Recitation 5: Spark

CIS 5450

Recitation Goals

- To expose you to the programming paradigms needed for Spark
- To teach you the Spark skills you need for HW3
- If we have time: more SQL exercises for HWs 2 and 3

What is Spark?

- Recall the old adage:
    - “**Many hands make light work**”

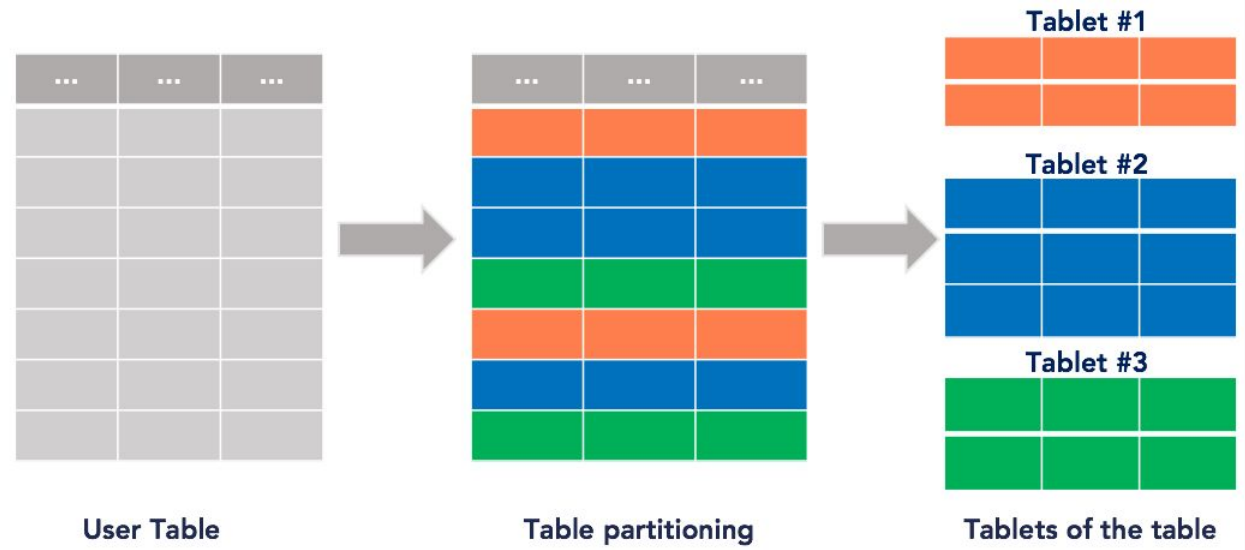
What is Spark?

- Recall the old adage:
  - “**Many hands make light work**”

- **Idea:** distribute work among multiple machines to reduce computation time & burden

What is Spark?

- Platform for distributed data processing
- Shards data onto different machines



Who uses PySpark?

Trivago

Computing value-per-click for ad auctions

Walmart

Forecasting anomalies in refrigeration in stores

Adidas Runtastic

Performing daily validation tests on incoming data

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

- Leslie Lamport (developer of LaTeX)

Dealing with failures

Problems with cluster computation

- Machines may fail / machines might not all be equally powerful
- Network I/O is expensive compared to computation (need to send data / coordinate work)

Dealing with failures

Problems with cluster computation

- Machines may fail / machines might not all be equally powerful
- Network I/O is expensive compared to computation (need to send data / coordinate work)

Solution

- Spark periodically checkpoints / snapshots what happened
- If a node dies, Spark can restart computation

Query optimization

- Spark SQL has a *query optimizer* called **Catalyst**

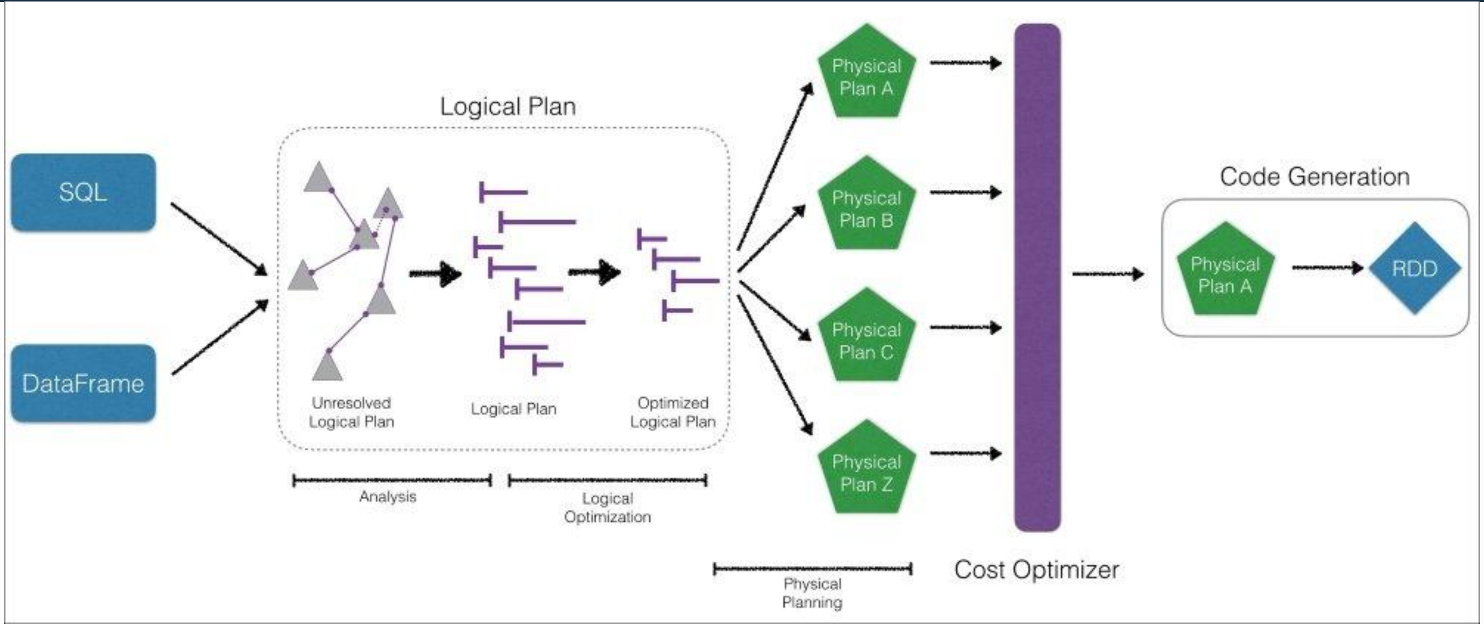
Query optimization

- Spark SQL has a *query optimizer* called **Catalyst**
    - How should we initially partition our data?
    - How can we minimize the number of shuffles?
    - What order should we apply the operators for maximum performance?

Query optimization

- Spark SQL has a *query optimizer* called **Catalyst**
    - How should we initially partition our data?
    - How can we minimize the number of shuffles?
    - What order should we apply the operators for maximum performance?
- Catalyst estimates the sizes of each computation & their runtime, and picks the best strategy

Query optimization



The Spark programming paradigm

"It is not only the violin that shapes the violinist, we are all shaped by the tools we train ourselves to use, and in this respect **programming languages have a devious influence: they shape our thinking habits.**"

- Edsger Dijkstra (of graph algorithms fame)

Lazy Computation

- Nothing gets computed until...
    - You explicitly ask Spark to **save**, **show** or **collect** the final answer

Lazy Computation

- Nothing gets computed until...
    - You explicitly ask Spark to **save**, **show** or **collect** the final answer

- Example:
    - Filtering 1 TB of census data and finding the first row corresponding to Chicago

Lazy Computation

- If Spark were to run each instruction on-the-go:
  - Load all the data → Filter for the Chicago records → Pick out the 1st record
  - Time-consuming!

Lazy Computation

- If Spark were to run each instruction on-the-go:
  - Load all the data → Filter for the Chicago records → Pick out the 1st record
  - Time-consuming!
- **Lazy evaluation:** Spark waits for all instructions to be specified before computing anything
  - Query optimizer can optimize wrt the entire sequence of instructions
  - Spark will just find the first Chicago record, then emit that as the answer

Typed schemas in Spark

- Pandas can automatically infer the type of columns on-the-fly
- Spark can automatically infer types – but this can be faulty / slow!
    - Solution: manually define a typed schema

Typed schemas in Spark

Suppose we have a CSV of US census data:

```
2019_rank|City|State_Code|2019_estimate|2010_Census|Change
1|New York[d]|NY|8336817|8175133|0.0198
2|Los Angeles|CA|3979576|3792621|0.0493
```

Typed schemas in Spark

Suppose we have a CSV of US census data:

```
2019_rank|City|State_Code|2019_estimate|2010_Census|Change
1|New York[d]|NY|8336817|8175133|0.0198
2|Los Angeles|CA|3979576|3792621|0.0493
```

Should “8336817” be interpreted as a:

- String?
- Integer?
- Float?

Typed schemas in Spark

Letting Spark automatically infer column types: **1.75s**

Cmd 4

With InferSchema

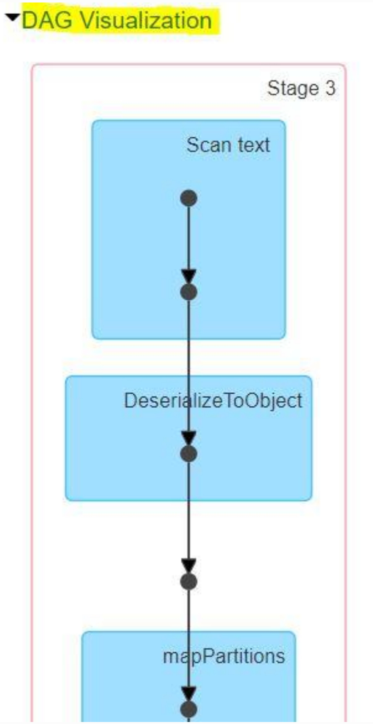
```
df = spark.read.option("header", True) \
  .option("delimiter", "|") \
  .option("inferSchema", True) \
  .csv(file_location)
```

▾ (2) Spark Jobs
- ▸ Job 2 View (Stages: 1/1)
- ▸ Job 3 View (Stages: 1/1)

▾ df: pyspark.sql.dataframe.DataFrame

```
2019_rank: integer
City: string
State_Code: string
2019_estimate: integer
2010_Census: integer
Change: string
```

Command took 1.75 seconds -- by azar.s91@gmail.com at 10/25/2020,



Typed schemas in Spark

Manually defining a typed schema beforehand: **0.42s**

- Spark parses the data nearly 4 times quicker!

```
from pyspark.sql.types import *

sch=StructType([
  StructField("2019_rank",IntegerType(),True),
  StructField("City",StringType(),True),
  StructField("State_Code",StringType(),True),
  StructField("2019_estimate",IntegerType(),True),
  StructField("2010_Census",IntegerType(),True),
  StructField("Change",StringType(),True),
])

df = spark.read \
  .option("header", True) \
  .option("delimiter", "|") \
  .schema(sch) \
  .csv(file_location)
```

```
df: pyspark.sql.dataframe.DataFrame
    2019_rank: integer
    City: string
    State_Code: string
    2019_estimate: integer
    2010_Census: integer
    Change: string
```

```
Command took 0.42 seconds -- by azar.s91@gmail.com at 10/25/2020, 4:07:56 PM on learntospark
```

Typed schemas in Spark

- Schemas are defined using **StructType** objects
  - StructType object = collection of **StructField**s that specify the structure & type of each column

Typed schemas in Spark

- Schemas are defined using **StructType** objects
  - StructType object = collection of **StructField**s that specify the structure & type of each column

- Example types:
  - StringType
  - IntegerType
  - FloatType
  - BooleanType
  - DateType
  - TimestampType

Typed schemas in Spark

- Each StructField has the form **(name, type, nullable)**.
- Nullable flag defines that the specified field may be empty
- See examples on the following slides

Typed schemas in Spark

Suppose we have the following JSON data:


```
{
 "student_name": "Data Wrangler",
 "GPA": 1.4,
 "courses": [
    {"department": "Computer and Information Science",
     "course_id": "CIS 545",
     "semester": "Fall 2021"},
    {"department": "Computer and Information Science",
     "course_id": "CIS 555",
     "semester": "Fall 2021"}
 ],
 "grad_year": 2022
 }
```

Typed schemas in Spark

One would define its typed schema like so:

```
schema = StructType([
           StructField("student_name", StringType(), nullable=True),
           StructField("GPA", FloatType(), nullable=True),
           StructField("courses", ArrayType(
                StructType([
                  StructField("department", StringType(), nullable=True),
                  StructField("course_id", StringType(), nullable=True),
                  StructField("semester", StringType(), nullable=True)
                ])
           ), nullable=True),
           StructField("grad_year", IntegerType(), nullable=True)
         ])
```

Programming for Spark

- Prefix each Colab cell with **%%spark**
    - Tells Colab that the cell should be *executed remotely* on a Spark machine

- my_df.**createOrReplaceTempView**("my_sql_table")
    - Creates a temporary view of a Pandas df, allowing you to run SQL queries
- sqlContext.sql("select * from my_sql_table")

Programming for Spark

- More coding examples in the associated Colab Notebook

Pandas vs Spark

In practice,
when should one use Spark over Pandas
& vice versa?

Three considerations:

1. Memory
2. Types of Data Analytics / Transformations
3. Developer productivity

Pandas vs Spark: Memory

“My rule of thumb for Pandas is that **you should have 5 to 10 times as much RAM as the size of your dataset**. So if you have a 10 GB dataset, you should really have about 64, preferably 128 GB of RAM if you want to avoid memory management problems”

- Wes McKinney, creator of Pandas

Pandas vs Spark: Memory

- Pandas has a strict memory limit:
  - **MemoryError**: when Pandas is unable to allocate enough memory to store a dataframe
  - Process gets killed immediately

- If your dataset size is 10s / 100s of GBs (or more), consider using Spark (on Elastic MapReduce / Livy)
  - See Module 9 for details

Pandas vs Spark: Analytics

- What type of data transformations are you planning on doing?

- Spark may be more suitable for:
    - Computing summary statistics
    - Counts / ratios / transformations expressible using SQL

- Pandas may be more appropriate for:
    - Complex/interactive visualizations with Seaborn / Plotly
    - Pivoting tables, data wrangling, resampling

Pandas vs Spark: Developer productivity

- Pandas can be used out-of-the-box
- Pandas is typically easier to debug

- Do you have time to set up a Spark development environment?
  - Need to set up a EMR compute cluster
  - Need to wait for resources to be provisioned

Pandas vs Spark

	Spark	Pandas
Memory	No memory limit	Memory limit of 100GB
Types of Data Analytics/ Transformations	Computing summary statistics Counts / ratios / transformations expressible using SQL	Complex/interactive visualizations with Seaborn / Plotly Pivoting tables, data wrangling, resampling
Developer Productivity	More involved 👹	Easy

PySpark Example

See associated Colab notebook

Fin

- Start Homework 2 early!
- Come to office hours!