

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

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

Spark Supports Many Languages

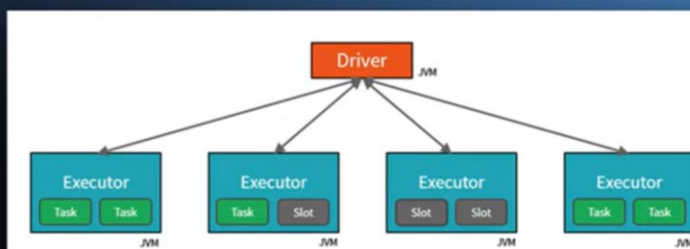
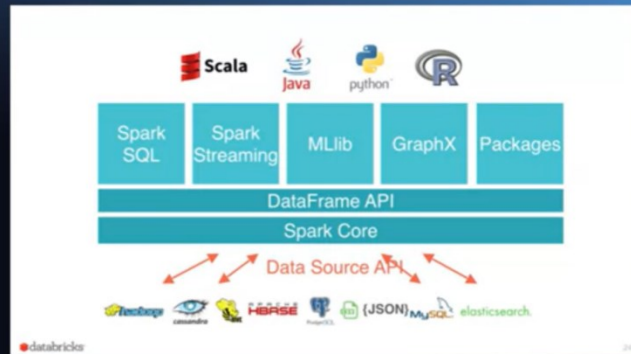
SQL

Python

Scala

Java

R



One driver

Optimizes queries

Delegates tasks



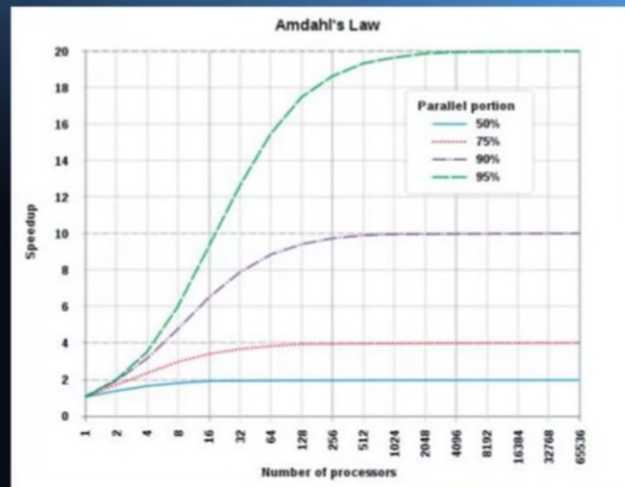
One or many executors

Perform actual queries

More is not always faster

Amdahl's Law

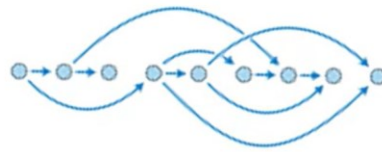
The amount of acceleration we would see from parallelizing a task is a function of what portion of the task can be completed in parallel



Resilient Distributed Datasets

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

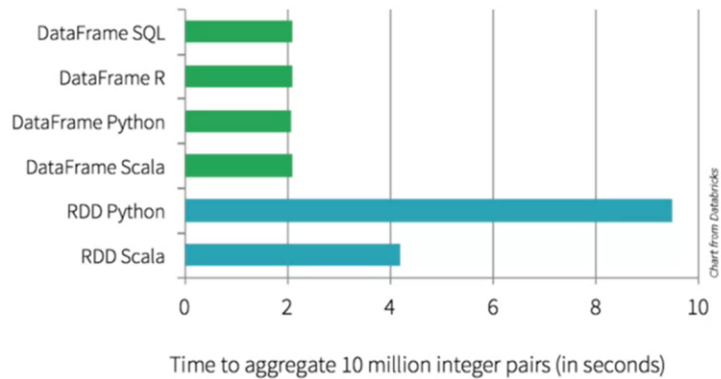
Directed Acyclic Graph



Series of transformations to apply to your data

However, you cannot change any of the transformations that came before you in this graph

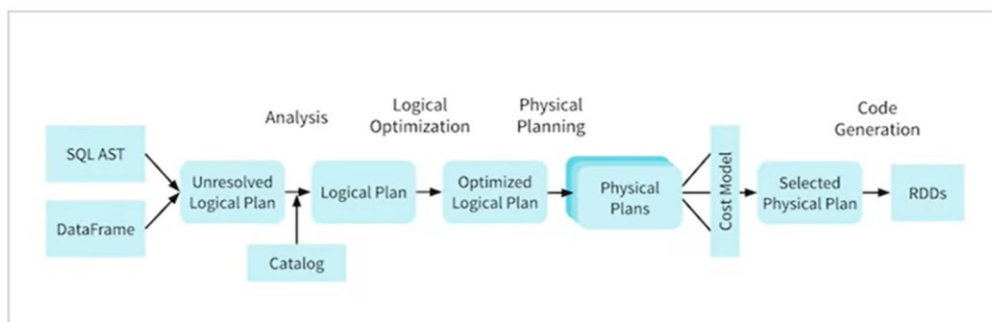
DataFrame > RDD



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

Spark DataFrame Execution

1. Unresolved logical plan before look-up in data catalog
2. Then Catalyst resolves them and creates a logical plan



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

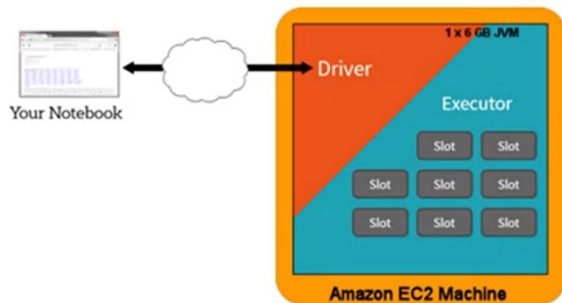
- [Spark SQL and DataFrames and Datasets Guides](#)
- [SQL Guide from Databricks](#)

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-Executor can be single Compute instance TDM



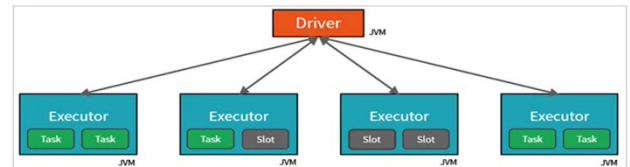
Slot is essentially thread of execution

- Machines * Cores * Threads

In production, Spark clusters runs multiple compute instances

Units of Parallelism within a Spark Cluster

4 executors * 2 slots = 8 units of parallelism



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

2.3-Caching (SQL)

Detached | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revision history

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

Storage

Parquet IO Cache

Data Read from External Filesystem	Data Read from IO Cache	Data Written to IO Cache	Estimated Size of Repeatedly Read Data	Cache Metadata Manager Peak Disk Usage
781.0 MiB	0.0 B	0.0 B	0.0 B (0 %) - 0.0 B (0 %)	0.0 B

Cmd 6

Count

Let's see how long it takes to count all of the records in our dataset.

Cmd 7

```
1 SELECT count(*) FROM fireCalls
```

▼ (1) Spark Jobs
► Job 19 View (Stages: 2/2)

count(1)
1 240613

Showing all 1 rows.

Command took 12.32 seconds -- by rasik.kane@ucdconnect.ie at 9/22/2020, 5

Cmd 10

1

CACHE TABLE fireCalls

▼ (1) Spark Jobs

► Job 22 View (Stages: 2/2)

OK

Command took 10.16 seconds -- by rasik.kane@udconnect.ie at 9/22/2020, 5:48:26 PM on My Cluster

Cmd 11

%fs ls /mnt/davis/fire-calls/fire-calls-truncated-comma.csv

Show cell

Cmd 12

%md ## Count (Again) Although it took a while to cache our dataset

Cmd 13

1

SELECT count(*) FROM fireCalls

▼ (1) Spark Jobs

► Job 23 View (Stages: 2/2)

	count(1)
1	240613

Showing all 1 rows.

Command took 8.46 seconds -- by rasik.kane@udconnect.ie at 9/22/2020, 5:48:36 PM on My Cluster

Jobs

Stages

Storage

Environment

Executors

SQL

JDBC/ODBC Server

Structured Streaming

Storage

Parquet IO Cache

Data Read from External Filesystem	Data Read from IO Cache	Data Written to IO Cache	Estimated Size of Repeatedly Read Data	Cache Metadata Manager Peak Disk Usage
866.5 MiB	0.0 B	0.0 B	0.0 B (0 %) - 0.0 B (0 %)	0.0 B

▼RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
131	In-memory table "databricks"."fireCalls"	Disk Memory Deserialized 1x Replicated	8	100%	59.6 MiB	0.0 B

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]

Cmd 16

Lazy Cache

Instead of waiting a minute to cache this dataset, we could do a "lazy cache". This means it will only cache the data as it is queried.

Cmd 17

1

UNCACHE TABLE fireCalls;

2

CACHE LAZY TABLE fireCalls;

OK

Command took 0.31 seconds -- by rasik.kane@udconnect.ie at 9/22/2020, 5:48:26 PM on My Cluster

Cmd 18

Small Query

This query was a lot faster to run. But, did it cache our entire dataset?

Cmd 19

1

SELECT * FROM fireCalls LIMIT 100

▼ (1) Spark Jobs

► Job 24 View (Stages: 1/1)

	Call Number	Unit ID	Incident Number	Call Type	Call Date	Watch Date	Received
1	1030110	E08	30625	Medical Incident	04/12/2000	04/12/2000	04/12/2000
2	1030122	M18	30630	Medical Incident	04/12/2000	04/12/2000	04/12/2000
3	1030154	M36	30662	Medical Incident	04/12/2000	04/12/2000	04/12/2000
4	1040007	E12	30697	Structure Fire	04/13/2000	04/13/2000	04/13/2000
5	1040021	M14	30711	Medical Incident	04/13/2000	04/12/2000	04/13/2000
6	1040061	M43	30749	Medical Incident	04/13/2000	04/12/2000	04/13/2000
7	1040079	E10	30766	Alarms	04/13/2000	04/13/2000	04/13/2000

Showing all 100 rows.

Command took 1.36 seconds -- by rasik.kane@udconnect.ie at 9/22/2020, 5:48:32 PM on My Cluster

Jobs

Stages

Storage

Environment

Executors

SQL

JDBC/ODBC Server

Structured Streaming

Storage

Parquet IO Cache

Data Read from External Filesystem	Data Read from IO Cache	Data Written to IO Cache	Estimated Size of Repeatedly Read Data	Cache Metadata Manager Peak Disk Usage
877.7 MiB	0.0 B	0.0 B	0.0 B (0 %) - 0.0 B (0 %)	0.0 B

▼RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
146	In-memory table "databricks"."fireCalls"	Disk Memory Deserialized 1x Replicated	1	13%	8.0 MiB	0.0 B

Cmd 20

Cache I

Now let's do a count() on our dataset. Count forces you to go through every record of every partition of our dataset, so it will cache the entire dataset.

Cmd 22

1

SELECT count(*) FROM fireCalls

▼ (1) Spark Jobs

► Job 25 View (Stages: 2/2)

	count(1)
1	240613

Showing all 1 rows.

Command took 8.25 seconds -- by rasik.kane@udconnect.ie at 9/22/2020, 5:50:04 PM on My Cluster

Cmd 23

Count

Look at how fast this call to count() is now!

Cmd 24

1

SELECT count(*) FROM fireCalls

▼ (1) Spark Jobs

► Job 26 View (Stages: 2/2)

	count(1)
1	240613

Showing all 1 rows.

Command took 0.07 seconds -- by rasik.kane@udconnect.ie at 9/22/2020, 5:51:04 PM on My Cluster

Jobs

Stages

Storage

Environment

Executors

SQL

JDBC/ODBC Server

Structured Streaming

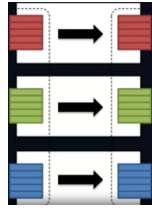
Storage

Parquet IO Cache

▼RDDs

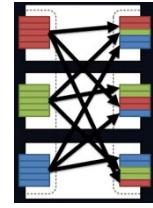
Narrow Transformation [In-Partition]

- 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).

Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Shuffle Write Size / Records	Errors
0	1474	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.3 s	62 ms	892.0 B / 13	0.0 B / 0	
1	1475	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.3 s	62 ms	527.0 B / 6	0.0 B / 0	
2	1476	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.3 s	62 ms	0.0 B / 0	0.0 B / 0	
3	1477	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.3 s	62 ms	0.0 B / 0	0.0 B / 0	
4	1478	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.3 s	62 ms	0.0 B / 0	0.0 B / 0	
5	1479	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.3 s	62 ms	0.0 B / 0	0.0 B / 0	
6	1480	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.3 s	62 ms	0.0 B / 0	0.0 B / 0	
7	1481	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	0.4 s	62 ms	728.0 B / 7	0.0 B / 0	
8	1482	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	8 ms		0.0 B / 0	0.0 B / 0	
9	1483	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/02/22 00:25:59	3 ms		0.0 B / 0	0.0 B / 0	

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
```

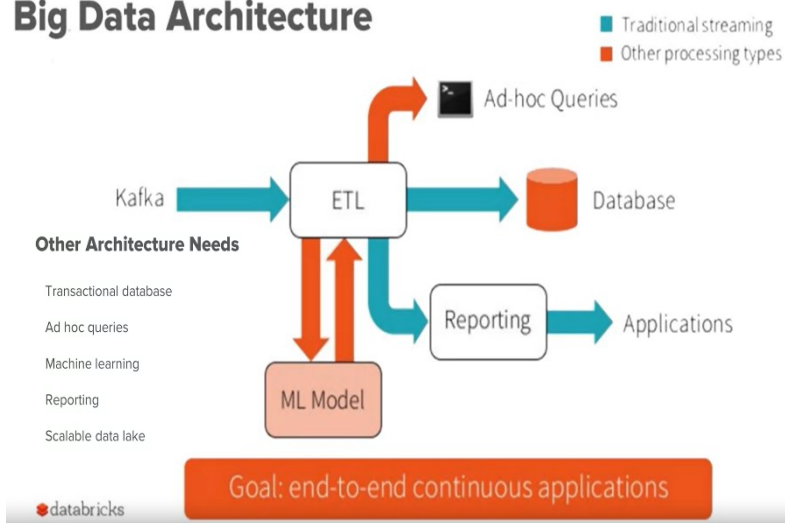
Readings

- [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

Big Data Architecture



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**

IO vs. CPU Bound Problems

IO is network transfer (data transfer)
CPU is computation (processing data)
Data transfer is normally the bottleneck for tasks
Spark solves this by colocating your storage

Data Sources

Traditional databases like Postgres, SQL Server, and MySQL
NoSQL databases/documents stores like MongoDB
Distributed databases like Cassandra and Redshift
Distribute data across clusters (like spark)
Blob stores like S3 and the Azure Blob
Infinitely scalable storage of any data objects
Message brokers like Kafka and Kinesis
Queue
Data warehouses like Hive
Reporting
File types like CSV, Parquet, and Avro

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**
- **Java Data Base Connection** : 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

TRANSLATING 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

How Businesses Use Data Over Time

Early Stages

- May not use many statistics
- Summary statistics and key business metrics
- Little awareness or support for the value of data

Middle Stages

- Data starts to be seen as valuable
- Use data to highlight current business processes

Mature Stages

- Organization becomes increasingly data-driven
- Use data prescriptively to steer the organization in new directions
- Leads to discovering and addressing otherwise unknown customer segments

Churn analysis

What is churn? No purchase, No web visit

How to predict? Lesser visits, newer customers

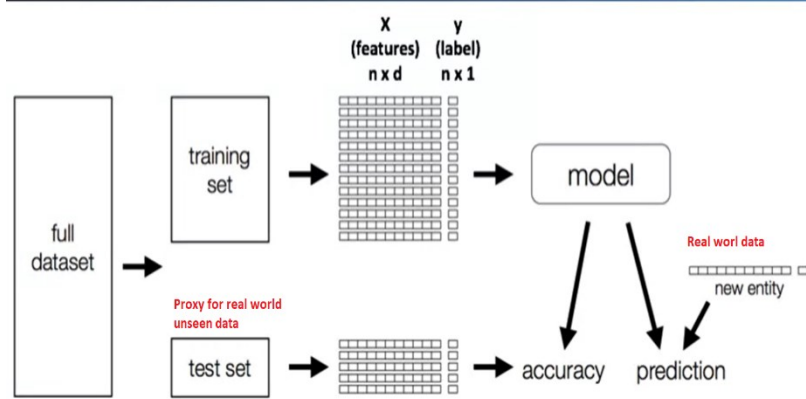
Predict? Future purchases

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

Model Training Lifecycle



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>**