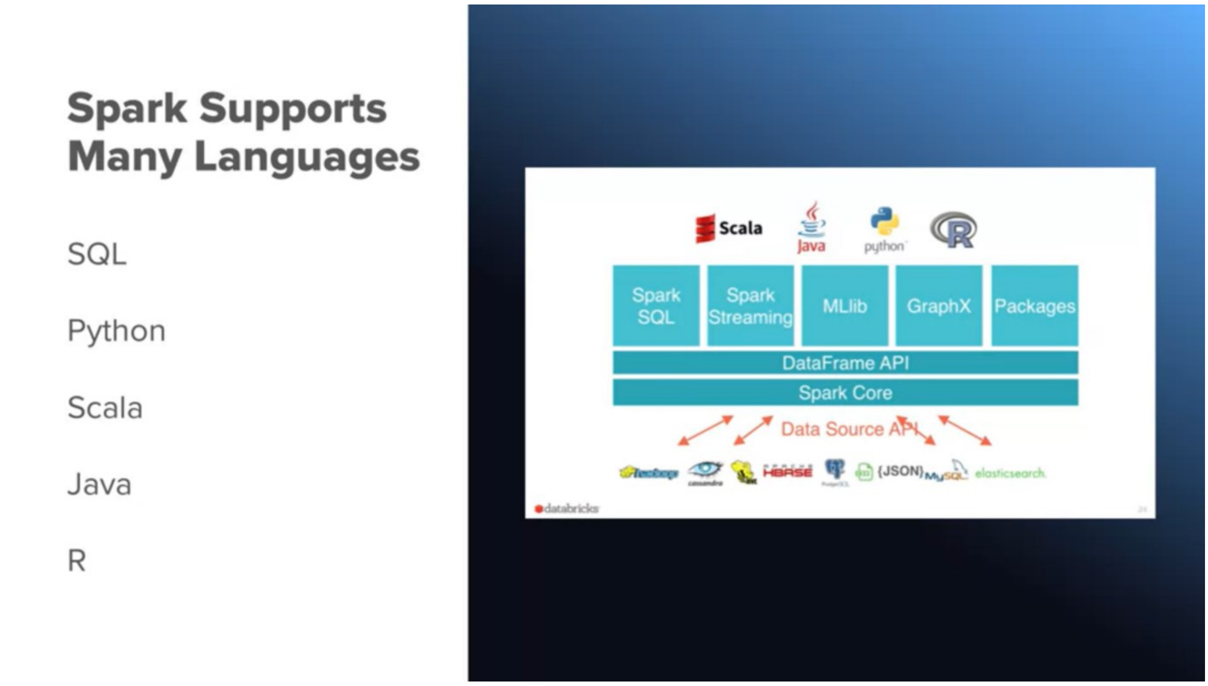
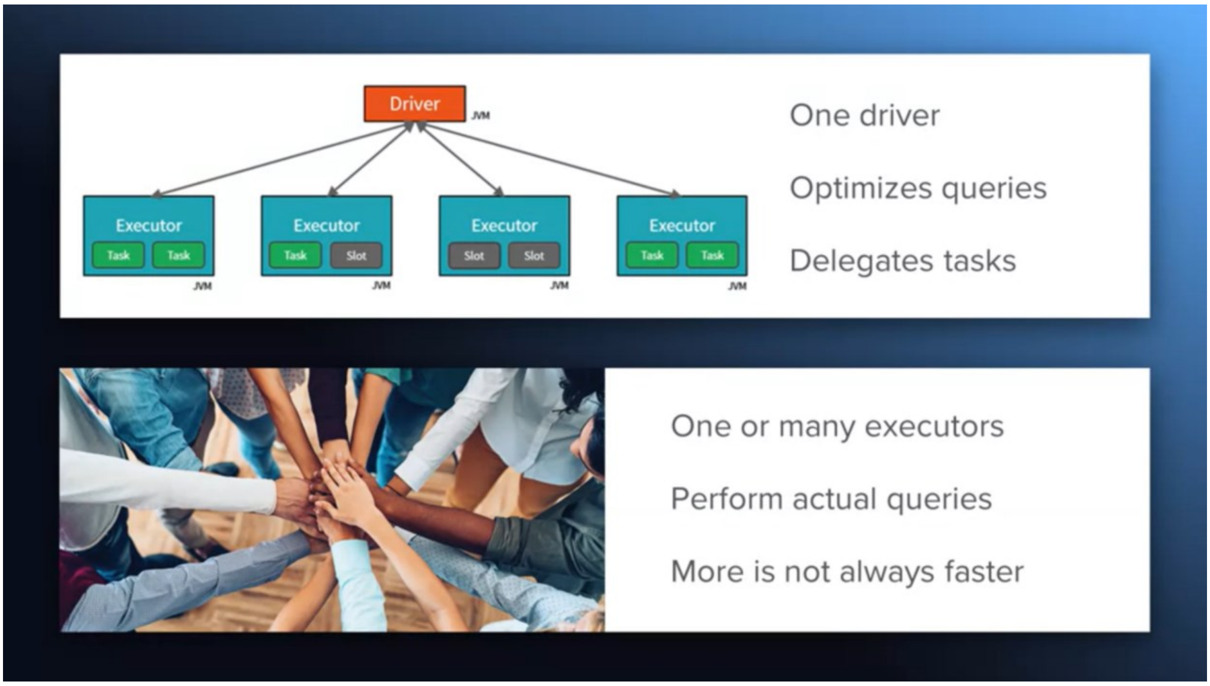
WEEK 1 DISTRIBUTED COMPUTING - DATA FRAME, RUNNING SQL QUERIES AND SCALE USING SPARK

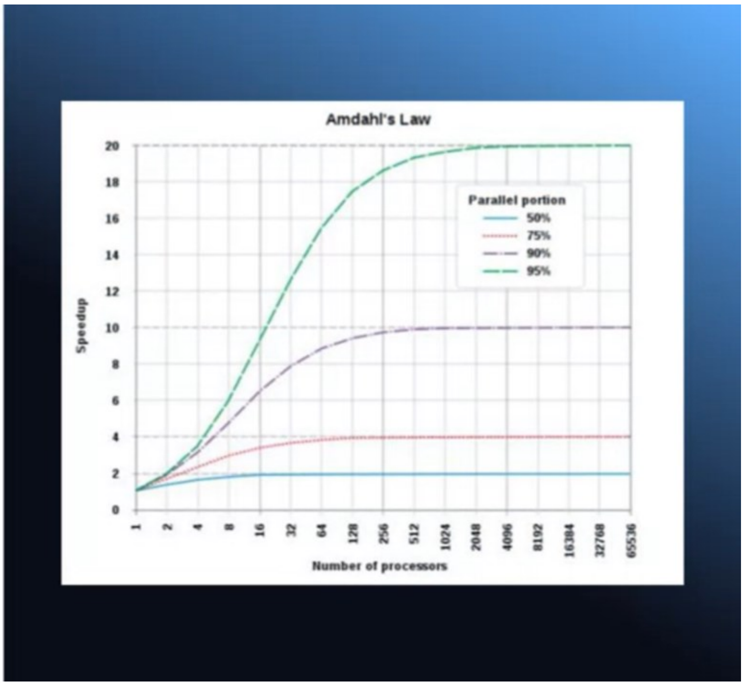
- Volume [size] – Velocity [Speed] – Variety [Datatype] – Veracity [validity of data &| source]





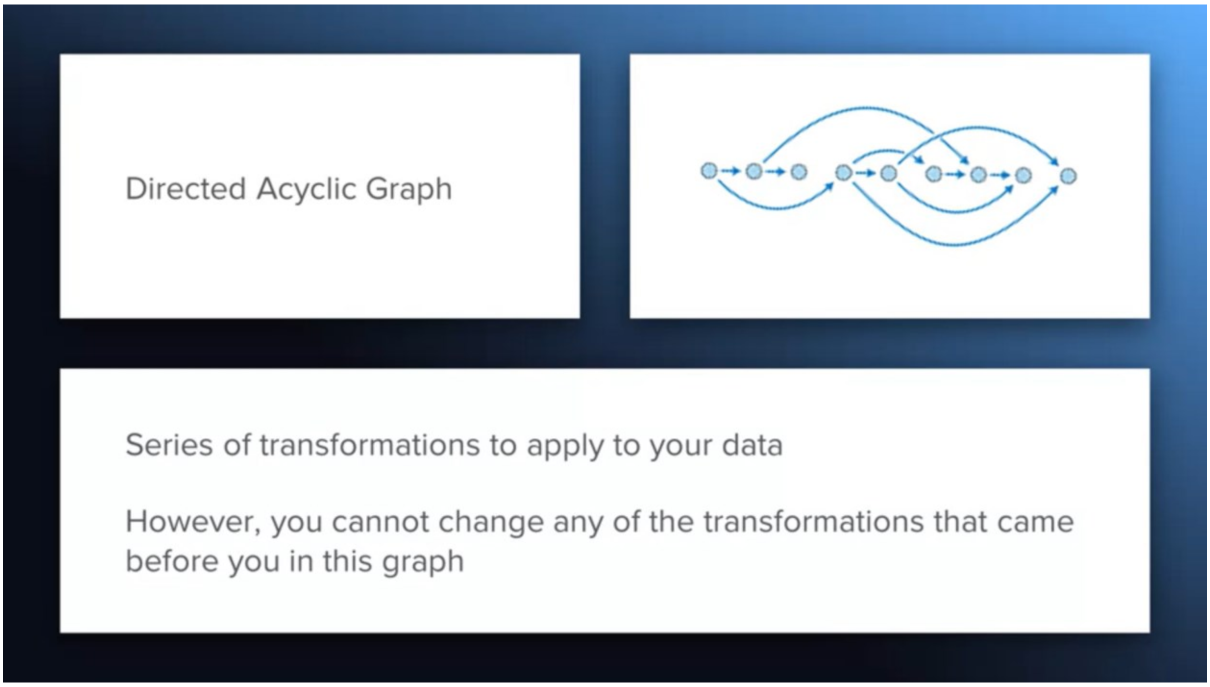
Amdahl's Law

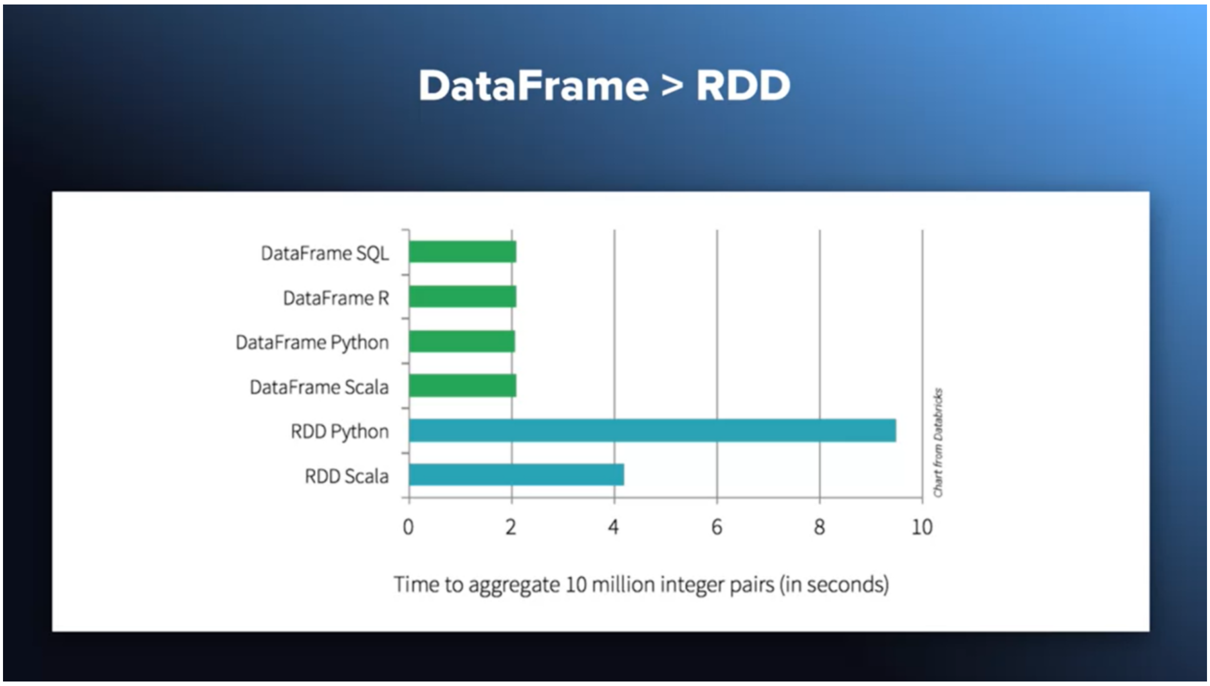
The amount of acceleration we would see from parallelizing a task is a function of what portion of the task can be completed in parallel



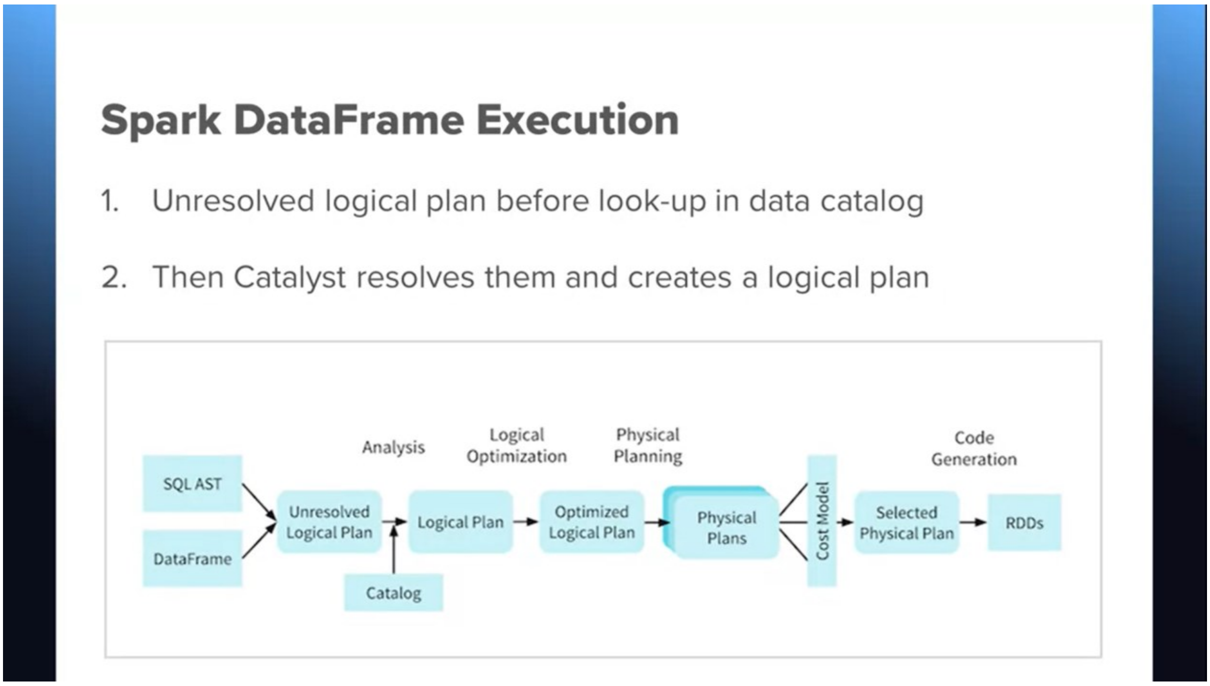
Resilient Distributed Datasets

- Acyclic chunks of computations – fault
- Distributed dataset across executers – each one operates on their share of data





Spark is declarative [WHAT] rather than imperative[HOW]



Databricks manages:

1. **Networking clusture**
2. **Easily deploy spark environment on AWS, Azure**

Readings and Resources

Course Notebooks:

- **https://files.training.databricks.com/courses/davis/Lessons.dbc**

Readings

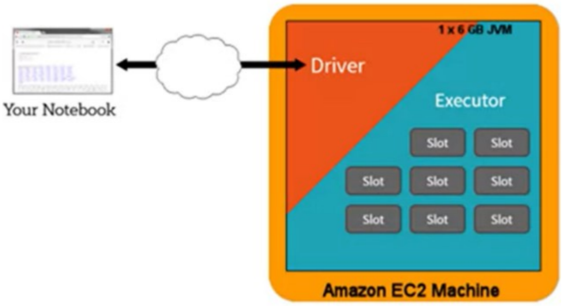
- **Spark SQL and DataFrames and Datasets Guides**
- **SQL Guide from Databricks**

Additional Resources

- **This is an optional resource that you could consider purchasing from this seller. This link is to get you started on an eBook version that does have a free trial option. Spark: The Definitive Guide**
- **This is a free gitbook: Introduction - The Internals of Spark SQL.**

WEEK 2 SQL FOR SPARK - CACHING, DEBUG SLOW QUERIES WITH SPARK UI

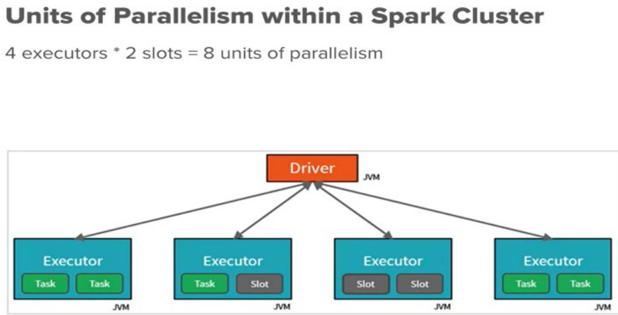
In local mode, Spark Driver-Executer can be single Compute instance TDM



Slot is essentially thread of execution

- **M**achines * **C**ores * **T**hreads

In production, Spark clusters runs multiple compute instances

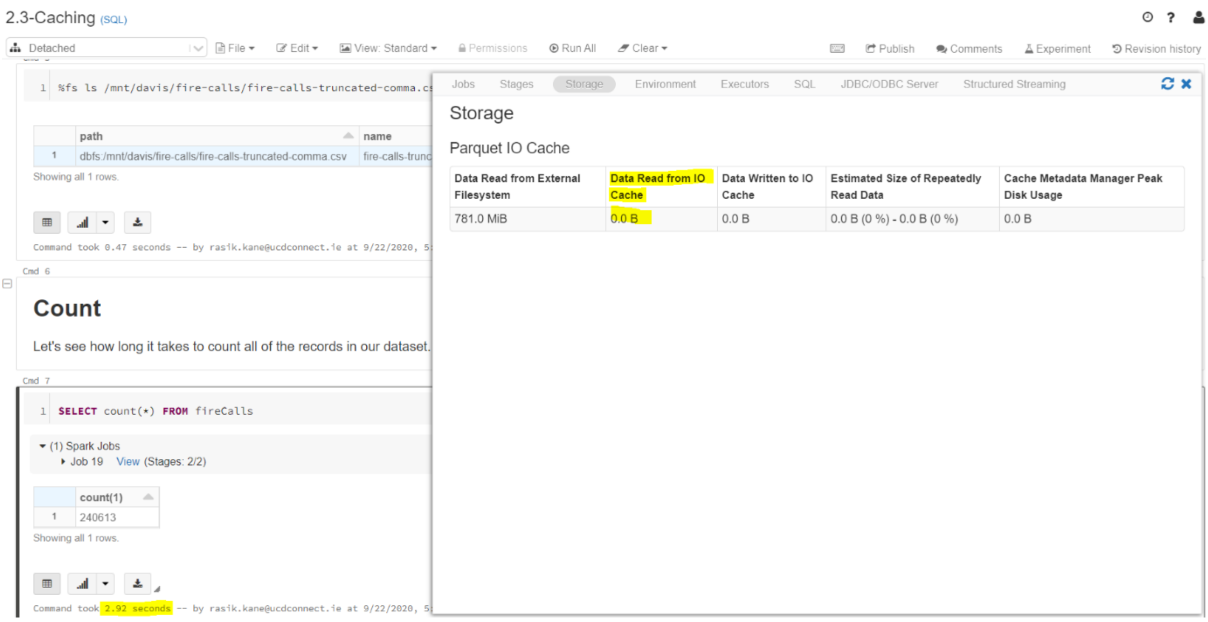


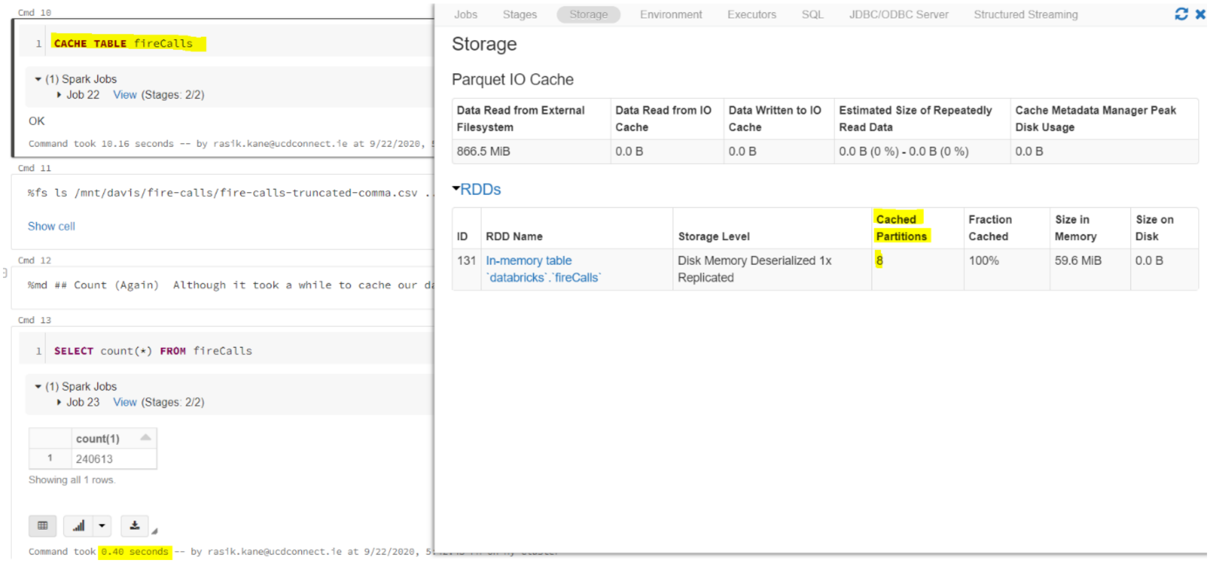
Partitions are Units of parallelism of data – they are subsets of data distributed across system [Count depends on size, feature governing partitions, cluster config]

- Designing partition size is handled by Databricks – Its tricky to balance **Computation**[Data/ compute] and **communication**[Number of compute cycles]

Cached Table: Entire table data partitioned and saved for faster access

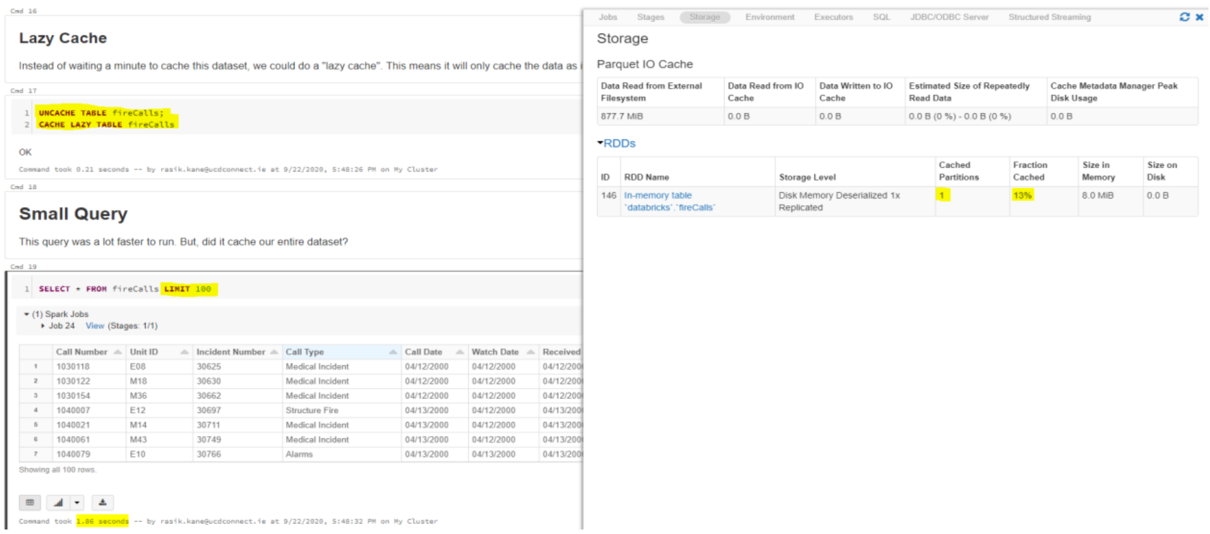
- Project Tungsten optimizes Data
- All data partitions are materialized

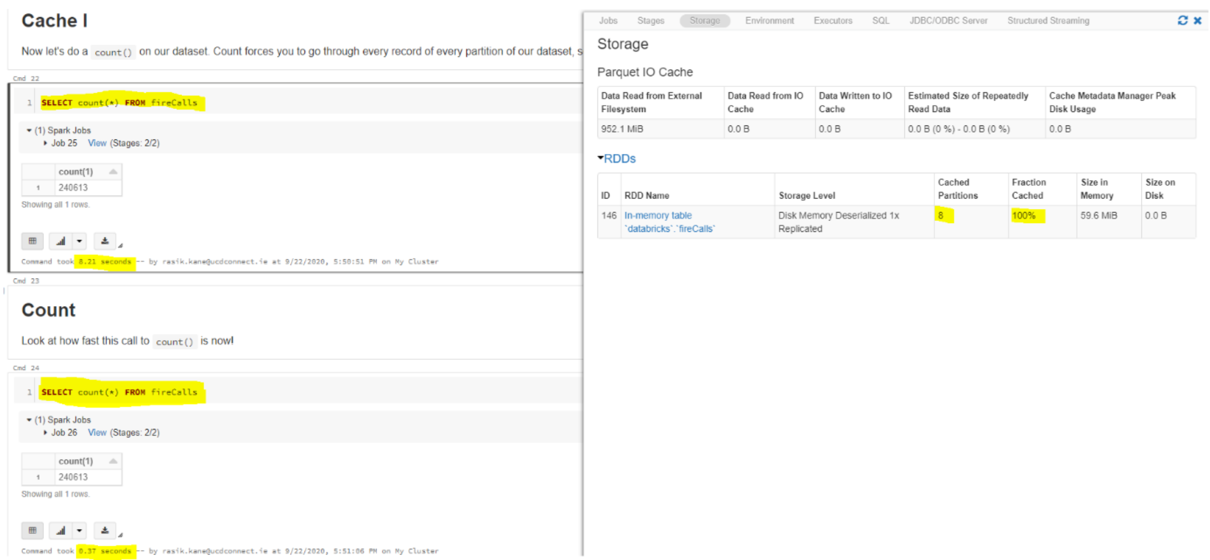




Lazy Cache Table: Only data view recently queried is considered for caching

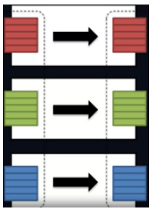
- **As data is queried; more and more partitions are materialized [physically created in cache]**





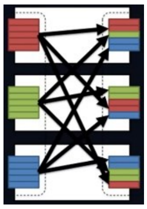
Narrow Transformation [In-Partition]

- Transform locally - eg. Select, Where
- Filter operations done on all partitions in parallel

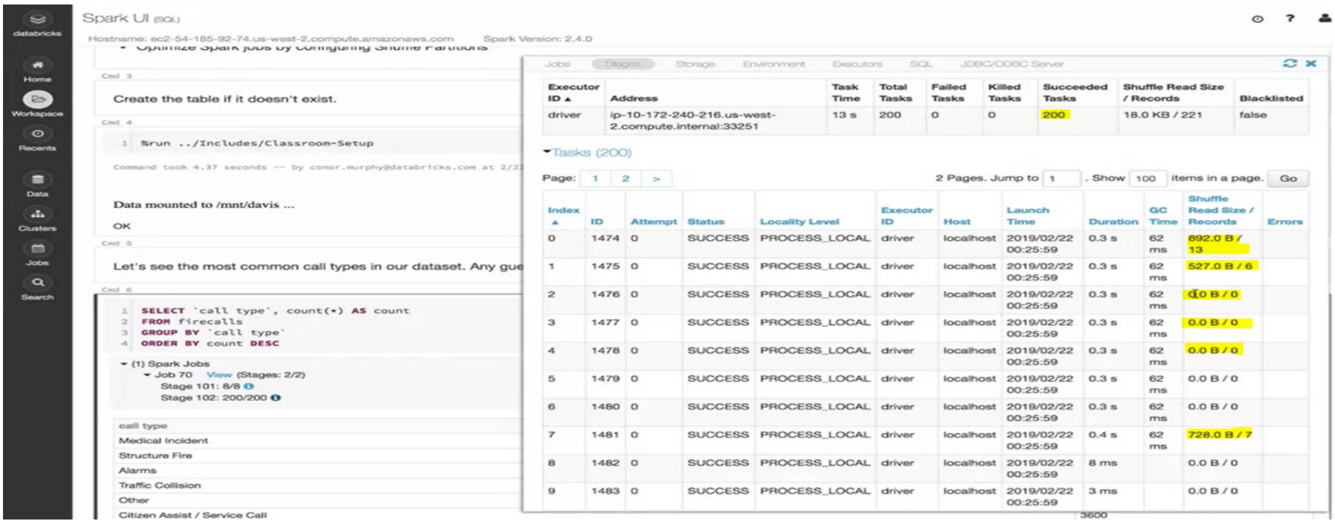


Wide transformation [cross-Partition]

- Aggregation operation – eg. Groupby, orderby
- Data shuffle required
- **Default shuffle partition is 200**
- **work → shuffle (write) results to local disc → shuffle (read) using 200 tasks → output formed aggregating 200 partitions**



spark.sql.shuffle.partitions parameter controls how many resulting partitions there are after a shuffle (wide transformation).



Broadcast Joins decrease data shuffle → decrease query time

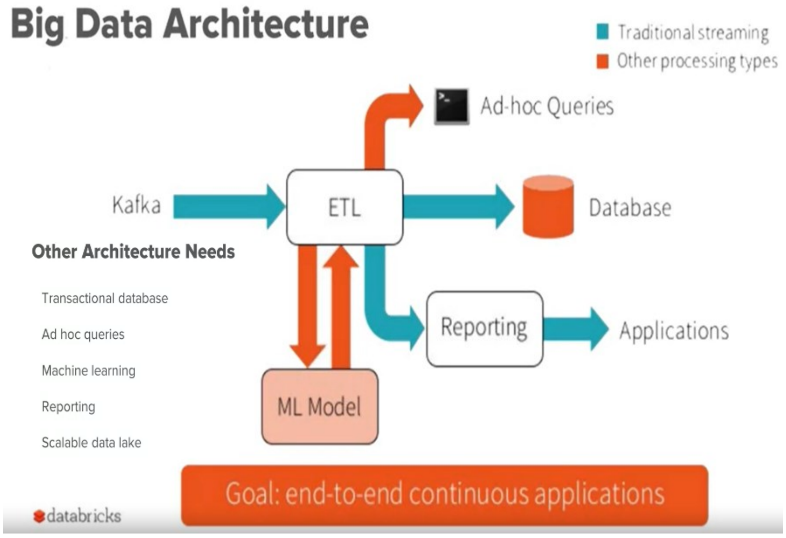
- Spark analyses **data size** and using this heuristics, it decides type of JOIN operation
  - Default size limit for opting broadcast is 10MB
    %python
    spark.conf.get("spark.sql.autoBroadcastJoinThreshold")
    >> ‘10485760b’
  - Can be enforced by :
    SELECT /*+ BROADCAST(fireCalls) */ *
    FROM < T1_name >
    JOIN <T2_name>on T1_name.dimension1 = T2_name.dimension2

Readings

- [Deep Dive into Spark SQL’s Catalyst Optimizer]
- [Cost Based Optimizer in Apache Spark 2.2]
- [Understanding your Apache Spark Application Through Visualization]

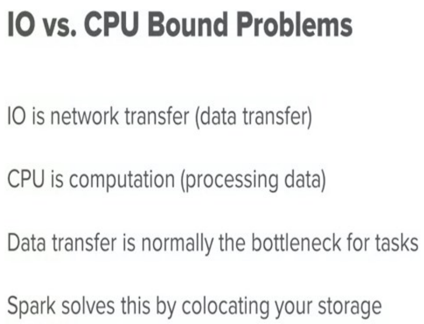
WEEK 3 ENGINEERING DATA PIPELINES - ETL, SCHEMAS, FILE FORMATS, JDBC PROTOCOL

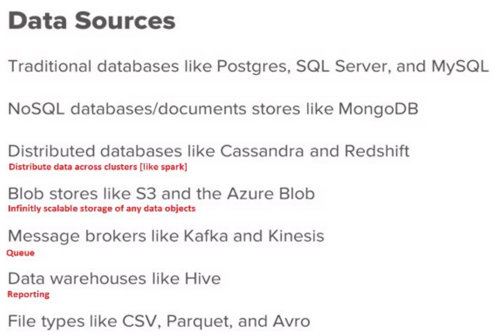
- **Kafka/ AWS Kinesis/ Azure event hub**
  - Queue : Arriving data should not overload other infra
  - Data producers write data into topics and consumers read them
- **ETL**
  - Raw data [Data link backed by blob store like AWS S3, Azure blob] → Process → Database
- **Decoupled storage and compute**
  - Spin spark cluster → connect data → shut down after use
  - No necessity of cluster version control



Computation bottlenecks

- Task : count no of records in a 10 GB DB
  Bottleneck : **fetching [IO]** 10Gb data from DB into cluster
- Task :Training ML model
  Bottleneck : threading capacity of **CPU**





Online Analytical processing : Analyze data for business needs [Share market trend of last day]
Online Transaction processing: Transaction against DB in Real time [update profile, pay to merchant]

Data Connection : Blob stores, JDBC
Spark uses-

- **Predicate push down :** Process predicate level SQL operations [filter] at database level before fetching data into cluster – **Save network transfers and IO bottlenecks**
- **J**ava **D**ata **B**ase **C**onnection : Java API

File formats:

- CSV Storage in row format
- Parquet : Storage in column format – FASTEST method to read/ write data

Compression types:

- gz
- bzip : very high storage efficiency

Spark is fast given its:

- **In memory computation**
- **Knowledge of Data types operated on –** it helps optimizing logic behind data access, partitioning and shuffling
- **Flexibility to work with Semi structured data [JSON, NOSQL] awa tabular data [CSVs, RDBs]**

Schemas and types:

- **JSON:** Data schema evolves over time
- **USER DEFINED schema:** Save efforts of spark in schema inference [an extra bottleneck job]

Database write & read operations:

- **Parallelized using more partitions**
  - When data is written to DB, 1 connection per partition exists
  - **COALESCE [narrow transformation]**
    CREATE OR REPLACE TEMPORARY VIEW <view1>
    AS
    SELECT /*+ COALESCE(1) */ *
    FROM <table1>
  - **Partition[wide transformation]**
    CREATE OR REPLACE TEMPORARY VIEW <view1>
    AS
    SELECT /*+ REPARTITION(8) */ *
    FROM <table1>

Managed and unmanaged tables

Use unmanaged/ external table when

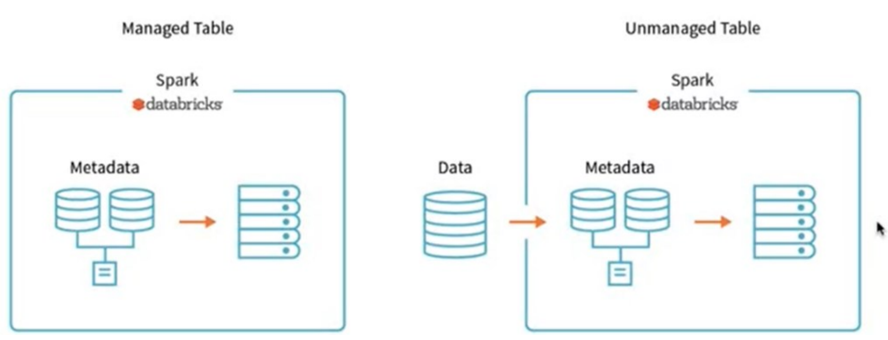
- we want data to persist after shutting cluster

Use managed table when

- Data is ephemeral

A **managed table** is a table that manages both the data itself as well as the metadata. In this case, a `DROP TABLE` command removes both the metadata for the table as well as the data itself.

Unmanaged tables manage the metadata from a table such as the schema and data location, but the data itself sits in a different location, often backed by a blob store like the Azure Blob or S3. Dropping an unmanaged table drops only the metadata associated with the table while the data itself remains in place.



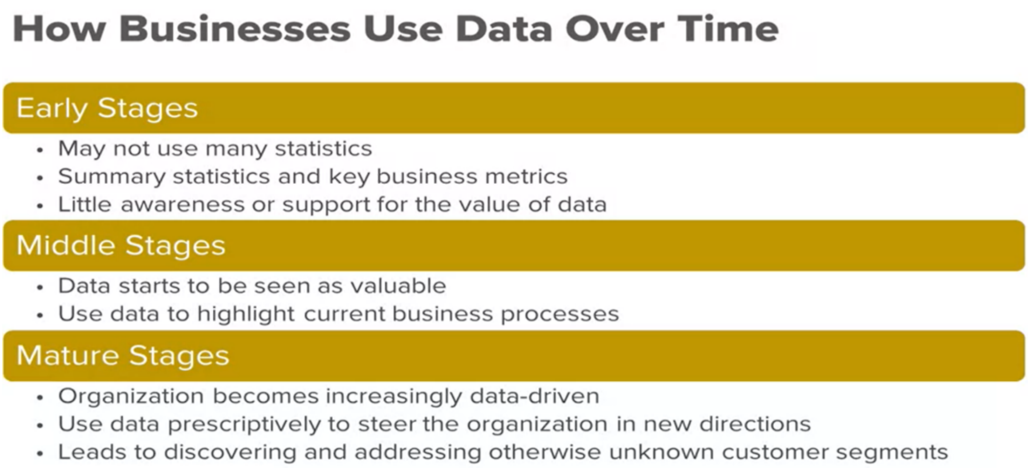
Module 3 Readings

- [Why You Should Care about Data Layout in the Filesystem]
- [Lessons From the Field: Applying Best Practices to Your Apache Spark Applications]
- [Working with Complex Data Formats with Structured Streaming in Apache Spark 2.1: Part 2 of Scalable Data at Databricks]

WEEK 4 REAL WORLD ML MODEL - REGRESSION AND CLASSIFICATION

TRANSALATING BUSINESS PROBLEM TO DATA PROBLEM IS MOST IMPORTANT 21 CENTURY BUSINESS PROBLEM

Spark applications : Graph processing, ML, streaming
ML applications:
Real time fraud detection
NLP : Classify medical records, chatbots
Computer vision : Medical image analysis, self-driven cars, facial recognition

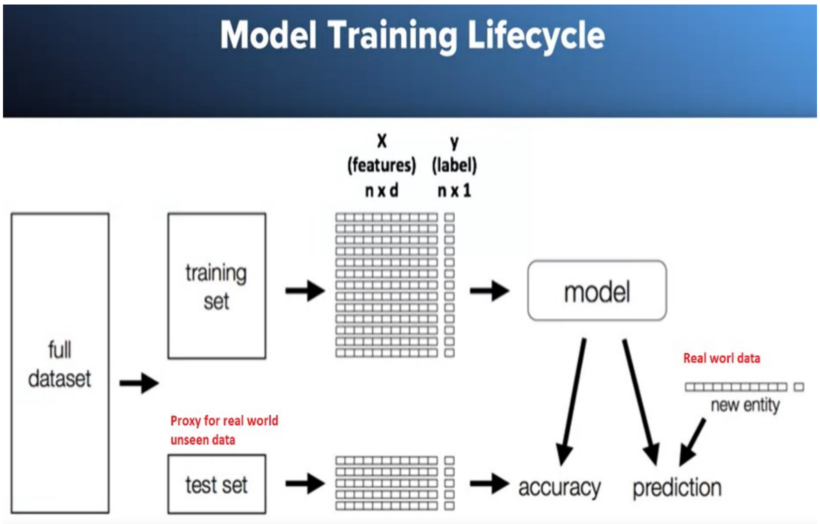


Churn analysis
What is churn? No purchase, No web visit
How to predict? Lesser visits, newer customers
Predict? Future purchases

Machine learning is array of techniques that learn patterns without explicit programming. It maps features [customer history] to outputs [probability of churn]

- Supervised learning : labeled data – regression[predict continuous value on scale], classification [predict label category]
- Unsupervised learning : unlabeled data – clustering [learn structure of data and reveal data segments]

Baseline model [like average value] is comparison benchmark for trained ML model



Choice of model

- Ask stakeholders if they want **interpretable model** [contribution of independent features to prediction – Decision tree, Linear regression] or Accurate model [NN]
- Assumptions made by algorithms about data [eg. Linear regression assumes linear relation between I/O]

SQL

1. USE <database>
2. DESCRIBE <tableName>
3. CREATE TABLE <tableName>
4. CREATE OR REPLACE TABLE <tableName>
5. CREATE OR REPLACE VIEW <viewName>
6. CREATE OR REPLACE TEMPORARY VIEW<tableName>
7. USING <filetype for creation>