`pyspark`: RDDs

Justin Post

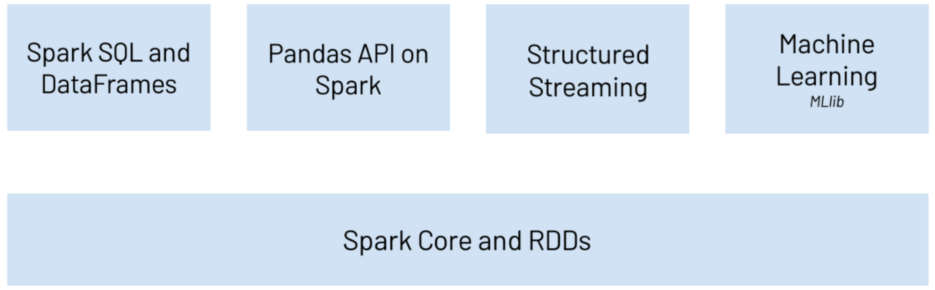
Spark Recap

Spark - Distributed processing software for big data workloads

- Generally faster than Hadoop's MapReduce (and much more flexible)
- DAGs make it fault tolerant and improve computational speed

Five major parts to (py)Spark

- Spark Core and RDDs as its foundation
- Spark SQL and DataFrames
- Pandas on Spark
- Spark Structured Streaming
- Spark Machine Learning (MLlib)



Starting a Spark Instance

- Use `pyspark.sql.SparkSession` to create a spark instance (or link to an existing one)

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master('local[*]').appName('my_app').getOrCreate()
```

- `local[*]` implies we are using a local machine
- Could use a URL instead to connect to a spark session already created
- App name useful if you have multiple spark processes running

Three Major Data Structures

- RDD
    - Usually hidden underneath
    - Not as user friendly
- Spark SQL Data Frame
    - Use SQL style code to interact with the data
- pandas-on-spark Data Frame
    - Use pandas style code to interact with the data

Resilient Distributed Datasets (RDDs)

- Can create explicitly using the `.sparkContext.parallelize()` method

```
#create some 'data' to put into an RDD
quick_cat = lambda x: "a" if x < 20 else "b"
my_data = [(quick_cat(x), x) for x in range(1,51)]
my_data[:3]
```

```
## [('a', 1), ('a', 2), ('a', 3)]
```

Resilient Distributed Datasets (RDDs)

- Can create explicitly using the `.sparkContext.parallelize()` method

```
#create some 'data' to put into an RDD
quick_cat = lambda x: "a" if x < 20 else "b"
my_data = [(quick_cat(x), x) for x in range(1,51)]
my_data[:3]
```

```
## [('a', 1), ('a', 2), ('a', 3)]
```

```
#spark session available through spark object
my_rdd = spark.sparkContext.parallelize(my_data)
my_rdd
```

```
ParallelCollectionRDD[1] at readRDDFromFile at PythonRDD.scala:274
```

RDD objects

- RDD object stored over multiple partitions

```
my_rdd.getNumPartitions()
```

```
128
```

RDD objects

- RDD object stored over multiple partitions

```
my_rdd.getNumPartitions()
```

```
128
```

- Doesn't return data unless you ask it to!

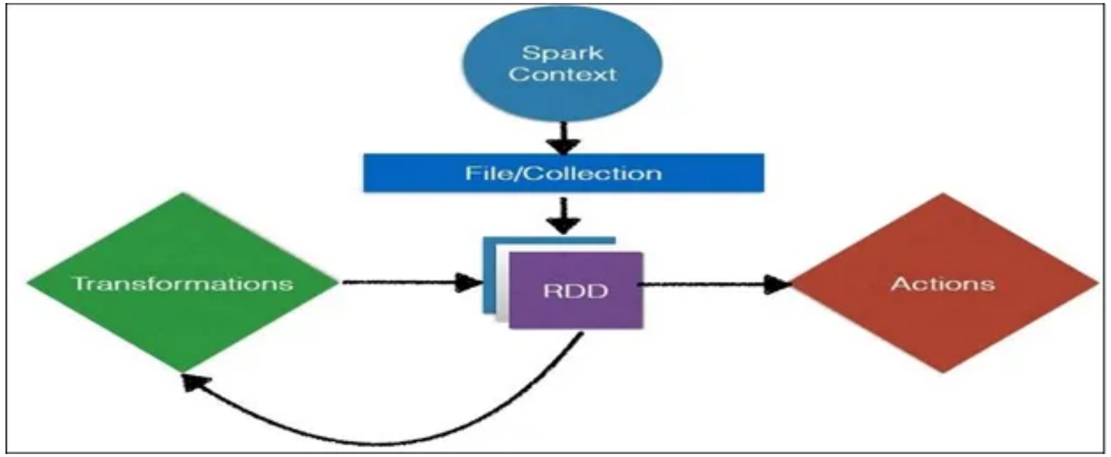
```
my_rdd.take(3)
```

```
[('a', 1), ('a', 2), ('a', 3)]
```

Resilient Distributed Datasets (RDDs)

Two types of operations (see here for reference):

- Transformation: Something that creates a new RDD
- Action: Operation applied to an RDD that performs a computation and sends the result back



Actions on RDDs

- Action: Operation applied to an RDD that performs a computation and sends the result back

Must perform an **action** to actually see the data:

- `.collect()`, `.take()`, & `.first()`
- `.reduce()`
- `.count()`, `.min()`, `.max()`, `.countByKey()`
- `.aggregate()`, `.foreach()`

```
my_rdd.take(3)    #specify how many to take
my_rdd.collect() #would return everything
my_rdd.first()    #just the first
```

Actions on RDDs

- Action: Operation applied to an RDD that performs a computation and sends the result back

Must perform an **action** to actually see the data:

- `.collect()`, `.take()`, & `.first()`
- `.reduce()`
- `.count()`, `.min()`, `.max()`, `.countByKey()`
- `.aggregate()`, `.foreach()`

```
my_rdd.count()
```

```
50
```

```
my_rdd.countByKey()
```

```
defaultdict(int, {'a': 19, 'b': 31})
```

Transformations on RDDs

Common transformations:

- `map()` (and `mapValues()` for key/value pairs)
  - Apply a function to each RDD (or just the values) return an RDD of the same *structure*
- `flatMap()` (and `flatMapValues()` for key/value pairs)
  - Apply a function to each RDD (or just the values) return an RDD with more or less elements
- `filter()` return a subsetted RDD

Usually write `lambda` functions with these!

- Often include things like `.groupBy()` or `.groupByKey()`

Transformations on RDDs

- Find the number of values for each key (alternative to `.countByKey()`)

```
#same calculation as before but returned as an RDD
my_rdd \
    .groupByKey() \
    .mapValues(len) \
    .collect()
```

```
[('b', 31), ('a', 19)]
```

Transformations on RDDs

- Find the number of values for each key (alternative to `.countByKey()`)

```
#same calculation as before but returned as an RDD
my_rdd \
    .groupByKey() \
    .mapValues(len) \
    .collect()
```

```
[('b', 31), ('a', 19)]
```

- Allows us to do more transformations!

```
from numpy import log
my_rdd \
    .groupByKey() \
    .mapValues(len) \
    .map(lambda x: (x[0], x[1], log(x[1]))) \
    .collect()
```

```
[('b', 31, 3.4339872044851463), ('a', 19, 2.9444389791664403)]
```

RDD Transformations & Actions

Tough to use many of the functions!

- Suppose we want to find the total sum for each key of our original RDD

```
my_rdd \
    .groupByKey() \
    .mapValues(sum) \
    .collect()
```

```
[('b', 1085), ('a', 190)]
```

- Documentation says use `aggregateByKey()` instead...

RDD Transformations & Actions

Tough to use many of the functions!

- Suppose we want to find the total sum for each key of our original RDD

```
#Combine values on the same partition first, then across partitions
my_rdd \
  .aggregateByKey(0, #initial value for each partition
                  lambda within_1, within_2: within_1 + within_2,
                  lambda across_1, across_2: across_1 + across_2) \
  .collect()
```

```
[('b', 1085), ('a', 190)]
```

To Jupyter Lab

- Working with RDDs can be tough!
- `DataFrames` (two types: SQL or pandas-on-spark) are much easier!
- DAG idea holds for all
    - Transformations
    - Actions
- Let's do a quick MapReduce example explicitly using RDDs in pyspark

Recap

- Use `SparkSession` to use spark
- RDDs are the underlying data structure
    - Perform transformations & actions
- Can be difficult to work with! Focus on `DataFrames` instead!