Week 14
Apache Spark: Distributed Computing Framework

Mastering Cloud Computing

Coleman Kane

(based on material by Paul Talaga)

Distributed Application Development Framework

Attempts to address some perceived shortcomings of older Hadoop and MapReduce architectures:

- Longer development cycles
- Data locality / access-times
- Interactive development capability

Resilient Distributed Datasets (RDD)

Partitioned data structure abstraction

In-memory / memory-resident store

Immutable (create/destroy read-only)

Typed

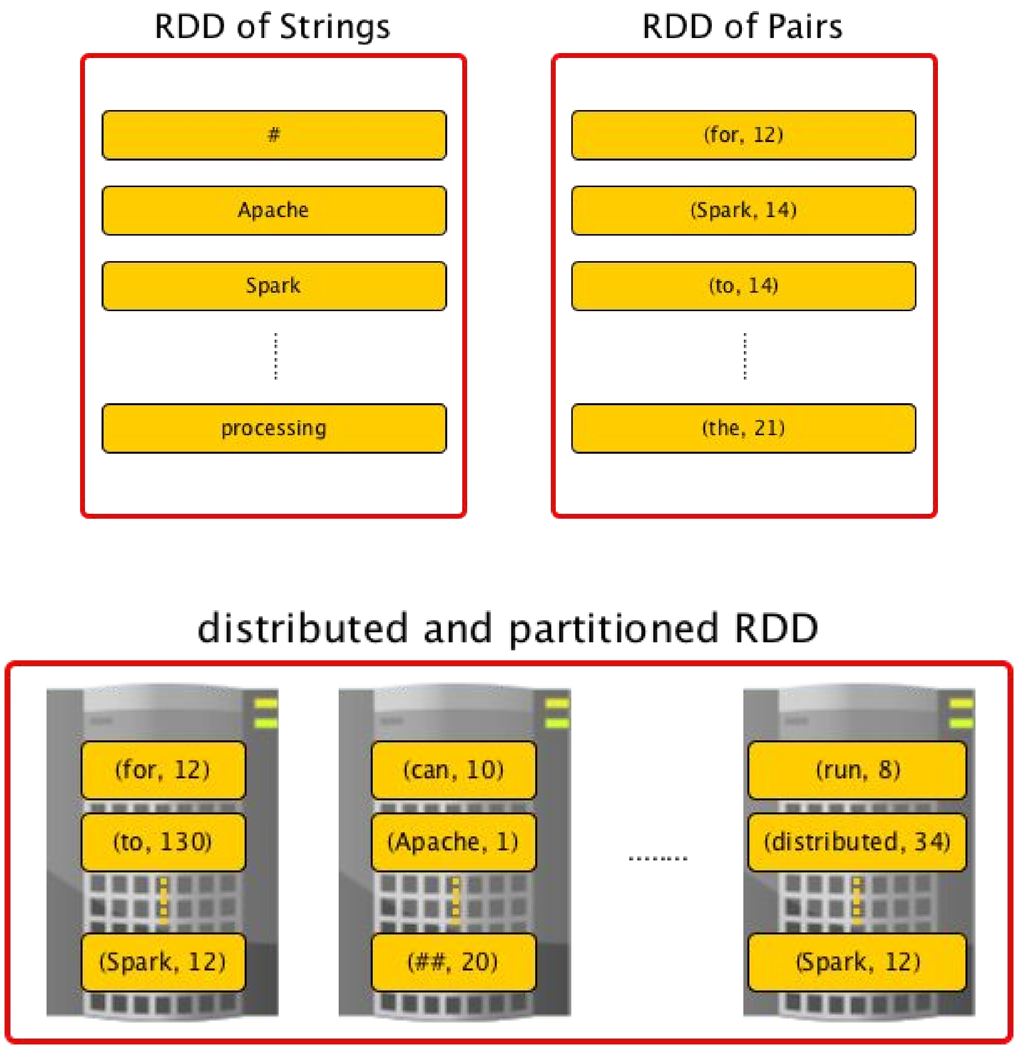
Supports lazy-evaluation

Parallelized

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

UNIVERSITY OF Cincinnati

Resilient Distributed Datasets



UNIVERSITY OF Cincinnati

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd.html

Spark “Shell” Interfaces

A largely popular improvement is the addition of “interactive shell” interfaces as a programming environment:

- Native Spark (Scala - a JVM language) - http://spark.apache.org/docs/latest/quick-start.html
- PySpark - http://spark.apache.org/docs/0.9.0/python-programming-guide.html
- SparkR - https://spark.apache.org/docs/1.6.2/sparkr.html

UNIVERSITY OF Cincinnati

Python Interface

Provides import/instantiation code primitives, so you don’t need to.

Provides mechanisms to define translation relationships between RDDs through map, reduce, and other (user-defined) functions

Lazy evaluation can be implemented here

Simple Example

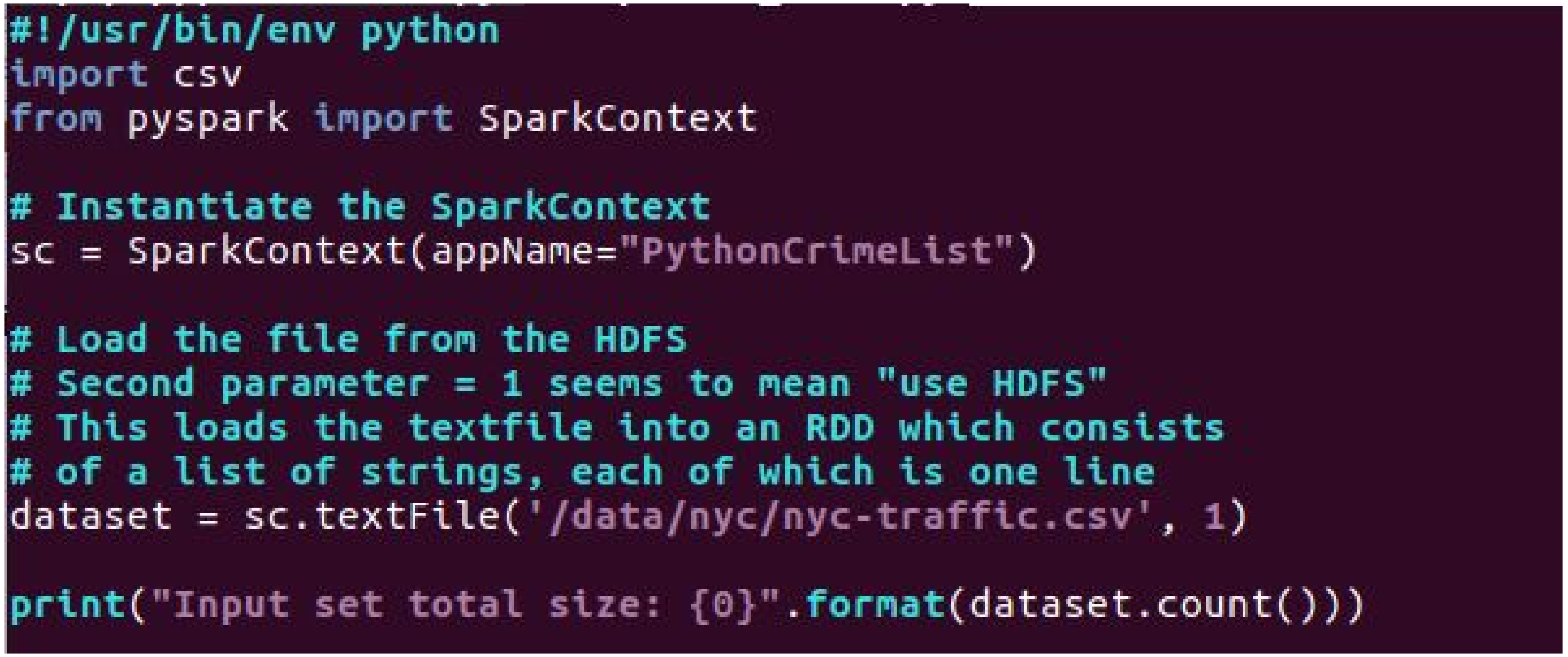
Link to example code for the following slides:

https://github.com/ckane/CS6056-PySpark-HDFS-Demos

UNIVERSITY OF Cincinnati

Simple Example

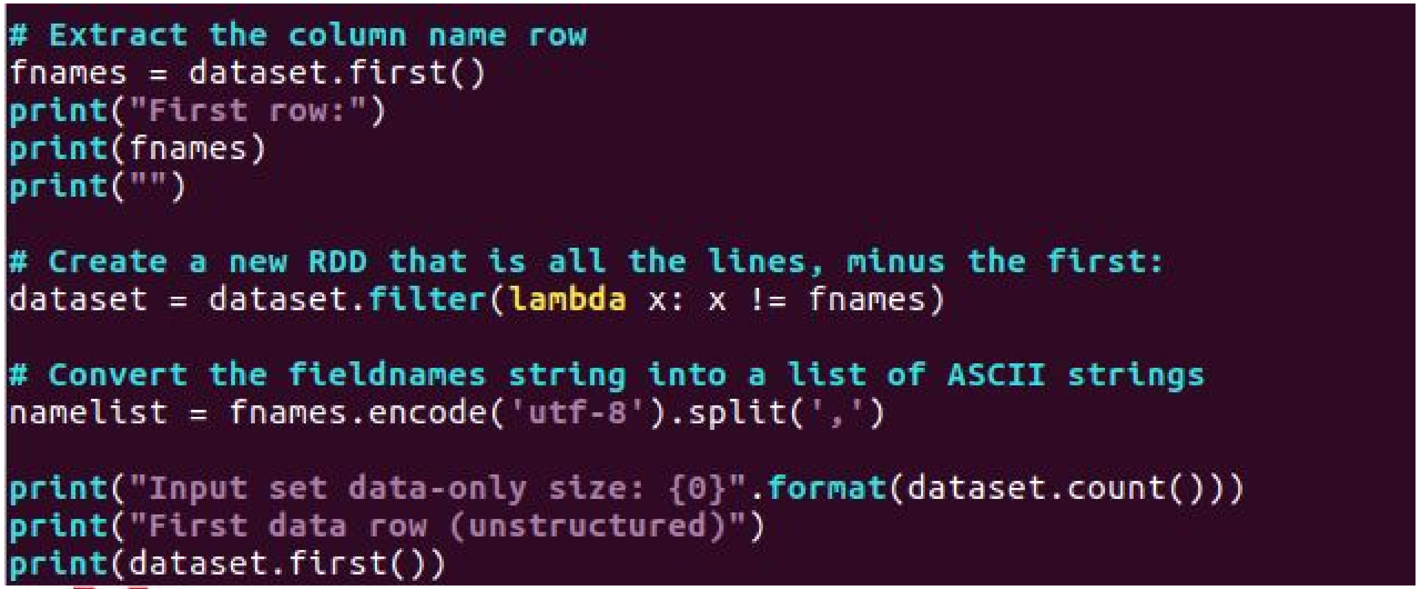
Read in data from HDFS file:



```
#!/usr/bin/env python
import csv
from pyspark import SparkContext

# Instantiate the SparkContext
sc = SparkContext(appName="PythonCrimeList")

# Load the file from the HDFS
# Second parameter = 1 seems to mean "use HDFS"
# This loads the textfile into an RDD which consists
# of a list of strings, each of which is one line
dataset = sc.textFile('/data/nyc/nyc-traffic.csv', 1)

print("Input set total size: {0}".format(dataset.count()))
```

Simple Example

First line of RDD is the “header row”, so want to extract it and translate RDD into new RDD without that row:

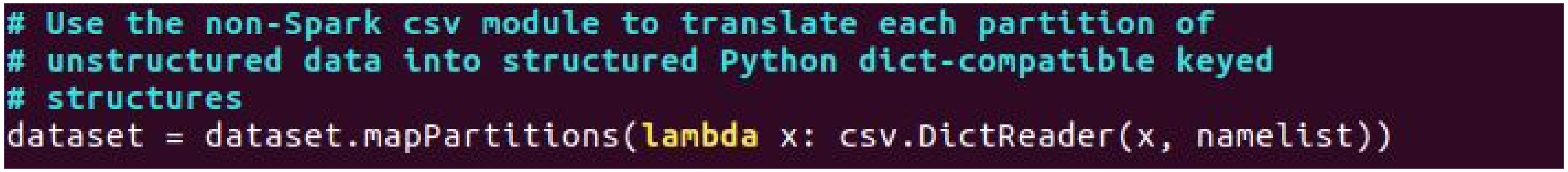


```
# Extract the column name row
fnames = dataset.first()
print("First row:")
print(fnames)
print("")

# Create a new RDD that is all the lines, minus the first:
dataset = dataset.filter(lambda x: x != fnames)

# Convert the fieldnames string into a list of ASCII strings
namelist = fnames.encode('utf-8').split(',')

print("Input set data-only size: {0}".format(dataset.count()))
print("First data row (unstructured)")
print(dataset.first())
```

UNIVERSITY OF Cincinnati

Simple Example

We want to use the “csv” Python module to convert data rows from strings into structured dict objects:

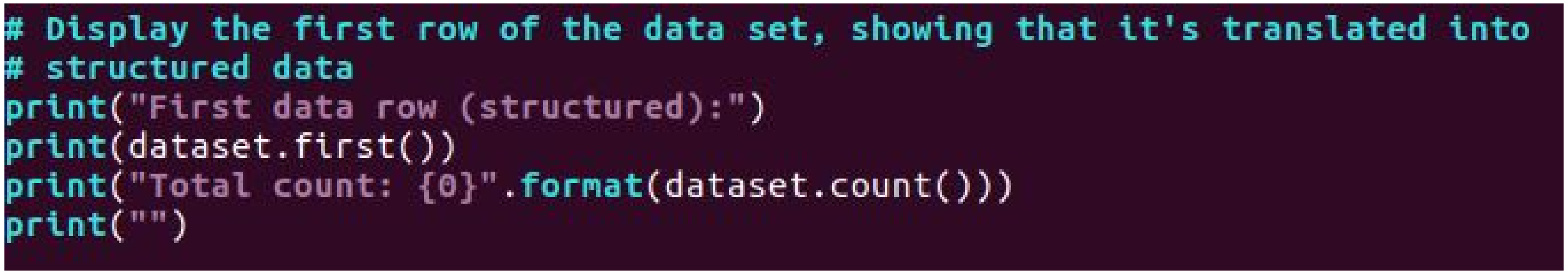


```python
# Use the non-Spark csv module to translate each partition of
# unstructured data into structured Python dict-compatible keyed
# structures
dataset = dataset.mapPartitions(lambda x: csv.DictReader(x, namelist))
```

UNIVERSITY OF Cincinnati

Simple Example

Output the first row of the translated data set to demonstrate that it’s been translated:



```
# Display the first row of the data set, showing that it's translated into
# structured data
print("First data row (structured):")
print(dataset.first())
print("Total count: {0}".format(dataset.count()))
print("")
```

Simple Example

Output from running the script:

```
Input set total size: 924098
First row:
DATE,TIME,BOROUGH,ZIP CODE,LATITUDE,LONGITUDE,LOCATION,ON STREET NAME,CROSS STREET NAME,OFF
STREET NAME,NUMBER OF PERSONS INJURED,NUMBER OF PERSONS KILLED,NUMBER OF PEDESTRIANS
INJURED,NUMBER OF PEDESTRIANS KILLED,NUMBER OF CYCLIST INJURED,NUMBER OF CYCLIST
KILLED,NUMBER OF MOTORIST INJURED,NUMBER OF MOTORIST KILLED,CONTRIBUTING FACTOR VEHICLE
1,CONTRIBUTING FACTOR VEHICLE 2,CONTRIBUTING FACTOR VEHICLE 3,CONTRIBUTING FACTOR VEHICLE
4,CONTRIBUTING FACTOR VEHICLE 5,UNIQUE KEY,VEHICLE TYPE CODE 1,VEHICLE TYPE CODE 2,VEHICLE
TYPE CODE 3,VEHICLE TYPE CODE 4,VEHICLE TYPE CODE 5

Input set data-only size: 924097
First data row (unstructured)
11/10/2016,4:11,BROOKLYN,11214,40.5857713,-73.9867258,"(40.5857713, -73.9867258)",,,146
BAY 50 STREET,0,0,0,0,0,0,0,0,Alcohol Involvement,Unspecified,,,,3557864,PASSENGER
VEHICLE,PASSENGER VEHICLE,,,
```

Simple Example

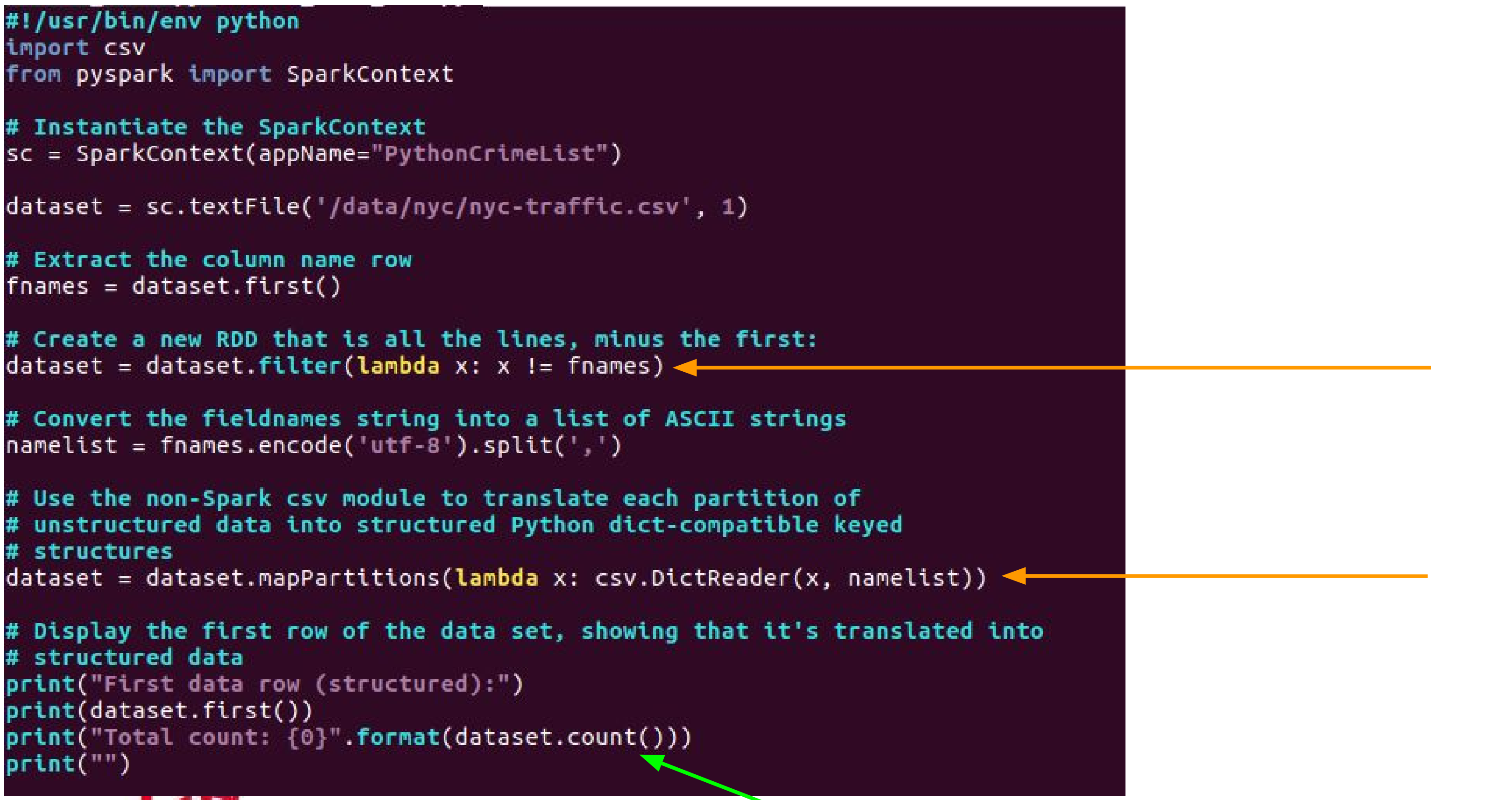
Output from running the script (cont.):

```
First data row (structured):
{'NUMBER OF CYCLIST KILLED': '0', 'CONTRIBUTING FACTOR VEHICLE 1': 'Alcohol Involvement',
'CONTRIBUTING FACTOR VEHICLE 2': 'Unspecified', 'CONTRIBUTING FACTOR VEHICLE 3': '',
'CONTRIBUTING FACTOR VEHICLE 4': '', 'CONTRIBUTING FACTOR VEHICLE 5': '', 'ZIP CODE':
'11214', 'DATE': '11/10/2016', 'OFF STREET NAME': '146       BAY 50 STREET', 'NUMBER OF
MOTORIST KILLED': '0', 'LOCATION': '(40.5857713, -73.9867258)', 'ON STREET NAME': '',
'UNIQUE KEY': '3557864', 'CROSS STREET NAME': '', 'NUMBER OF MOTORIST INJURED': '0',
'LONGITUDE': '-73.9867258', 'NUMBER OF PEDESTRIANS INJURED': '0', 'NUMBER OF PEDESTRIANS
KILLED': '0', 'VEHICLE TYPE CODE 4': '', 'VEHICLE TYPE CODE 5': '', 'VEHICLE TYPE CODE 1':
'PASSENGER VEHICLE', 'VEHICLE TYPE CODE 2': 'PASSENGER VEHICLE', 'NUMBER OF PERSONS KILLED':
'0', 'VEHICLE TYPE CODE 3': '', 'NUMBER OF CYCLIST INJURED': '0', 'TIME': '4:11',
'LATITUDE': '40.5857713', 'NUMBER OF PERSONS INJURED': '0', 'BOROUGH': 'BROOKLYN'}

Total count: 924096
```

UNIVERSITY OF Cincinnati

Example: “Lazy Evaluation”

The intermediate steps/RDDs don’t evaluate until needed in the count():



```
#!/usr/bin/env python
import csv
from pyspark import SparkContext

# Instantiate the SparkContext
sc = SparkContext(appName="PythonCrimeList")

dataset = sc.textFile('/data/nyc/nyc-traffic.csv', 1)

# Extract the column name row
fnames = dataset.first()

# Create a new RDD that is all the lines, minus the first:
dataset = dataset.filter(lambda x: x != fnames)

# Convert the fieldnames string into a list of ASCII strings
namelist = fnames.encode('utf-8').split(',')

# Use the non-Spark csv module to translate each partition of
# unstructured data into structured Python dict-compatible keyed
# structures
dataset = dataset.mapPartitions(lambda x: csv.DictReader(x, namelist))

# Display the first row of the data set, showing that it's translated into
# structured data
print("First data row (structured):")
print(dataset.first())
print("Total count: {0}".format(dataset.count()))
print("")
```

Full evaluation delayed until here