Introduction to Parallel and Distributed Processing

Introduction to Apache Spark

Jaroslaw 'Jaric' Zola

http://www.jzola.org/

Suggested Reading

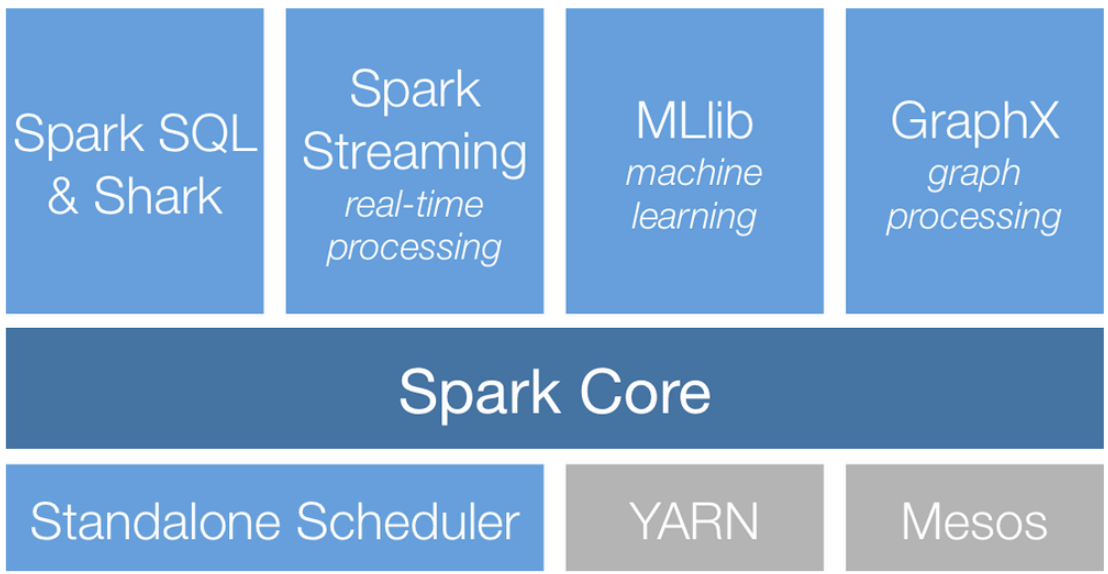
- Apache Spark http://spark.apache.org/
- “Learning Spark” Karu, H. et al. O’Reilly 2015

Spark Overview

- Distributed memory computing platform extending Map/Reduce model
- Fairly unified stack working with different cluster management systems and storage systems
- Support for iterative processing and “shared variables”

Spark Stack

- Core implementation in Scala, bindings to Java and Python
- Support for SQL, BSP model on graphs and real-time apps built on top



General Idea

- Use a master-worker model, with “driver program” and executors
- Driver creates the execution context and keeps track of the data
- Executors do the actual work and store the data

Spark Context

- Abstraction representing compute cluster
- Can be used to “configure” cluster (e.g. set environment)

```
1  #!/usr/bin/python
2  # ex01.py
3
4  from pyspark import SparkContext
5  import time
6
7  if __name__ == "__main__":
8      sc = SparkContext(appName="Spark App")
9      time.sleep(120)
10     sc.stop()
```

```
1  spark-submit --num-executors=8 --executor-cores 4 ex01.py
```

Resilent Distributed Datasets

- Read-only, distributed collection of objects
- Programmer has control over partitioning and persistance
- RDD can be created either by transforming other RDD or from stable storage
- RDDs use lazy evaluation
- Fault tolerance is achieved via “lineage”

Creating RDD

```
1  #!/usr/bin/python
2  # ex02.py
3
4  from pyspark import SparkConf, SparkContext
5  from random import random
6
7  if __name__ == "__main__":
8      sc = SparkContext()
9      # not parallel
10     rn = sc.parallelize([random() for i in xrange(100000)])
11     sc.stop()
```

Creating RDD

```
1  #!/usr/bin/python
2  # ex03.py
3
4  from pyspark import SparkConf, SparkContext
5  import sys
6
7  if __name__ == "__main__":
8      conf = SparkConf().setAppName("RDDcreate")
9      sc = SparkContext(conf = conf)
10     lines = sc.textFile(sys.argv[1])
11     sc.stop()
```

Creating RDD

```
1  #!/usr/bin/python
2  # ex04.py
3
4  from pyspark import SparkConf, SparkContext
5  import json
6  import sys
7
8  if __name__ == "__main__":
9      sc = SparkContext()
10     rdd = sc.textFile(sys.argv[1]).map(lambda x: json.loads(x))
11     sc.stop()
```

Transformations

- Create new RDD by applying some processing to already existing RDD
- Examples: `flatMap`, `reduceByKey`, `filter`, `repartition`

```
1  #!/usr/bin/python
2  # ex05.py
3
4  from pyspark import SparkConf, SparkContext
5  import sys
6
7  def vertex(line):
8      l = line.split(" ")
9      v = [ int(x) for x in l ]
10     return [(v[0], 1), (v[1], 1)]
11
12 if __name__ == "__main__":
13     conf = SparkConf().setAppName("RDDcreate")
14     sc = SparkContext(conf = conf)
15     lines = sc.textFile(sys.argv[1])
16     V = lines.flatMap(vertex) \
17              .reduceByKey(lambda a, b: a + b)
18     sc.stop()
```

Actions

- Apply operation on RDD and return result to the driver, or write to stable storage
- Force evaluation of transformations
- Examples: `count`, `reduce`, `take`, `saveAsTextFile`

Actions

```
1  #!/usr/bin/python
2  # ex06.py
3
4  from pyspark import SparkConf, SparkContext
5  import sys
6
7  def vertex(line):
8      l = line.split(" ")
9      v = [ int(x) for x in l ]
10     return [(v[0], 1), (v[1], 1)]
11
12 if __name__ == "__main__":
13     conf = SparkConf().setAppName("RDDcreate")
14     sc = SparkContext(conf = conf)
15     lines = sc.textFile(sys.argv[1])
16     V = lines.flatMap(vertex) \
17               .reduceByKey(lambda a, b: a + b)
18     n = V.count()
19     print n
20     V.saveAsTextFile(sys.argv[2])
21     sc.stop()
```

Transformations and Actions

- In terms of implementation, action triggers DAG generation for RDD
- DAG reflects transformations
- Narrow transformations do not involve shuffling
- Wide transformations involve shuffling
- Narrow/wide transformations form stages
- Execution plan is created and passed to scheduler

Persistence

- RDD is materialized when action is performed
- By default RDD is not preserved in the memory
- Unless `persist` method is invoked
- Type of storage and compression can be configured

```
1  #!/usr/bin/python
2  # ex07.py
3
4  from pyspark import SparkConf, SparkContext
5  import sys
6
7  def edge(line):
8      v = [ int(x) for x in line.split(" ") ]
9      return (min(v[0], v[1]), max(v[0], v[1]))
10
11 if __name__ == "__main__":
12     sc = SparkContext()
13     E = sc.textFile(sys.argv[1]).map(edge)
14     E.persist()
15     m = E.count()
16     print m
17     sc.stop()
```

Passing Functions

- Functions/lambdas passed to transformations/actions and their closures are sent to executors whenever needed
- Be careful when using transformation/actions that modify local variables

```
1  #!/usr/bin/python
2  # ex08.py
3
4  from pyspark import SparkConf, SparkContext
5  import sys
6
7  if __name__ == "__main__":
8      sc = SparkContext()
9      c = 0
10
11     # BAD IDEA!
12     def edge(line):
13       global c
14       c += 1
15       v = [ int(x) for x in line.split(" ") ]
16       return (min(v[0], v[1]), max(v[0], v[1]))
17
18     E = sc.textFile(sys.argv[1]).map(edge)
19     n = E.count()
20     sc.stop()
```

Shared Variables

- Shared variables provide limited mechanism to accumulate/broadcast dependent data
- Accumulators do not trigger RDD evaluation!

```
1  #!/usr/bin/python
2  # ex09.py
3
4  from pyspark import SparkConf, SparkContext
5  import sys
6
7  if __name__ == "__main__":
8      sc = SparkContext()
9      cacc = sc.accumulator(0)
10
11     def edge(line):
12       global cacc
13       cacc += 1
14       v = [ int(x) for x in line.split(" ") ]
15       return (min(v[0], v[1]), max(v[0], v[1]))
16
17     E = sc.textFile(sys.argv[1]).map(edge)
18     n = E.count()
19     print n, cacc
20     sc.stop()
```

Shared Variables

```
1  #!/usr/bin/python
2  # ex10.py
3
4  from pyspark import SparkConf, SparkContext
5  import random
6  import sys
7
8  if __name__ == "__main__":
9      sc = SparkContext()
10     L = [random.random() for i in xrange(100000)]
11
12     # NOT EFFICIENT
13     def label(line):
14       global L
15       v = [ int(x) for x in line.split(" ") ]
16       return ((min(v[0], v[1]), max(v[0], v[1])), random.choice(L))
17
18     E = sc.textFile(sys.argv[1]).map(label)
19     E.saveAsTextFile(sys.argv[2])
20     sc.stop()
```

Shared Variables

- Broadcasting

```
1  #!/usr/bin/python
2  # ex11.py
3
4  from pyspark import SparkConf, SparkContext
5  import random
6  import sys
7
8  if __name__ == "__main__":
9      sc = SparkContext()
10     L = sc.broadcast([random.random() for i in xrange(100000)])
11
12     def label(line):
13       global L
14       v = [ int(x) for x in line.split(" ") ]
15       return ((min(v[0], v[1]), max(v[0], v[1])), random.choice(L.value))
16
17     E = sc.textFile(sys.argv[1]).map(label)
18     E.saveAsTextFile(sys.argv[2])
19     sc.stop()
```

For Fun

- Consider the problem of connected components over the list of edges. Implement the initialization step in Spark.