



COMP6231 - Lab 9

PySpark RDD & MapReduce

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.

Streaming SQL ML Graph



Recap: Spark programming interface

- **Driver**
 - Runs user program
 - Defines RDDs and invokes actions
 - Tracks the lineage
- **Workers**
 - Long-lived processes
 - Store RDD partitions in RAM across operations

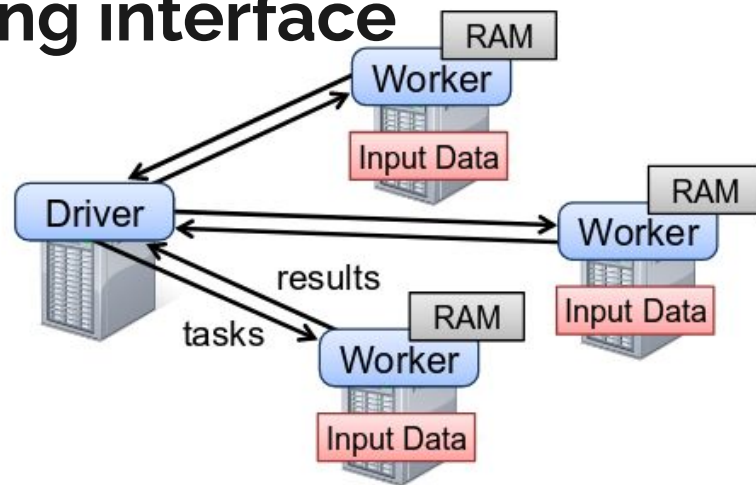


Figure 2: Spark runtime. The user's driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.



Recap: Spark Features

- **Spark keeps data in memory**
 - Reduces disk I/O
 - Spills to disk if not enough space
- **Lazy Evaluation**
 - Transformations do not compute RDDs immediately
 - The returned RDD object is a representation of the result
 - Actions trigger computations
 - This allows optimization in the execution plan



Recap: Spark Features

- **Persistence (or caching)**
 - By default RDDs are ephemeral
 - Users can persist RDDs in memory or on disk, by calling `.persist()`
- **Fault tolerance**
 - Lineage Graph: RDDs maintain information on how they were derived from other datasets.
 - When a task fails, if parent partitions are available: re-submit to another node. Otherwise, resubmit task.

Recap: RDD transformations and actions

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

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

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

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())
```



Installing PySpark

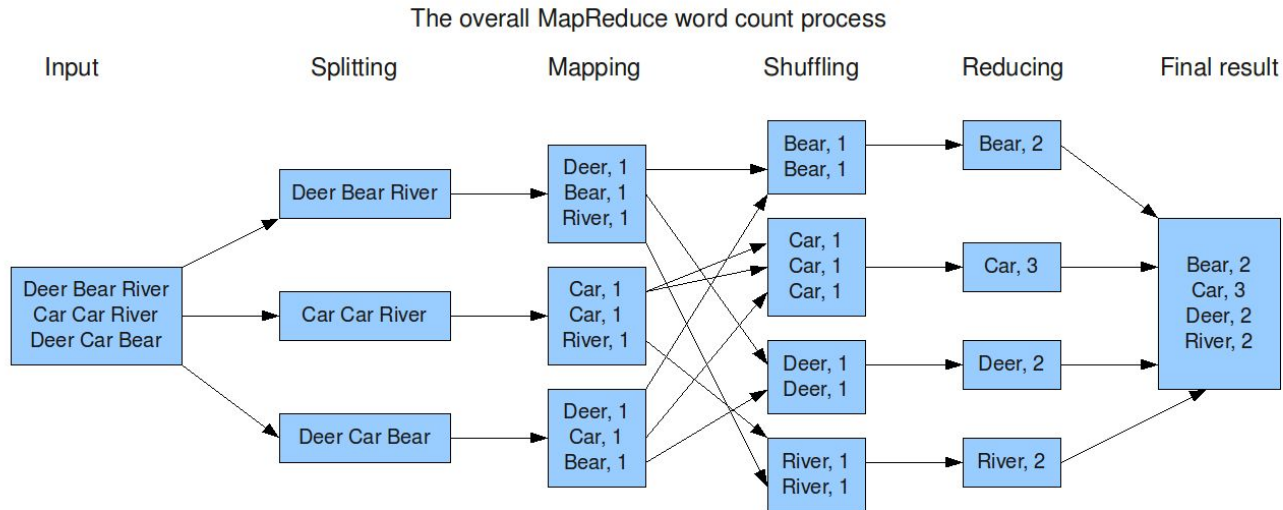
- Make sure Java (JDK) is installed.
- Install PySpark:

```
pip install pyspark
```

- For Windows, make sure environment variables are set correctly and winutils.exe is installed. For a step-by-step guide, see [here](#).

MapReduce

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





MapReduce Programming model

- Spark supports MapReduce on RDDs.
- **Input**
 - A set of *input* key-value pairs
- **Map function**
 - Provided by user
 - Takes *an* input pair
 - Produces *intermediate* key-value pairs
- Shuffle and sort
 - Done by framework
 - Aggregate intermediate values by key
- Reduce function
 - Provided by user
 - Takes *intermediate* key and list of values
 - Produces a set of *output* key-value pairs



Example: Word Count (1)

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

# read the text file
words = sc.textFile('grocery.txt')
# map each word with one occurrence
words_mapped = words.map(lambda x: (x, 1))
# reduce each word tuples by summing the occurrences
counts = words_mapped.reduceByKey(lambda x, y: x+y)

print(counts.collect())
```

Input:

```
butter
mango
pizza
croissant
stew
wheat
stew
empanada
butter
nectarine
mousse
pizza
croissant
```

Output:

```
butter, 2
mango, 1
pizza, 2
croissant, 2
stew, 2
wheat, 1
empanada, 1
nectarine, 1
mousse, 1
```



Example: Word Count (2)

```
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())
```

Input:

```
From fairest creatures we desire increase
That thereby beauty's rose might never die
But as the ripper 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
...
```

RDDs can map any function - Finding Primes

```
def isprime(n):
    if n < 2:
        return False
    if n == 2:
        return True
    if not n & 1:
        return False
    for x in range(3, int(n**0.5)+1, 2):
        if n % x == 0:
            return False
    return True

from pyspark import SparkContext
sc = SparkContext("local[*]")
nums = sc.parallelize(range(10000000))
num_primes = nums.map(lambda x: isprime(x))
primes = num_primes.filter(lambda x: x)
print(primes.count())
```



Exercises

- **Write MapReduce programs in PySpark that:**
 - Given receipt.txt containing items and their prices, calculate the total revenue of the store.
 - Given students.txt containing the courses registered by each student, calculate the total number of students per course.