Distributed Data Processing

Spark

- Apache Spark is a fast and general-purpose cluster computing system
- It processes data in streams or batches
- Typically runs on Hadoop clusters
- Offers multiple extensions
  - SparkQL
  - MLLib
  - GraphX

How does Spark work?

- You build an execution graph
- Calculations are not done until you explicitly ask for the results
- Spark executes the graph on the provided cluster and given data
  - Runs operations in parallel
  - Moves data between nodes
  - Optimizes operations

Spark - basic concepts

- Resilient Distributed Dataset (RDD) - low level dataset
  - “a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel”
  - Offers more control over transformations
  - Good for unstructured data
- Dataset/DataFrame - higher level data abstraction
  - Simplifies data operations
- Spark Session - a unified entry point for manipulating data with Spark
  - For RDD you should use Spark Context which is embedded within session

Spark - starting session

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
          .master("local") \
          .appName("App Name") \
          .config("spark.some.config.option", "some-value") \
          .getOrCreate()
```

Spark - loading data

```
df = spark.read.csv("path_to_file.csv")
df = spark.read.json("path_to_file.json")
df = spark.read.text("path_to_file.txt")
df = spark.read.parquet("path_to_file")
...
```

Spark - basic operations on DataFrame

- Similar to Pandas dataframes - but there are some subtle differences
- A lot of operations possible
  - Best to refer to API: http://spark.apache.org/docs/2.2.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame
- Operations that may be useful for assignment:

```
df.printSchema() # prints schema of the df
df.show(n) # shows top n rows (by default 20)
df.alias("alias") # adds an alias to df, useful for joins
df.count() # displays number of rows
df.where(condition) # selects only rows that meet condition (df.c == 2 or "df.c == 2")
df.withColumn("new_column_name", udf_function(column_to_work_on))
df.join(df2, df.col1 == df2.col1, "type_of_join")
df.groupBy(["column_name",...])
df.orderBy(["column_name",...],ascending=[is_ascending,...])
```

Spark UDFs

- UDF - User Defined Function
- Useful for more complex calculations
- A python function must be wrapped in **`udf`** function provided by spark
- The return type of the function must be specified

```
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

def a_function(value):
  return results_of_some_complex_calculations

a_function_udf = udf(a_function, IntegerType())
```

Spark - SQL support

- Spark offer Spark SQL a way to query data frames using SQL syntax
- Results of spark.sql calls are returned as spark DataFrame
- To access df a temporary view must be created

```
df.createOrReplaceTempView("table1")
new_df = spark.sql("SELECT * FROM table1")
```