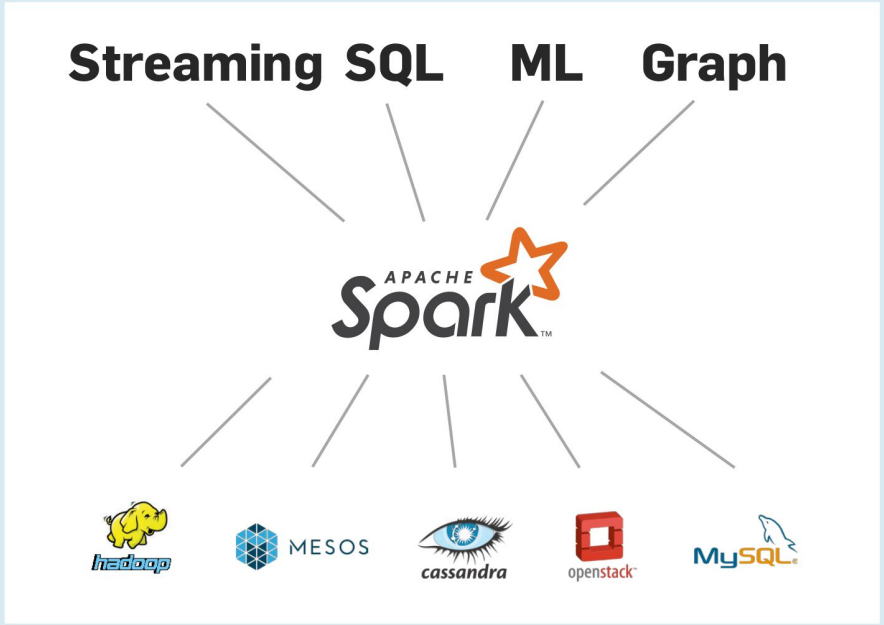
COMP6231 - Lab 10
PySpark DataFrames

Recap: Spark

- Spark is a framework for distributed computing.
- Built around a data structure: RDD. RDDs are fault-tolerant collections of objects partitioned across a cluster that can be manipulated in parallel.

Figure 1. Apache Spark software stack, with specialized processing libraries implemented over the core engine.



Recap: RDD transformations and actions

Transformations Return an RDD	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
Actions Return something else	$count()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Long}$
	$collect()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]}$
	$reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{T}$
	$lookup(k : \mathrm{K})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]}$ (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	:	Outputs RDD to a storage system, *e.g.*, HDFS

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

(K, V) is a key-value pair
MapReduce = flatMap + reduceByKey

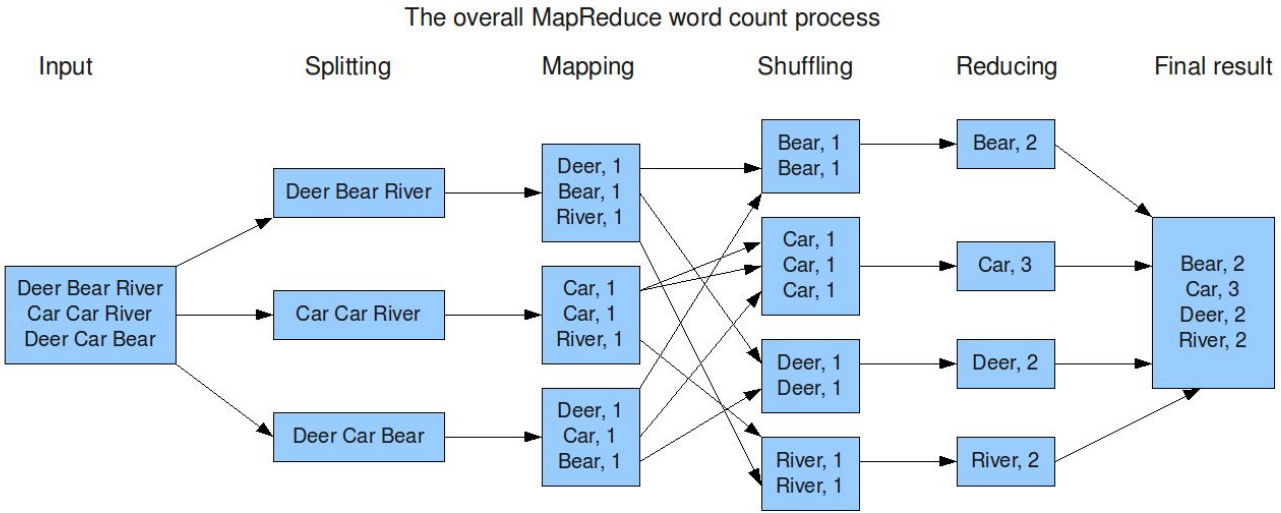
Recap: PySpark Hello World

```
import pyspark
sc = pyspark.SparkContext('local[*]')

txt = sc.textFile('sample.log')

error_lines = txt.filter(lambda line: 'error' in line)

print(error_lines.count())
```

Credit: https://realpython.com/pyspark-intro/

Recap: MapReduce

- MapReduce is a programming model for processing large data sets in a distributed fashion over a several machines.



Credit: https://nyu-cds.github.io/python-bigdata/02-mapreduce/

Recap: Example MapReduce: Word Count (2)

Input:

```
From fairest creatures we desire increase
That thereby beauty's rose might never die
But as the riper should by time decease
His tender heir might bear his memory
But thou contracted to thine own bright eyes
```

Output:

```
From, 1
fairest, 1
creatures, 1
we, 1
desire, 1
...
```

```
from pyspark import SparkContext
sc = SparkContext("local[*]")

# read the text file
lines = sc.textFile('shakespear.txt')
# map each lines to word occurrences
lines mapped = lines.flatMap(lambda x: [(word, 1) for word in x.split()])
# reduce each word tuples by summing the occurrences
counts = lines_mapped.reduceByKey(lambda x, y: x+y)

print(counts.collect())
```

PySpark DataFrames

- **A DataFrame is a set of records with a known schema (table), e.g. CSV file.**
- **Supports CSV, JSON, and Parquet files.**
- **DataFrame operations are built on top of RDD, i.e. all computations are distributed/parallelized.**
- **Supports a multitude of data analysis operations:**
  - **Aggregations**: mean, std, count, distinct, correlation, etc.
  - **Applying functions**: map, filter, orderBy, etc.
  - **Transformations**: select columns, add columns, remove NANs, etc.
  - **SQL Operations**: join, union, SQL queries
- **All supported DataFrame APIs:**

  https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/dataframe.html

PySpark DataFrame HelloWorld (Titanic)

```
from pyspark.sql import SparkSession

# initialize Spark
spark = SparkSession.builder.getOrCreate()
# read the CSV file
titanic = spark.read.csv('titanic.csv', header=True, inferSchema=True)
# print the schema
titanic.printSchema()
# print a few rows
titanic.show()

print('Number of rows in dataset:', titanic.count())
print('Number of rows without NaNs:', titanic.dropna().count())

# filtration & selection
survived = titanic.filter((titanic.Survived ==1) & (titanic.Age > 60))
survived names and ages = survived.select('Name', 'Age')
print(survived_names_and_ages.collect())
```

Exercises

1. How old is the youngest passenger?
2. Are there any people with the same name abroad?
3. How many unique ticket prices?
4. Is there a high correlation (> 0.5) between Age and Survival?
5. What is average age for male and female passengers?
6. What is the survival rate per age group (1-20, 20-40, 40+)?