CSE 547 : Spark Tutorial

Topics

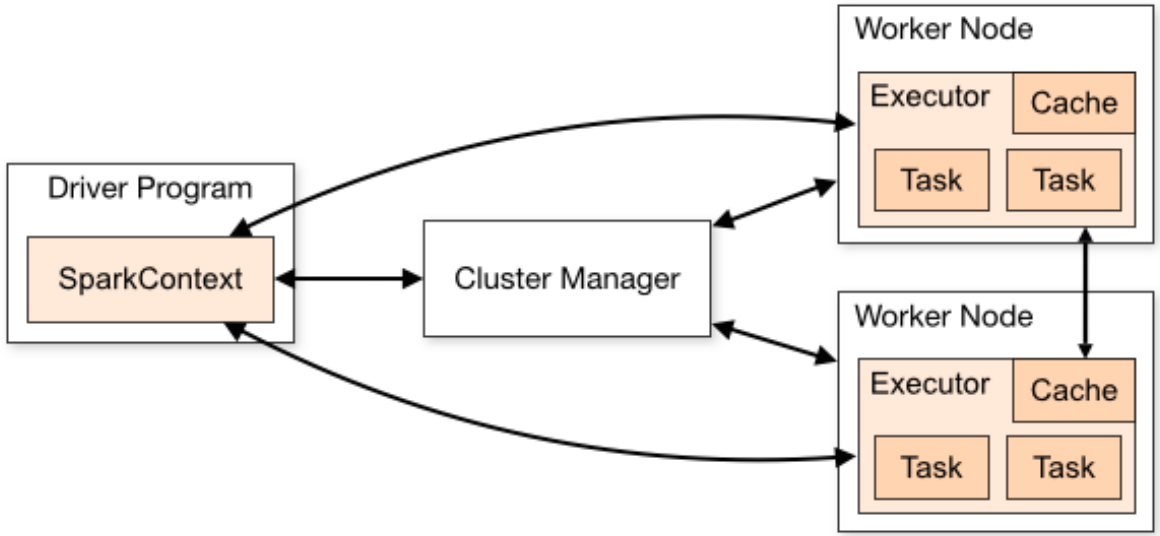
- Overview
- Useful Spark Actions and Operations
- Help session

Setup

- Follow instructions in HW 0
- Piazza
- Office Hour

Deployment Options

- Local
- Stand-alone clusters
- Managed clusters
  - e.g. YARN



Resilient Distributed Dataset (RDD)

- Contain various data type
  - Int, String, Pair …
- Immutable
- Lazily computed
- Cached
- Pair RDD: RDD that only contains tuples of 2 elements
  - (key, value)

RDD Actions

- Not produce new RDDs
  - Debug, output
- take(n)
- collect()
- count()
- saveAsTextFile(path)
- foreach(f)

RDD Operation: map

- RDD.map(f)
- Return a new RDD by applying a function to each element of this RDD

```
>>> rdd = sc.parallelize(["b", "a", "c"])
>>> sorted(rdd.map(lambda x: (x, 1)).collect())
[('a', 1), ('b', 1), ('c', 1)]
```

RDD Operation: flatMap

- RDD.flatMap(f)
- Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

```
>>> rdd = sc.parallelize([2, 3, 4])
>>> sorted(rdd.flatMap(lambda x: range(1, x)).collect())
[1, 1, 1, 2, 2, 3]
```

```
[2, 3, 4] -> [[1], [1, 2], [1, 2, 3]] -> [1, 1, 1, 2, 2, 3]
```

RDD Operation: mapValues

- PairRDD.mapValues(f)
- Pass each value in the key-value pair RDD through a map function without changing the keys; this also retains the original RDD's partitioning.

```
>>> x = sc.parallelize([
         ("a", ["apple", "banana", "lemon"]),
         ("b", ["grapes"])
        ])
>>> def f(x): return len(x)
>>> x.mapValues(f).collect()
[('a', 3), ('b', 1)]
```

RDD Operation: flatMapValue

- PairRDD.flatMapValue
- Pass each value in the key-value pair RDD through a flatMap function.

```
>>> x = sc.parallelize([("a", ["x", "y", "z"]), ("b", ["p", "r"])])
>>> def f(x): return x
>>> x.flatMapValues(f).collect()
[('a', 'x'), ('a', 'y'), ('a', 'z'), ('b', 'p'), ('b', 'r')]
```

RDD Operation: filter

- RDD.filter(f)
- Return a new RDD that only contains the element the function f returns true on.

```
>>> rdd = sc.parallelize([1, 2, 3, 4, 5])
>>> rdd.filter(lambda x: x % 2 == 0).collect()
[2, 4]
```

RDD Operation: groupByKey

- PairRDD.groupByKey()
- Group values with the same key together.
- Wide operation

```
>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 2)])
>>> result = rdd.groupByKey().collect()
>>> print(result)
[('a', <pyspark.resultiterable.ResultIterable object at 0x10ffad8d0>), ('b', <pyspark.resultiterable.ResultIterable object at 0x10ffad9b0>)]
>>> print([(pair[0], [value for value in pair[1]]) for pair in result])
[('a', [1, 2]), ('b', [1])]
```

RDD Operation: reduceByKey

- PairRDD.reduceByKey(f)
- Merge the values for each key using an associative and commutative reduce function.

```
>>> from operator import add
>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1), (”a”, 2)])
>>> sorted(rdd.reduceByKey(add).collect())
[('a', 4), ('b', 1)]
```

RDD Operation : sortBy

- RDD.sortBy(keyfunc)
- Sorts this RDD by the given keyfunc

```
>>> tmp = [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]
>>> sc.parallelize(tmp).sortBy(lambda x: x[0]).collect()
[('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]
>>> sc.parallelize(tmp).sortBy(lambda x: x[1]).collect()
[('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]
```

RDD Operation: subtract

- RDD.subtract(RDD)
- Return a new RDD containing each value in the original RDD that not in the other RDD.

```
>>> x = sc.parallelize([("a", 1), ("b", 4), ("b", 5), ("a", 3)])
>>> y = sc.parallelize([("a", 3), ("c", None)])
>>> sorted(x.subtract(y).collect())
[('a', 1), ('b', 4), ('b', 5)]
```

RDD Operatoin: join

- PairRDD.join(PairRDD)
- For each pair of pairs in the original and the other RDDs that have the same key, join their values together in the result RDD.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("a", 3)])
>>> sorted(x.join(y).collect())
[('a', (1, 2)), ('a', (1, 3))]
```

Example: Word Count

```
conf = SparkConf()
sc = SparkContext(conf=conf)
lines = sc.textFile(sys.argv[1])
words = lines.flatMap(lambda l: re.split(r'[^\w]+', l))
pairs = words.map(lambda w: (w, 1))
counts = pairs.reduceByKey(lambda n1, n2: n1 + n2)
counts.saveAsTextFile(sys.argv[2])
sc.stop()
```

Example: Word Count

```
words = lines.flatMap(lambda l:
re.split(r'[^\w]+', l))

pairs = words.map(
  lambda w: (w, 1)
)
counts = pairs.reduceByKey(
  lambda n1, n2: n1 + n2
)
counts.saveAsTextFile(sys.argv[2])
sc.stop()
```

```
lines = [’I love data mining.’,
‘data mining is great.’]

words = [‘I’, ‘love’, ‘data’,
‘mining’, ‘data’, ‘mining’,
‘is’, ‘great’]

pairs = [(‘I’, 1), (‘love’, 1),
(‘data’, 1), (‘mining’, 1),
(‘data’, 1), (‘mining’, 1),
(‘is’, 1), (‘great’, 1)]

counts = [(‘I’, 1), (‘love’, 1),
(‘data’, 2), (‘mining’, 2),
(‘is’, 1), (‘great’, 1)]
```

Help Session

References

- PySpark 2.4.0 documentation https://spark.apache.org/docs/2.4.0/api/python/pyspark.html