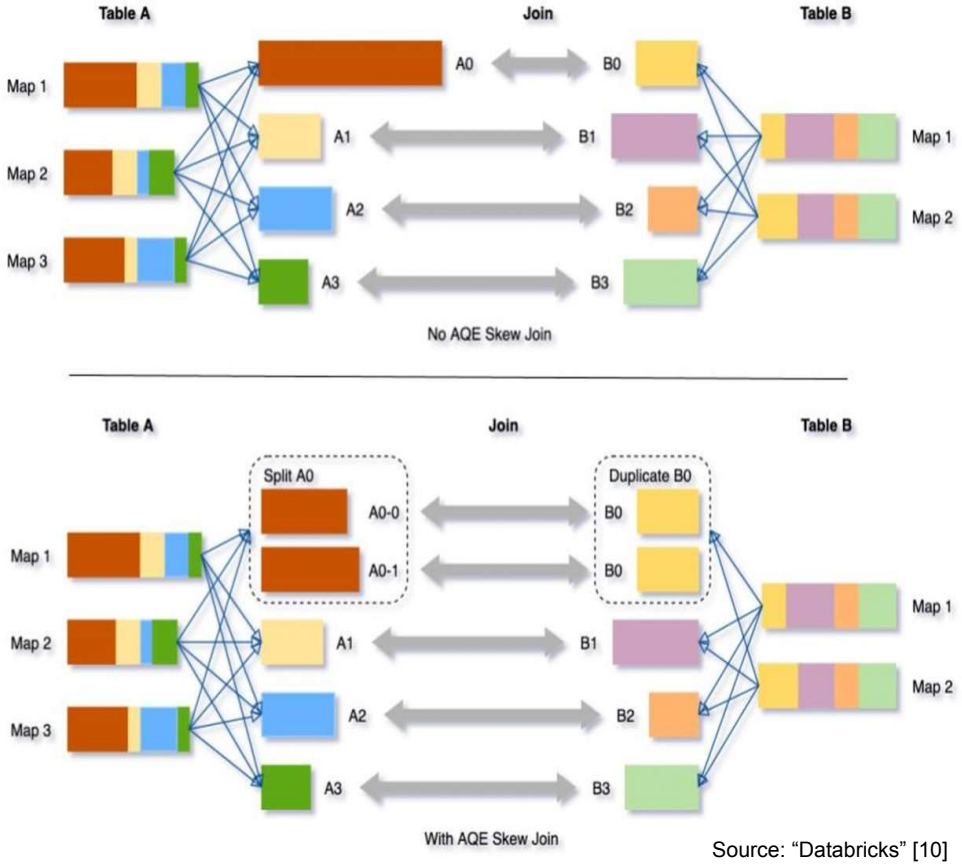
- ★ **If there are too few partitions.**
  - data size of each partition may be very large.
  - large partitions may need to spill data to disk
- ★ **If there are too many partitions.**
  - data size of each partition may be very small.
  - lot of small network data fetches to read the shuffle blocks.
  - inefficient I/O pattern.



Source: “Databricks” [9]

Adaptive Query Execution | optimize skew joins

- ★ Data skew occurs when data is unevenly distributed among partitions in the cluster.
- ★ **AQE skew join optimization** detects such skew automatically from shuffle file statistics.
  - splits the skewed partitions into smaller subpartitions.
  - parallelize skew processing and achieve better overall performance.



Source: “Databricks” [10]

Adaptive Query Execution In Practice

★ A stage in spark resubmits its optimized DAG as a new job.

★ In Spark 2:
- Stage has 200 tasks (default number of tasks after a shuffle), 170 KB per task, and took 18 seconds to complete.

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 s	1 s	1 s	2 s	3 s
GC Time	0 ms	0 ms	0 ms	0 ms	0.2 s
Output Size / Records	5.7 KB / 72	6.2 KB / 89	6.5 KB / 96	6.7 KB / 103	7.5 KB / 126
Shuffle Read Size / Records	7.4 KB / 77	46.4 KB / 3585	172.3 KB / 16267	396.6 KB / 45727	2.4 MB / 302711

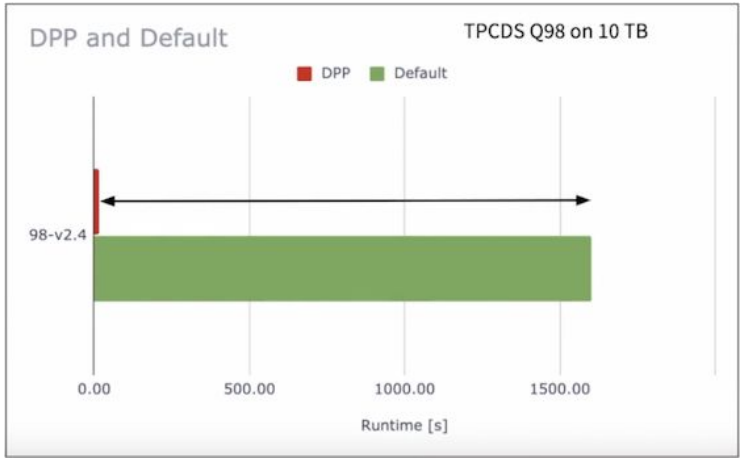
★ In Spark 3:
- Stage has 50 tasks, 1450 KB, and took 5 seconds to complete.
- 68% performance gain.

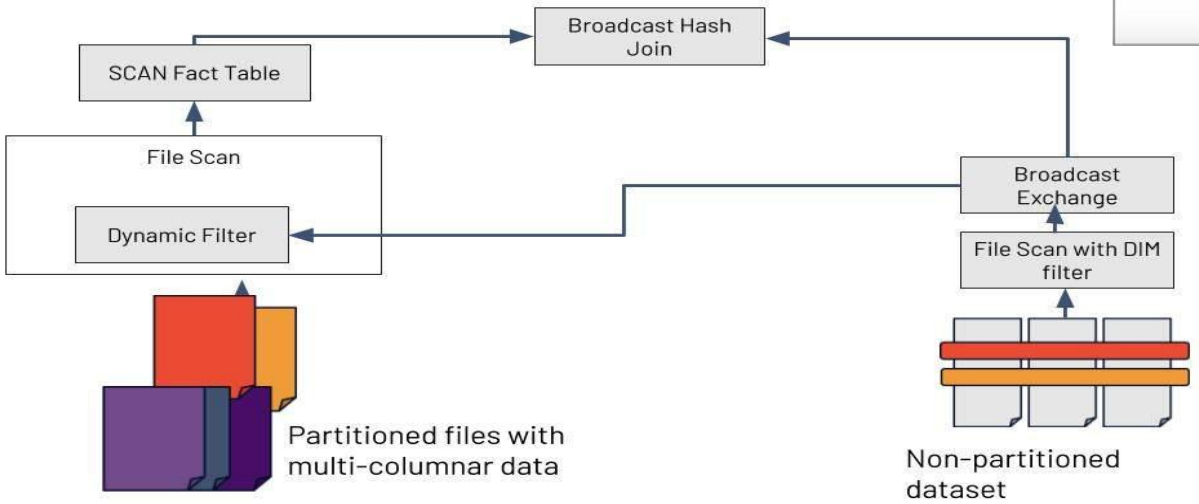
Summary Metrics for 50 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 s	3 s	3 s	3 s	4 s
GC Time	0 ms	0 ms	0 ms	0 ms	0.2 s
Output Size / Records	6.1 KB / 87	7.1 KB / 115	12.2 KB / 279	21.3 KB / 563	50.7 KB / 1544
Shuffle Read Size / Records	63.7 KB / 4805	944.0 KB / 110498	1458.2 KB / 167408	1654.6 KB / 217137	2.4 MB / 302711

Dynamic Partition Pruning

- ★ Auto-optimize queries to make them more performant automatically.
- ★ DPP allows to read only **as much data as you need**.
- ★ DPP's Optimisation is implemented both on the logical plan optimization and the physical planning.



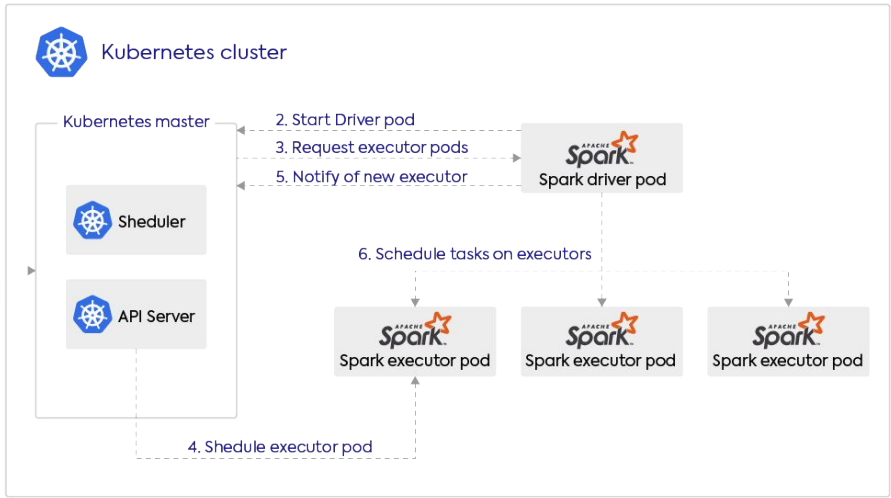


Source: "Databricks" [11]

Better Kubernetes Integration

- ★ Matured Spark support for Kubernetes in **3.x** and easy to use in production.
- ★ New shuffle service for Spark on Kubernetes that will allow dynamic scale up and down.
- ★ Spark 3.0 also supports GPU support with pod level isolation for executors.
  - More flexible scheduling on a cluster with GPUs.
  - Spark Authentication support on Kubernetes.
- ★ Spark Driver and Executors run inside the Kubernetes pod.

Apache Spark on Kubernetes Architecture



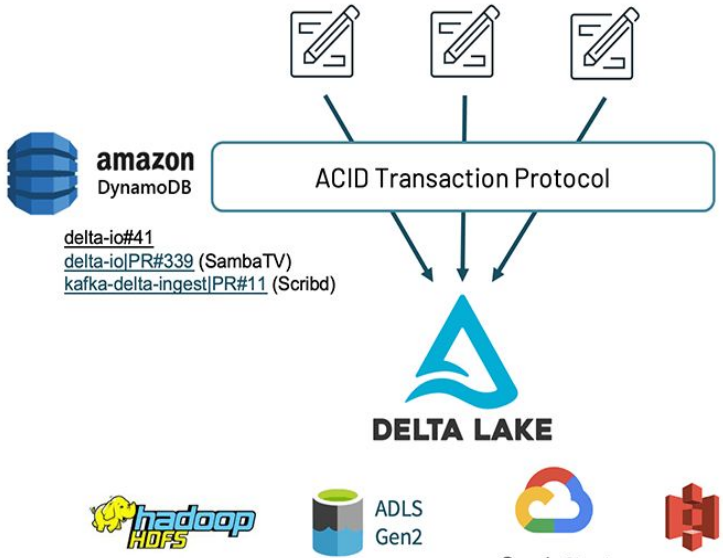
ACID Transactions with Delta Lake

★ Delta Lake is an open-source storage layer that brings ACID transactions to Apache Spark 3.0.
- brings reliability to Data Lakes

★ It solves issues presented when data in the data lake is modified simultaneously by multiple modifiers.
- allows to focus on logic and not worry from inconsistencies.
- It maintains the state in access log file.

★ Built on top of Apache Parquet file format.

Multi-cluster Transactions*



* Work in-progress

Improved Pandas UDFs and Arrow Integration

- ★ New UDFs have been redesigned to leverage Python type hints.
  - no longer need to remember any UDF types.
  - can use type hints to express the new Pandas UDF types.
- ★ **Apache Arrow** is an in-memory columnar data structure for efficient analytical operations.
  - cross-language platform.
  - zero-copy streaming messaging
  - interprocess communications without serialization costs.
- ★ Arrow is used to improve the interchange between the Java and Python processes.
  - enables new features like Arrow accelerated UDFs.

Spark 2.3

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

@pandas_udf('long', PandasUDFType.SCALAR)
def pandas_plus_one(v):
    # `v` is a pandas Series
    return v + 1  # outputs a pandas Series

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

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


# New type of Pandas UDF in Spark 3.0.
@pandas_udf('long', PandasUDFType.SCALAR_ITER)
def pandas_plus_one(itr):
    # `iterator` is an iterator of pandas Series.
    return map(lambda v: v + 1, itr)  # outputs an iterator of pandas Seri

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

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

@pandas_udf("id long", PandasUDFType.GROUPED_MAP)
def pandas_plus_one(pdf):
    # `pdf` is a pandas DataFrame
    return pdf + 1  # outputs a pandas DataFrame

# `pandas_plus_one` can _only_ be used with `groupby(...).apply(...)`
spark.range(10).groupby('id').apply(pandas_plus_one).show()
```

Spark 3.0

```
def pandas_plus_one(v: pd.Series) -> pd.Series:
    return v + 1
```

```
def pandas_plus_one(itr: Iterator[pd.Series]) -> Iterator[pd.Series]:
    return map(lambda v: v + 1, itr)
```

```
def pandas_plus_one(pdf: pd.DataFrame) -> pd.DataFrame:
    return pdf + 1
```

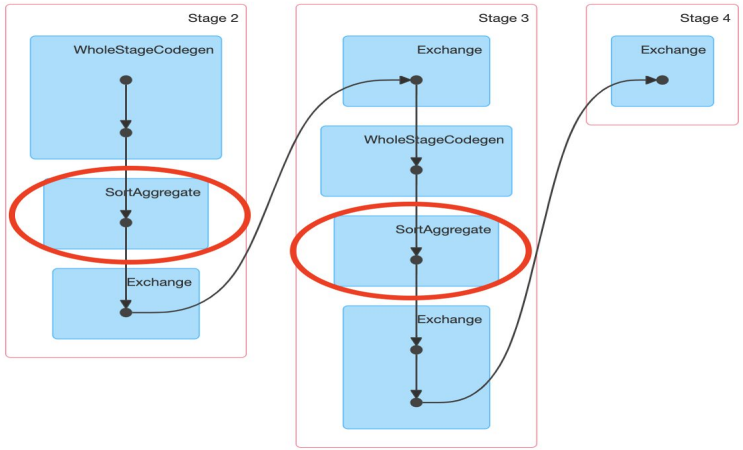
APACHE ARROW

Insights about using Spark 3

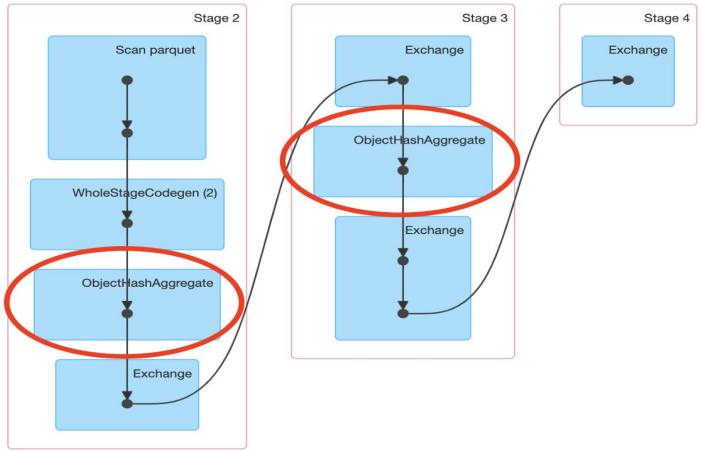
★ While aggregating, no use of serializing and deserializing of tuples.
- It reduces the pressure on Garbage collection.
- ~15% speedup.

★ The `SortAggregate` replaced by `ObjectHashAggregate`
- It saves to perform a sort step.
- ~20% time duration decrease in a stage.

★ In Spark 2:



In Spark 3:



Outline

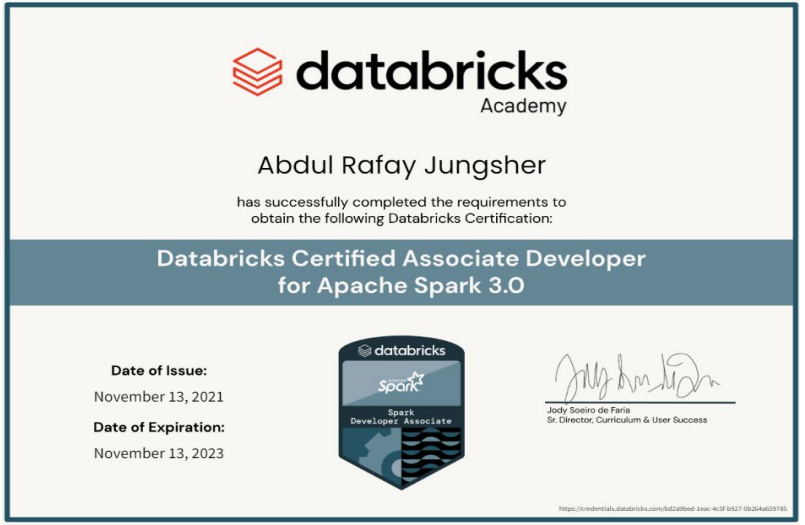
1. Introduction
2. Spark Deconstructed
3. Case Studies
4. Spark 3 Features
5. **Summary**

- ★ Contribute to Spark and related OSS projects via the email lists:
  - dev@spark.apache.org for people who want to contribute code
- ★ Learning Spark Holden Karau, Andy Kowinski, Matei Zaharia
  - O'Reilly (2015*) shop.oreilly.com/product/0636920028512.do
- ★ Databricks Certification - Apache Spark 3.0
  - https://databricks.com/learn/certification#specialty

Presenter Contact:

Abdul Rafay: a.abdulrafay@stud.uni-goettingen.de

- Databricks Certified Apache **Spark Developer**
  - DevOps Working Student at **SAP**



Summary

- ★ Spark is in-memory lightening fast data processing engine.
  - A framework for distributed computing which is based on RDDs.
  - In-memory, fault tolerant data structures.
  - API that supports Scala, Java, Python, R, SQL.
- ★ Computing Engine: perform computations over data somewhere (cloud, file, SQL database, Hadoop, Amazon S3 etc.). Not store it.
- ★ Fault-tolerant: Recompute lost partitions of dataset on failure.
- ★ Apache Spark 3.0 builds on many of the innovations from Spark 2.x
  - Every major release is the improvements in terms of speed and efficiency.
  - Spark SQL is the top active component in this release.
  - These enhancements benefit all the higher-level libraries.
- ★ Adaptive Query Execution (AQE) framework improves performance by generating a better execution plan at runtime.
- ★ Dynamic Partition Pruning improves the overall job performance.
- ★ Spark 3.0 roughly 2x better than Spark 2.4 in total runtime.