Yandex

Broadcast & accumulator variables

Sample data

- Financial data from http://finance.yahoo.com for NASDAQ
- Stored in the CSV format in the file 'nasdaq.csv'

```
2017-01-03,5425.620117,5452.569824,5397.990234,5429.080078,5429.080078,1886200000
2017-01-04,5440.910156,5482.350098,5440.240234,5477.000000,5477.000000,1883360000
2017-01-05,5474.390137,5495.850098,5464.359863,5487.939941,5487.939941,1792610000
2017-01-06,5499.080078,5536.520020,5482.810059,5521.060059,5521.060059,1710770000
2017-01-09,5527.580078,5541.080078,5517.140137,5531.819824,5531.819824,1885500000
2017-01-10,5536.540039,5564.250000,5528.109863,5551.819824,5551.819824,1796500000
2017-01-11,5550.720215,5564.080078,5524.029785,5563.649902,5563.649902,1954720000
2017-01-12,5542.560059,5550.669922,5496.819824,5547.490234,5547.490234,1801750000
2017-01-13,5557.569824,5584.259766,5557.200195,5574.120117,5574.120117,1605110000
2017-01-17,5555.160156,5557.049805,5527.220215,5538.729980,5538.729980,1757030000
...
```

$ pyspark

```
>>> from collections import namedtuple
>>> Record = namedtuple("Record", ["date", "open", "high", "low", "close",
                        "adj_close", "volume"])
>>> def parse_record(s):
...   fields = s.split(",")
...   return Record(fields[0], *map(float, fields[1:6]), int(fields[6]))
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> import time
>>> import random
>>> def super_regressor(v):
...   time.sleep(random.random() / 1000.0)
...   return 0.5 * ((v - 1910949928.057554) / 284610509.115) ** 2.0
...
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> import time
>>> import random
>>> def super_regressor(v):
...   time.sleep(random.random() / 1000.0)
...   return 0.5 * ((v - 1910949928.057554) / 284610509.115) ** 2.0
...
>>> time_spent = sc.accumulator(0.0)
>>> time_spent
Accumulator<id=0, value=0.0>
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> import time
>>> import random
>>> def super_regressor(v):
...     time.sleep(random.random() / 1000.0)
...     return 0.5 * ((v - 1910949928.057554) / 284610509.115) ** 2.0
...
>>> time_spent = sc.accumulator(0.0)
>>> time_spent
Accumulator<id=0, value=0.0>
>>> def timed_super_regressor(v):
...     before = time.time()
...     result = super_regressor(v)
...     after = time.time()
...     time_spent.add(after - before)
...     return result
...
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> import time
>>> import random
>>> def super_regressor(v):
...   time.sleep(random.random() / 1000.0)
...   return 0.5 * ((v - 1910949928.057554) / 284610509.115) ** 2.0
...
>>> time_spent = sc.accumulator(0.0)
>>> time_spent
Accumulator<id=0, value=0.0>
>>> def timed_super_regressor(v):
...   before = time.time()
...   result = super_regressor(v)
...   after = time.time()
...   time_spent.add(after - before)
...   return result
...
>>> estimates = parsed_data.map(lambda r: timed_super_regressor(r.volume)).collect()
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> import time
>>> import random
>>> def super_regressor(v):
...   time.sleep(random.random() / 1000.0)
...   return 0.5 * ((v - 1910949928.057554) / 284610509.115) ** 2.0
...
>>> time_spent = sc.accumulator(0.0)
>>> time_spent
Accumulator<id=0, value=0.0>
>>> def timed_super_regressor(v):
...   before = time.time()
...   result = super_regressor(v)
...   after = time.time()
...   time_spent.add(after - before)
...   return result
...
>>> estimates = parsed_data.map(lambda r: timed_super_regressor(r.volume)).collect()
>>> time_spent.value
0.09158062934875488
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> time_spent = sc.accumulator(0.0)
>>> def timed_super_regressor(v):
...   # measure the regressor time and update the 'time_spent' accumulator
...
>>> estimates = parsed_data.map(lambda r: timed_super_regressor(r.volume)).collect()
>>> time_spent.value
0.09158062934875488
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> time_spent = sc.accumulator(0.0)
>>> def timed_super_regressor(v):
...   # measure the regressor time and update the 'time_spent' accumulator
...
>>> estimates = parsed_data.map(lambda r: timed_super_regressor(r.volume)).collect()
>>> time_spent.value
0.09158062934875488
>>> estimates = parsed_data.map(lambda r: timed_super_regressor(r.volume)).collect()
>>> time_spent.value
0.18324017524719238
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> from pyspark import AccumulatorParam
>>> class MaxAccumulatorParam(AccumulatorParam):
...
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> from pyspark import AccumulatorParam
>>> class MaxAccumulatorParam(AccumulatorParam):
...   def zero(self, initial_value):
...     return initial_value
...
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> from pyspark import AccumulatorParam
>>> class MaxAccumulatorParam(AccumulatorParam):
...   def zero(self, initial_value):
...     return initial_value
...   def addInPlace(self, accumulator, delta):
...     return max(accumulator, delta)
...
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> from pyspark import AccumulatorParam
>>> class MaxAccumulatorParam(AccumulatorParam):
...   def zero(self, initial_value):
...     return initial_value
...   def addInPlace(self, accumulator, delta):
...     return max(accumulator, delta)
...
>>> time_persist = sc.accumulator(0.0, MaxAccumulatorParam())
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> from pyspark import AccumulatorParam
>>> class MaxAccumulatorParam(AccumulatorParam):
...   def zero(self, initial_value):
...     return initial_value
...   def addInPlace(self, accumulator, delta):
...     return max(accumulator, delta)
...
>>> time_persist = sc.accumulator(0.0, MaxAccumulatorParam())
>>> def persist_to_external_storage(iterable):
...   for record in iterable:
...     before = time.time
...     time.sleep(random.random() / 1000.0)  # --party-- persist hard
...     after = time.time()
...     time_persist.add(after - before)
...
>>> parsed_data.foreachPartition(persist_to_external_storage)
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> from pyspark import AccumulatorParam
>>> class MaxAccumulatorParam(AccumulatorParam):
...   def zero(self, initial_value):
...     return initial_value
...   def addInPlace(self, accumulator, delta):
...     return max(accumulator, delta)
...
>>> time_persist = sc.accumulator(0.0, MaxAccumulatorParam())
>>> def persist_to_external_storage(iterable):
...   for record in iterable:
...     before = time.time
...     time.sleep(random.random() / 1000.0)  # --party-- persist hard
...     after = time.time()
...     time_persist.add(after - before)
...
>>> parsed_data.foreachPartition(persist_to_external_storage)
>>> time_persist.value
0.0012042522430419922
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>>
>>> def super_regressor(v):
...     time.sleep(random.random() / 1000.0
...     return 0.5 * ((v - 1910949928.057554) / 284610509.115) ** 2.0
...
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> params = sc.broadcast({"mu": 1910949928.057554, "sigma": 284610509.115})
>>> def super_regressor(v):
...   time.sleep(random.random() / 1000.0)
...   return 0.5 * ((v - 1910949928.057554) / 284610509.115) ** 2.0
...
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> params = sc.broadcast({"mu": 1910949928.057554, "sigma": 284610509.115})
>>> def super_regressor(v):
...   time.sleep(random.random() / 1000.0)
...   return 0.5 * ((v - params.value["mu"]) / params.value["sigma"]) ** 2.0
...
>>>
```

$ pyspark

```
...
>>> parsed_data = sc.textFile("nasdaq.csv").map(parse_record).cache()
>>> params = sc.broadcast({"mu": 1910949928.057554, "sigma": 284610509.115})
>>> def super_regressor(v):
...   time.sleep(random.random() / 1000.0)
...   return 0.5 * ((v - params.value["mu"]) / params.value["sigma"]) ** 2.0
...
>>> parsed_data.map(lambda x: super_regressor(x.volume)).top(1)
[10.00570754168115]
>>>
```

Summary

- You have learned how to:
  - create and use accumulator variables
  - use a custom associative and commutative operation in an accumulator
  - create and use broadcast variables
  - use the 'foreachPartition' action to invoke arbitrary code on a data set

BigDATAteam