# Data Engineering in the Cloud

## Apache Spark - Part II

Xuemao Zhang
East Stroudsburg University

January 18, 2025

# Outline

- Lab 1: Playing with RDDs
- Lab 2: Creating Dataframes

# Lab 1

- install Apache Spark with Hadoop

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
# you may try the latest version of Spark and  Hadoop
```

!wget
https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.7.tgz

- Now, we just need to unzip that folder.

```
!tar -xvzf spark-3.0.0-bin-hadoop2.7.tgz
```

```
!pip install -q findspark
```

```
import os
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop2.7"
import findspark
findspark.init()
```

# Lab 1

- Create SparkSession and SparkContext objects

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Spark Demo App")\
.master("local").getOrCreate()

print(type(spark))
sc = spark.sparkContext
print(type(sc))
```

# Lab 1

## Creating RDDs

- Spark provides two ways to create RDDs:
  - loading an external dataset and
  - parallelizing a collection in your driver program.
- Creating from Collection using SparkContext's `parallelize()` method

```
L = [1,2,3,4,5]
rdd = sc.parallelize(L)
print(type(rdd))
```

# Lab 1

- Creating RDD from External Source
- Text files are very simple to load from and save to with Spark. When we load a single text file as an RDD, each input line becomes an element in the RDD.

!wget –continue https://raw.githubusercontent.com/esumath/Math_418/main/data/users_01.dat -O users.dat

```
fileRdd = sc.textFile("users.dat")
print(type(fileRdd))
```

- `collect()` brings all the data to the driver node.

```
for x in fileRdd.collect():
  print(x)
```

# Lab 1

- Multiple methods are available to save RDD data to various formats or storage systems.
  - saveAsTextFile(path):Saves the RDD as a text file, where each element of the RDD is converted to a line of text.
  - saveAsSequenceFile(path): Saves the RDD as a Hadoop SequenceFile, which is a flat file consisting of binary key-value pairs.
  - saveAsObjectFile(path): Saves the RDD as a serialized Java object file using Java serialization.
  - saveAsTable(tableName): Saves the RDD to a Hive table (if using Spark with Hive support), which allows querying data using HiveQL.
- Google Colab's environment doesn't directly support Hadoop Distributed File System (HDFS) operations or saving directly to the local file system in the way you might expect in a traditional Spark setup.

# RDD Operations

- RDDs support two types of operations:
- transformations
- actions

```
L = list(range(1,101))
rdd = sc.parallelize(L)
```

# RDD Transformation

- The `.map(func)`: THe `map(func)` returns a new RDD by applying a function to each element of a parent RDD

```
result = rdd.map(lambda x:x*3)
result.take(20) #Taking Elements with take()
```

- The `.filter(predicateFn)`: The `.filter` returns a new RD containing only the elements in the parent RDD satiesfies the function inside the filter

```
rdd_01 = result.filter(lambda x:x%3==0 and x%5==0)
rdd_01.take(10)
```

# RDD Actions

- The `.take(num)`: fetch the first `num` records

```
for record in rdd.take(5):
  print(record)
```

- The `.collect()`: It print all the elements of an RDD

```
for record in rdd.collect():
  print(record)
```

# Data Analysis using RDD's

- Print the first 5 record

```
user_rdd = sc.textFile("users.dat")
for record in user_rdd.take(5):
  print(record)
```

```
header = user_rdd.first()
print(header)
```

# Data Analysis using RDD's

- A **lambda function** in Python is a small anonymous function defined with the lambda keyword. It can have any number of arguments, but only one expression.

```
# Example of a lambda function that squares a number
square = lambda x: x ** 2
print(square(5))  # Output: 25
```

```
## 25
```

- In Apache Spark, specifically when working with RDDs (Resilient Distributed Datasets) in Python, lambda functions are often used with RDD transformations like map(), filter(), reduce(), etc. These functions help in performing distributed operations on the elements of the RDD.

# Data AnNalysis using RDD's

```
print(header)
```

```
user_rdd_wo_header = user_rdd.filter(lambda record:record!=header)
for record in user_rdd_wo_header.take(5):
  print(record)
```

# Data Analysis using RDD's

- Transformation `map()` is used to transform each element of an RDD using a specified function and produce a new RDD.
  - It applies the function to each element independently and returns a new RDD consisting of the results of applying the function to each element.
  - Input: a function `func`
  - Output: Returns a new RDD where each element is the result of applying `func` to the corresponding element of the original RDD.

# Data Analysis using RDD's

- Print only the "id, asset"

```
# SQL cannot be used
rdd = user_rdd_wo_header.map(lambda record:record.\
split("|")[0]+"|"+record.split("|")[5])
```

```
for record in rdd.collect():
  print(record)
```

- `record.split("|")` splits each record into a list based on the | delimiter.
- `+ "|" +` concatenates the two extracted fields back into a string with | as a delimiter.
- `map()` applies the function `(lambda record:record.split("|")[0]+"|"+record.split("|")[5])` to each element/row (record) in the RDD (user_rdd_wo_header).

# Data Analysis using RDD's

- Select the id,assest_col_in_million

```
def parse_record(record):
  fields = record.split("|")
  return fields[0]+"|"+str(int(fields[5])/1000000)
```

```
rdd = user_rdd_wo_header.map(lambda record:parse_record(record))
for record in rdd.take(10):
  print(record)
```

- The transformation `map(parse_record)` applies the parse_record function to each record in user_rdd_wo_header.

# Data Analysis using RDD's

- Round of the asset to two decimals

```
def parse_record(record):
  fields = record.split("|")
  return fields[0]+"|"+str(round(int(fields[5])/1000000,2))
```

```
rdd = user_rdd_wo_header.map(lambda record:parse_record(record))
for record in rdd.take(10):
  print(record)
```

# Data Analysis using RDD's

- The function `create_pair_rdd` takes a record, splits it, and returns a tuple `(field[3], 1)`.
  - A **Pair RDD** is an RDD where each element is a tuple of the form (key, value).
- Use `map()` to apply `create_pair_rdd` to each record in the RDD.

```
def create_pair_rdd(record):
  fields = record.split("|")
  return (fields[3], 1)

rdd2 = user_rdd_wo_header.map(\
lambda record: create_pair_rdd(record))
rdd2.collect()
```

# Data Analysis using RDD's

- Collect values for the same key in an array
  - `mapValues(list)`: Applies the `list` function to each value in the RDD. The `list` function converts the iterable of values into a list.

```
rdd2.groupByKey().mapValues(lambda x:list(x)).collect()
```

- or use the `lambda` function

```
rdd2.groupByKey().mapValues(lambda x:list(x)).collect()
```

# Data Analysis using RDD's

- Use `reduceByKey` to sum the counts for each key.
  - the `reduceByKey` transformation is used to sum the values (counts) for each key in `rdd2`.

```
rdd2.reduceByKey(lambda x,y:x+y).collect()
```

# Lab 2: Creating Dataframes

- All methods to creat a dataframe is present insider DataFrameReader. To get a DataFrameReader object

```
dfr=spark.read
```

- File formats can be read include
  - .csv files
  - .json files
  - .jdbc files
  - .parquet files

# Lab 2: Creating Dataframes

- install Apache Spark

```
#install Apache Spark 3.0.1 with Hadoop 2.7 from here.
```

!wget
https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.7.tgz

```
# Now, we just need to unzip that folder.
!tar -xvzf spark-3.0.0-bin-hadoop2.7.tgz
!pip install findspark

import os
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop2.7"
import findspark
findspark.init()
```

# Lab 2: Creating Dataframes

- Initialize SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession.builder\
    .appName("dataframe examples")\
    .getOrCreate()
```

- Access DataFrame Reader

```
dfr = spark.read
```

- Reading from a CSV file https://raw.githubusercontent.com/esumath/Math_418/main/data/employee_skills.csv

!wget –continue https://raw.githubusercontent.com/esumath/Math_418/main/data/employee_skills.csv -O employee_skills.csv

# Lab 2: Creating Dataframes

```
df1 = dfr.csv("employee_skills.csv", header=True, inferSchema=True)
```

```
df1.printSchema()
```

```
df1.show()
```

# Lab 2: Creating Dataframes

- Reading from a JSON file https://jsonplaceholder.typicode.com/posts

!wget –continue https://jsonplaceholder.typicode.com/posts -O posts.json

- What error message do you get?

```
df2 = dfr.json("posts.json")

df2.show(truncate=True)
```

# Lab 2: Creating Dataframes

```
import json # import module json
```

```
# Read the JSON file into a Python dictionary
with open('posts.json', 'r') as f:
    json_data = json.load(f)
```

```
# Serialize the dictionary back into a single-line JSON string
single_line_json = json.dumps(json_data)
```

# Lab 2: Creating Dataframes

```
# Save the single-line JSON string to a new file
with open('posts_single_line.json', 'w') as f:
    f.write(single_line_json)
```

```
# Verifying the contents of the new file
with open('posts_single_line.json', 'r') as f:
    print(f.read())
```

# Lab 2: Creating Dataframes

```
# Read JSON file into DataFrame
df2 = dfr.json("posts_single_line.json")
```

```
df2.printSchema()
```

```
df2.show(truncate=True)
```

# Lab 2: Creating Dataframes

- If there are corrupt record in a JSON file

```
df3 = dfr\
    .option("inferSchema", "true")\
    .option("mode", "PERMISSIVE")\
    .option("columnNameOfCorruptRecord", "corrupt_record")\
    .json("posts_single_line.json")
```

```
# Filter out corrupt records (if any)
corrupt_records_count = df3.filter(df3["corrupt_record"].isNotNull()).count
# error message shows there are no corrupt records
if corrupt_records_count > 0:
    print(f"Number of corrupt records: {corrupt_records_count}")
    df3.filter(df3["corrupt_record"].isNotNull()).show(truncate=False)
```

# Lab 2: Creating Dataframes

```
df3.printSchema()
df3.show(truncate=True)
```

# Lab 2: Creating Dataframes

- Stop Spark Session if no longer needed

```
spark.stop()
```

# License