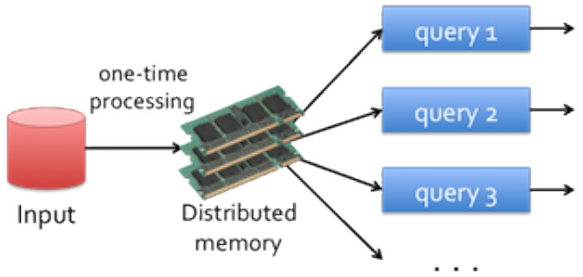
Big Data
Spark

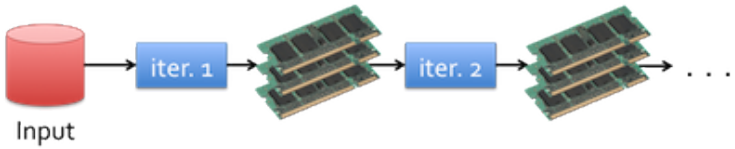
Dr. Wenceslao PALMA
wenceslao.palma@pucv.cl

Spark: Intro

- Written in Scala.
- Scalable, efficient analysis of Big Data.
  - It mantains MapReduce's linear scalability and fault tolerance.
  - it extends MapReduce with in-memory processing.
- 10-100x faster than network and disk.
- 2-5x less code.



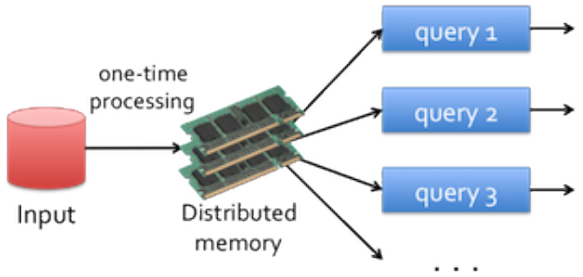
(a) Low-latency computations (queries)



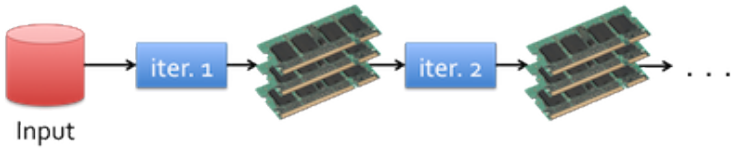
(b) Iterative computations

Spark: Intro

- Written in Scala.
- Scalable, efficient analysis of Big Data.
  - It mantains MapReduce's linear scalability and fault tolerance.
  - it extends MapReduce with in-memory processing.
- 10-100x faster than network and disk.
- 2-5x less code.



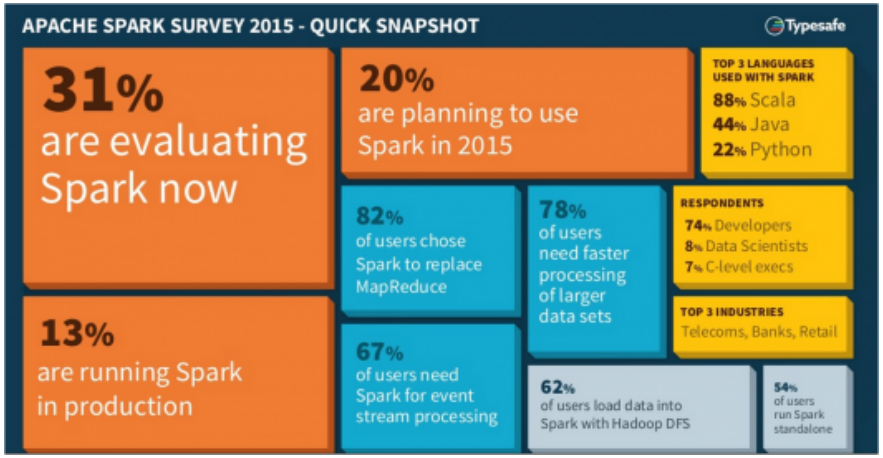
(a) Low-latency computations (queries)

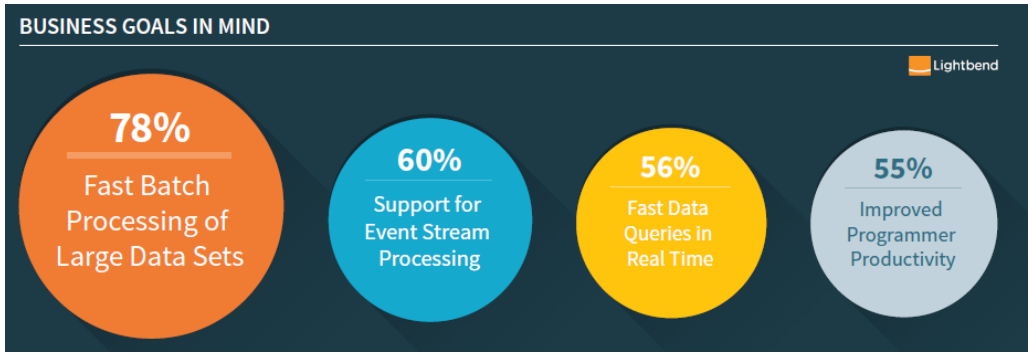


(b) Iterative computations

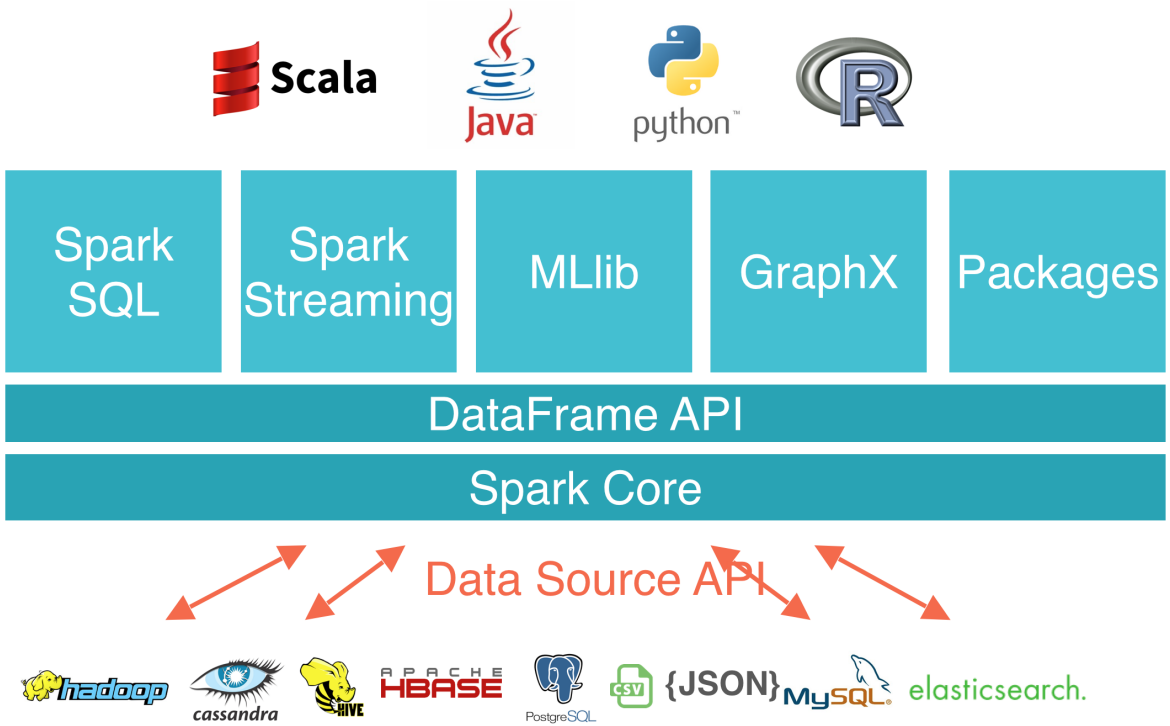
Underneath the covers, Spark uses the Java Virtual Machine.

Spark: Survey 2015





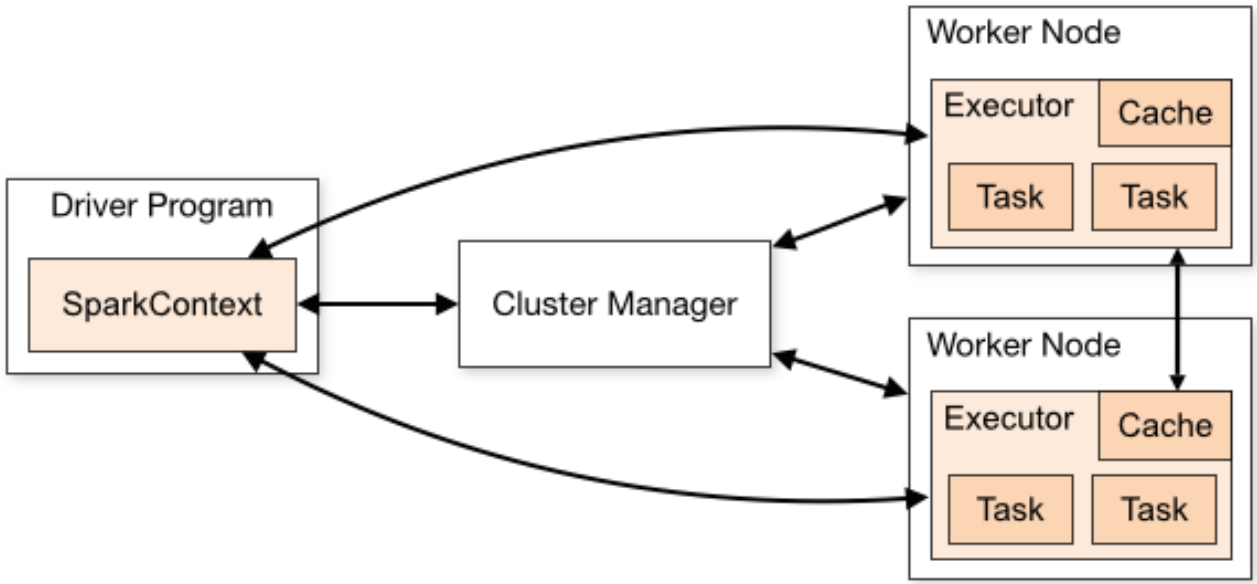
Spark: Stack



databricks

24

Spark: Cluster mode



Spark: The programming model

Writing a spark program involves typically the following steps:

- Defining a set of transformations on input data sets.
- Invoking actions that output the transformed data sets to persistent storage or return results to local memory.
- Running local computations on the results computed in a distributed fashion (it can help to decide what transformations and actions to tackle next).

Spark: DataFrames

- The key concept in Spark is the DataFrame.
- DataFrames are the primary abstraction in Spark.
- DataFrames are immutable once they are created.
- Spark efficiently recompute lost data using DataFrames.
- Spark performs two types of operations on DataFrames: transformations and actions. It doesn't execute transformations until an action occurs.

Spark: DataFrames

DataFrames are constructed by:

- parallelizing existing Python collections.
- transforming a previously created Spark DataFrame or a Pandas DataFrames.

Also, we can construc DataFrames from files in HDFS or any other storage system.

Spark: DataFrames

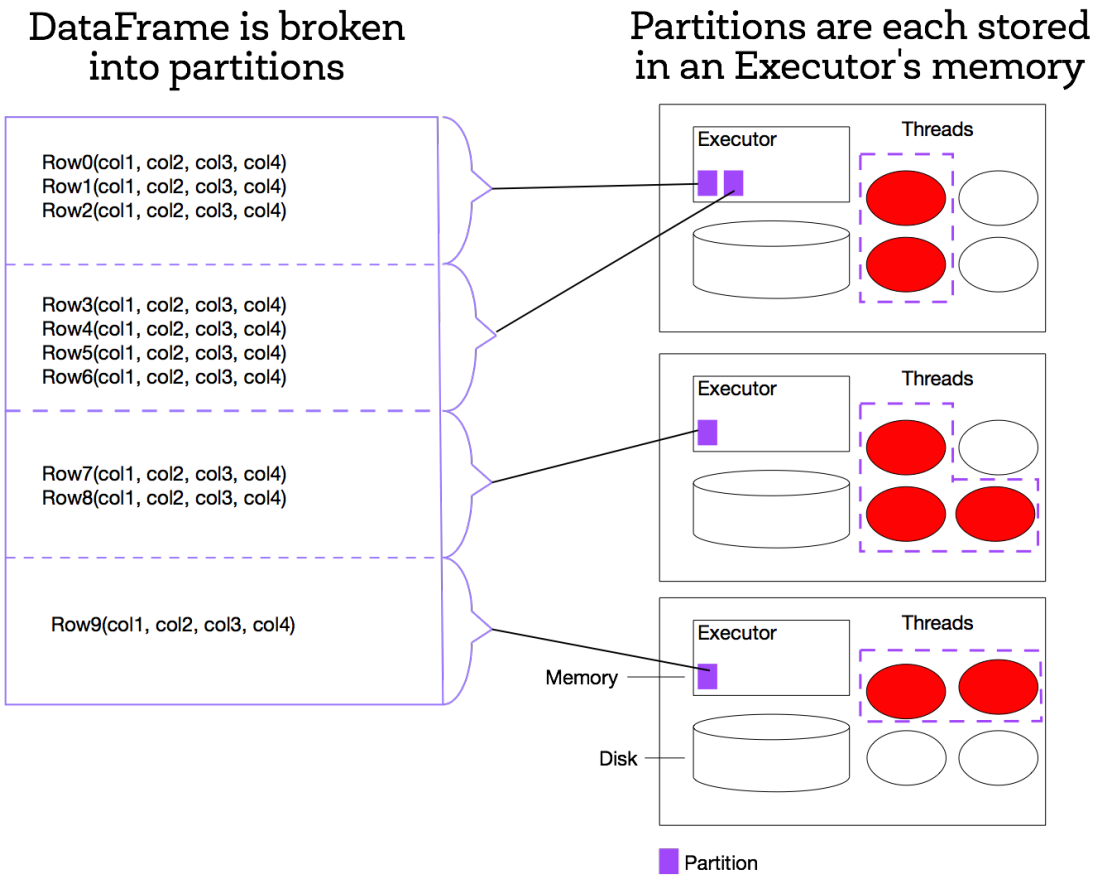
from a local file

```
>>> df = sc.textFile("file:///text.txt")
>>> df.collect()
>>> df.count()
```

from a Python list

```
>>> data = [('mike',1),('candace',2)]
>>> df = sqlContext.createDataFrame(data)
>>> df.count()
>>> df.collect()
```

Spark: How data frames are distributed



Spark: DataFrames transformations

the select method

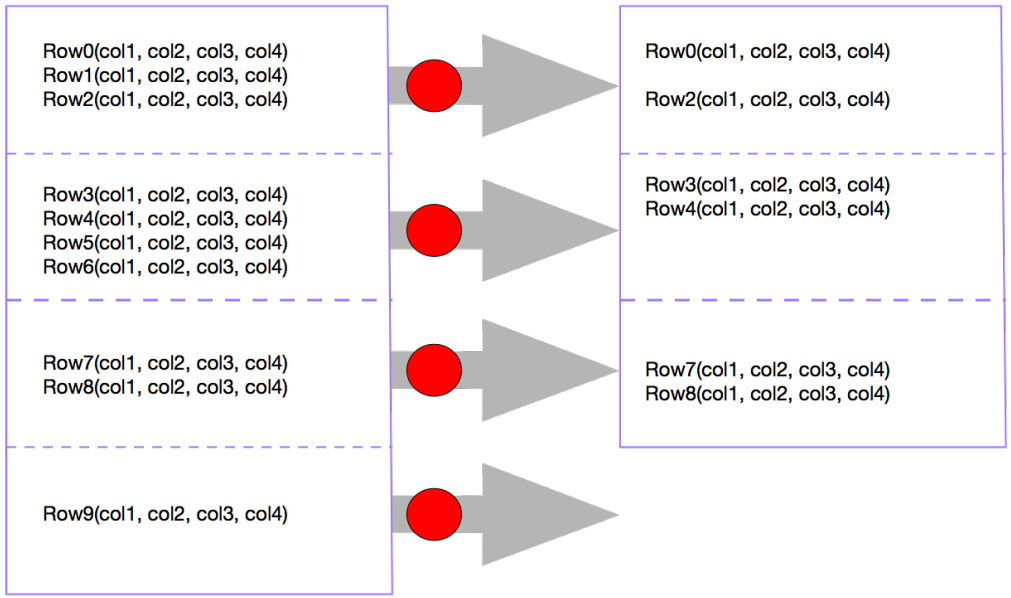
```
>>> data = [('mike','valpo'),('candace','villa alemana')]
>>> df = sqlContext.createDataFrame(data, ['name','address'])
>>> df.select('*')
>>> df.select('name')
```

Other transformations: drop, filter, distinct, orderBy, sort, map

```
>>> df.orderBy(df.name.asc()).collect()
```

Spark: The filter transformation

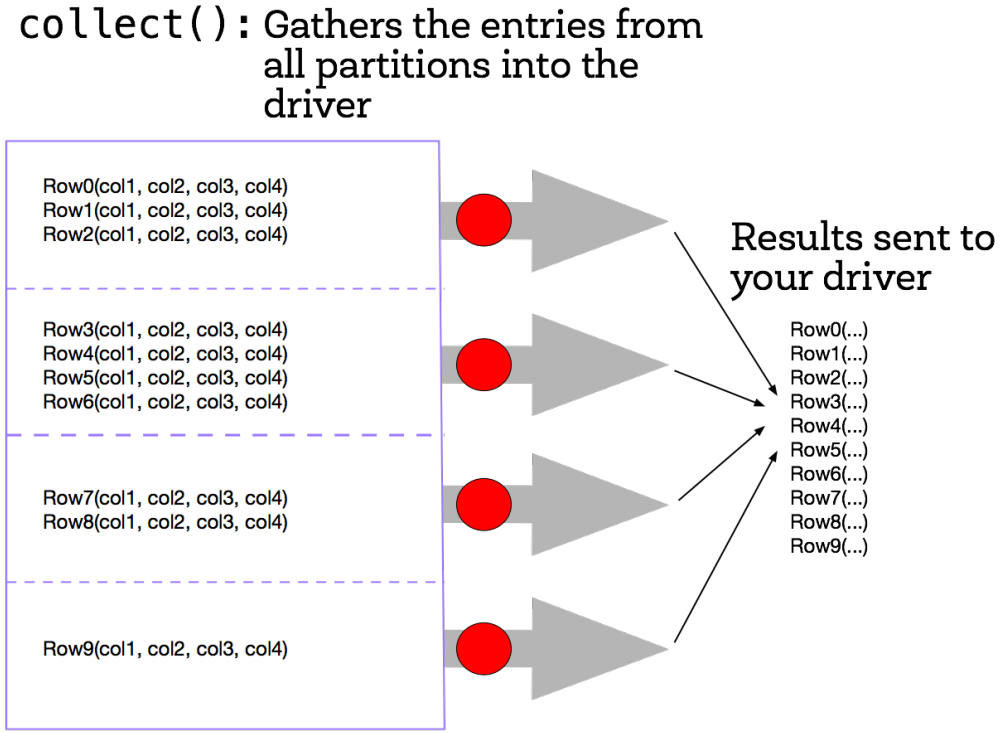
`filter()`: Each task makes a new partition with entries from the original partition that have an "age" column value less than 10.



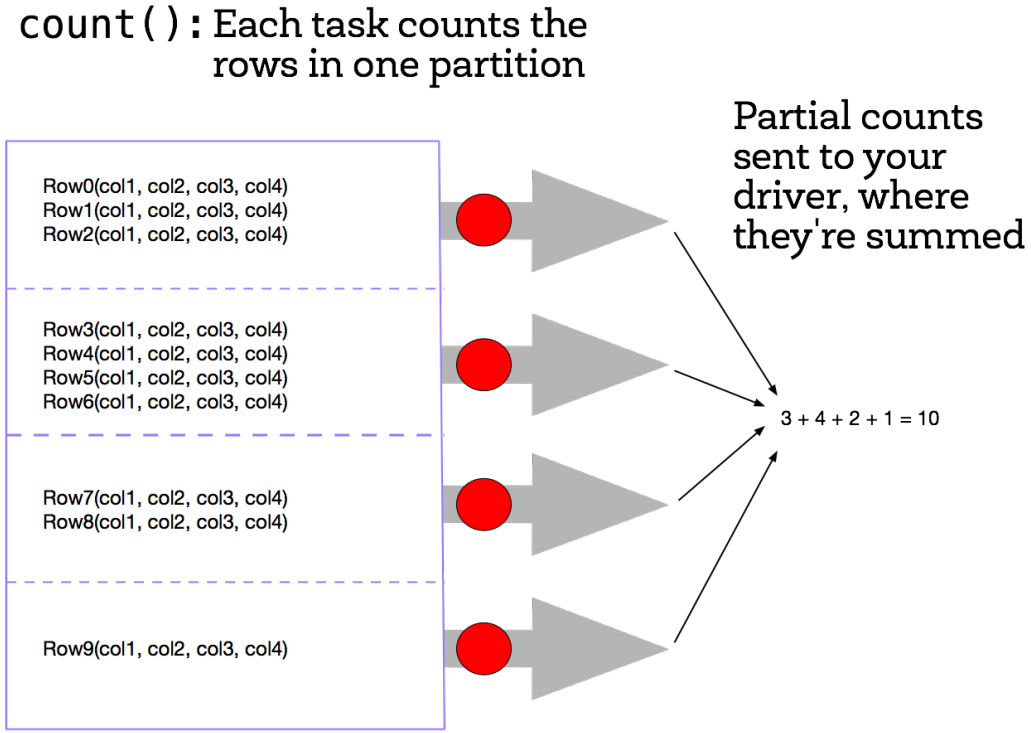
Spark: DataFrame actions

- take, collect, reduce, show, describe....
- An action is the mechanism for getting results.
- count performs a local sum on each of the workers and combines all of these sums in the driver.
- You must pay attention to collect action. All the distributed data of your DataFrame gets returned to the driver program. If it doesn't fit in the memory program you'll get an out of memory error.
- describe action works on numerical columns and returns count, mean, stddev, min and max :-)

Spark: DataFrame actions collect()



Spark: DataFrame actions count()



Spark: caching

```
>>> lines = sqlContext.read.text("hdfs://....")
>>> comments = lines.filter(isComment)
>>> print lines.count(), comments.count()
```

comments.count() recomputes lines DataFrame from disk! In order to avoid this extra recomputation step wecan tell Spark to cache the lines DataFrame.

```
>>> lines = sqlContext.read.text("hdfs://....")
>>> lines.cache()
>>> comments = lines.filter(isComment)
>>> print lines.count(), comments.count()
```

Spark: the wordcount example

```
# spark-submit wordcount-Spark.py
from pyspark import SparkConf
from pyspark import SparkContext

conf = SparkConf()
conf.setMaster('yarn-client')
conf.setAppName('spark-wordcount')
conf.set('spark.executor.instances', 3)
sc = SparkContext(conf=conf)

distFile = sc.textFile('hdfs://hadoop1/user/joe/t8.shakespeare.txt')

nonempty_lines = distFile.filter(lambda x: len(x) > 0)
# print 'Nonempty lines', nonempty_lines.count()

words = nonempty_lines.flatMap(lambda x: x.split(' '))

wordcounts = words.map(lambda x: (x, 1)) \
                .reduceByKey(lambda x, y: x+y) \
                .map(lambda x: (x[1], x[0])).sortByKey(False)

wordcounts.saveAsTextFile('hdfs://hadoop1/user/joe/wordCount-Output')
```

Spark: Spark SQL

```
# datasets http://ita.ee.lbl.gov/html/traces.html
# wget ftp://ita.ee.lbl.gov/traces/clarknet_access_log_Aug28.gz
# gzip -d clarknet_access_log_Aug28

from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

sc = SparkContext()
logs = sc.textFile('hdfs://hadoop1/user/joe/clarknet_access_log_Aug28')

ips = logs.map(lambda s: s.split(' ')[0])
count = ips.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)
ipsColumns = count.map(lambda p: Row(ip=p[0], access=int(p[1])))

sqlContext = SQLContext(sc)
schemaLog = sqlContext.createDataFrame(ipsColumns)
schemaLog.registerTempTable("accesos")

topTen = sqlContext.sql("SELECT ip,access FROM accesos order by access \
                         DESC LIMIT 100")
for result in topTen.collect():
      print "IP addr: "+str(result.ip)+" Number of Accesses: "+str(result.access)

print "-------------------------------------------------------------------
```

Spark: working with Data Frames

A step by step Spark code exercise: [1]

- Create a collection of records (using Faker).
- Create a Data Frame from that collection.
- Modify a value from all the records using select.
- Perform an action (collect()) to view results.
- Perform an action (count()) to view counts.
- Apply a transformation (filter()) and view results with collect().

[1] idea taken from DataBricks

Spark: Create a collection of records (using Faker).

```
from faker import Factory

fake=Factory.create()
fake.seed(4321)
numberOfRecords = 10000

def fake_record():
    name = fake.name().split()
    return (name[1],name[0],fake.ssn(),fake.job(),abs(2017 - fake.date_time().year) + 1,\
        fake.credit_card_provider())


def repeat(times, funcName, *args, **kwargs):
      for _ in xrange(times):
           yield funcName(*args, **kwargs)

begin = t.time()
print ('generating %d records.......') % (numberOfRecords)
records = list(repeat(numberOfRecords, fake_record))
end = t.time()

print ('Execution time = %d.3 s') % (end-begin)

print records[5000]
```

References

Advanced Analytics with Spark. Patterns from Learning from Data at Scale. Ryza S., Laserson U., Owen S., Wills J. Ed. O'reilly, 2015.