DADS 6002 / CI 7301
Big Data Analytics
Spark Lab

Spark

% nano test.txt

insert several lines of text into the file and copy it to a Hadoop file

% hadoop fs -mkdir /user/cloudera/sw

% hadoop fs -put test.txt /user/cloudera/sw/test.txt

% pyspark

>>> text = sc.textFile('/user/cloudera/sw/test.txt')

>>> text.collect()

>>> exit() (or ctrl-d)

Spark

- Use PySpark to solve Word Count problem

% pyspark

```
>>> text = sc.textFile('/user/cloudera/sw/test.txt')
>>> from operator import add
>>> def tokenize(line):
…             return line.split()
…
>>> words = text.flatMap(tokenize)
```

Spark

```
>>> wc = words.map(lambda x : (x,1) )
>>> counts = wc.reduceByKey(add)
>>> counts.saveAsTextFile('/user/cloudera/sw')
```

- Check the text file containing word counts.

```
% hadoop fs -ls /user/root/sw
% hadoop fs -cat /user/root/sw/part-00000
```

Spark

- Transformation Operations
- map(func) – returns a new RDD formed by passing each element of the source RDD through a function func.

```
>>> rdd = sc.parallelize( [1, 2, 3, 4 ] )
>>> out = rdd.map(lambda x: x * 2 )
>>> out.collect()
    [2, 4, 6, 8]
```

Spark

- filter(func) – returns a new dataset in an RDD formed by selecting those elements of the source on which func returns true.

```
>>> rdd = sc.parallelize([1,2,3,4])
>>> out = rdd.filter( lambda x: x % 2 == 0 )
>>> out.collect()
    [2, 4]
```

Spark

- distinct() – returns a new dataset in an RDD that contains the distinct elements of the source dataset

```
>>> rdd1 = sc.parallelize([1,2,3,2,4,3])
>>> out = rdd1.distinct()
>>> out.collect()
    [1,2,3,4]
```

Spark

- flatMap(func) – similar to map, but each input item can be mapped to 0 or more output items (so func should return a sequence of items)

```
>>> rdd = sc.parallelize([1,2,3,4])
>>> out = rdd.map(lambda x : [x, x+5] )
>>> out.collect(4)
    [[1,6],[2,7],[3,8],[4,9]]
>>> rdd = sc.parallelize([1,2,3,4])
>>> out = rdd.flatMap(lambda x : [x, x+5] )
>>> out.collect()
    [1,6,2,7,3,8,4,9]
```

Spark

- Action Operations
- reduce(func) – aggregate dataset's elements using function func. Func takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel (in a cluster).

```
>>> rdd = sc.parallelize([1,2,3,4])
>>> rdd.reduce(lambda a,b : a * b )
    24
```

Spark

- take(n) – returns an array with the first n elements of the RDD.

```
>>> rdd = sc.parallelize([1,2,3,4])
>>> rdd.take(2)
    [1,2]
>>> rdd.take(5)
    [1,2,3,4]
```

Spark

- collect() – returns all of the elements of the RDD as an array.

```
>>> rdd = sc.parallelize([1,2,3,4])
>>> rdd.collect()
    [1,2,3,4]
```

Spark

- takeOrdered(n, key=func) – returns an array of first n elements of the RDD in their natural order of key specified by the func.

```
>>> rdd = sc.parallelize([1,2,3,4])
>>> rdd.takeOrdered(3, lambda x : -x )
    [4, 3, 2]
```

Spark

- reduceByKey(func) – returns a new distributed dataset of (K,V) pairs in an RDD, where the values for each key are aggregated using the given reduce function func ((V,V) -> V)

```
>>> rdd = sc.parallelize([(1,2), (3,4), (3, 6), (1,3), (3,8)])
>>> out = rdd.reduceByKey(lambda a,b : a + b)
>>> out.collect()
    [(1,5), (3,18)]
```

Spark

- sortByKey() – returns a new RDD of (K,V) pairs sorted by key K in ascending order.

```
>>> rdd = sc.parallelize([(1,’c’), (3,’d’), (3, ‘a’), (1,’b’), (3,’e’)])
>>> out = rdd.sortByKey()
>>> out.collect()
    [(1,’c’), (1,’b’), (3,’d’), (3,’a’), (3, ‘e’)]
```

Spark

- groupByKey() – returns a new RDD of (K, iterable <V>) pairs.

```
>>> rdd = sc.parallelize([(1,’c’), (3,’d’), (3, ‘a’), (1,’b’), (3,’e’)])
>>> out = rdd.groupByKey()
>>> out.map(lambda x : (x[0], list(x[1]))).collect()
    [(1,[’c’,’b’]), (3,[’d’,’a’,’e’])]
```

Spark

- count() – returns the number of data items in the RDD.

```
>>> rdd = sc.parallelize([(1,’c’), (3,’d’), (3, ‘a’), (1,’b’), (3,’e’)])
>>> rdd.count()
    5
```

Spark

- foreach(func) – apply func to each item of the given RDD.

```
>>> def f(x):
…             print x
…
>>> rdd = sc.parallelize([1,2,3,4])
>>> rdd.foreach(f)
    1
    2
    3
    4
```

Spark

- Broadcast variables
- In driver program

```
>>> b = sc.broadcast([1,2,3,4])
```

- In closure

```
>>> def f(x):
…             return(b.value[x])
…
>>> rdd = sc.parallelize([0,3])
>>> out = rdd.map(f)
>>> out.collect()
     [1, 4]
```

Spark

- Accumulator variables

In a driver program, create a accumulator variable c with initial value of zero.

```
>>> rdd = sc.parallelize([1,2,3,4])
>>> c = sc.accumulator(0)
```

In a closure, access the global variable c

```
>>> def f(x):
…       global c
…       c += x
…
>>> rdd.foreach(f)
>>> c.value
        10
```

Spark

- Download the Python program “wordcount.py” from MS Teams into the shared directory, then copy it to the working directory.

```
#!/usr/bin/env python

from pyspark import SparkContext

sc = SparkContext(appName='WordCount')
input_file = sc.textFile('/user/cloudera/sw/test.txt')
tokens = input_file.flatMap(lambda line: line.split())
words = tokens.map(lambda word: (word, 1))
wc = words.reduceByKey(lambda a, b: a + b )
wc.saveAsTextFile('/user/cloudera/sw/output')
sc.stop()
```

Spark

- From the working directory, execute the python program on Spark as follows:

```
# spark-submit --master local[*] wordcount.py
# hadoop fs -ls /user/cloudera/sw/output
# hadoop fs -cat /user/cloudera/sw/output/part-00000
```