HADOOP WORKSHOP

LINGZI HONG
SCHEDULE

Spark Architecture
Spark Operation
Hands on Challenge
APACHE SPARK

Born at UC Berkeley

Same features of MapReduce and more

Capable of reusing Hadoop ecosystem, e.g. HDFS, YARN
SHORTCOMINGS OF MAPREDUCE

Force your pipeline into Map and Reduce steps

What about other workflows? i.e. join, filter, map-reduce-map
SHORTCOMINGS OF MAPREDUCE

Read from disk for each MapReduce job

What about iterative algorithms? i.e. clustering algorithm, gradient descent etc..
SHORTCOMINGS OF MAPREDUCE

Only native JAVA programming interface

Interactivity? Can we use other languages to analyze?
APACHE SPARK

Force your pipeline into Map and Reduce steps

What about other workflows? i.e. join, filter, map-reduce-map

~20 highly efficient distributed operations, any combination of them
APACHE SPARK

Read from disk for each MapReduce job

What about iterative algorithms? i.e. clustering algorithm, gradient descent etc..

in-memory caching of data, specified by the user
APACHE SPARK

Only native Java programming interface

Interactivity? Can we use other languages to analyze?
Python, Java, Scala, R
ARCHITECTURE
ARCHITECTURE

Cluster Manager:
Amazon EMR: YARN
Cloudera VM: Standalone
RESILIENT DISTRIBUTED DATASETS (RDD)

Distributed

• Distributed across the cluster of machines
• Divided in partitions, atomic data chunks

Resilient

• Recover from errors such as node failure (Directed Acyclic Graph Scheduler)
RESILIENT DISTRIBUTED DATASETS (RDD)

DAG

Nodes are RDDs
Arrows are transformations
Track dependencies for recovery
RESILIENT DISTRIBUTED DATASETS (RDD)

RDD can be generated:

• from HDFS, HBase, JSON, text files, SequenceFiles etc...

• a result of transforming another RDD
EXAMPLE

Scala

```scala
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Python

```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
  .map(lambda word: (word, 1))
  .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```
SCHEDULE

Spark Architecture
Spark Operation
Hands on Challenge
RESILIENT DISTRIBUTED DATASETS (RDD)

Dataset => RDD => ... => RDD => Results

Transformation

Action
CREATE RDD

Create RDD by reading the hdfs file
val textFile = sc.textFile("hdfs://...")

sc: Spark Context
textFile: Spark Context methods
many more: parallelize, range etc...
TRANSFORMATION

RDD are immutable
Can't modify RDD in place
Transform RDD to another RDD
TRANSFORMATION

Narrow Transformation

• No communication between data nodes, e.g. Map, Flatmap

Wide Transformation

• With communications between data nodes, e.g. groupByKey
TRANSFORMATION

Map: apply function to each element of RDD

PySpark

def lower(line):
    return line.lower()

lower_RDD = textFile.map(lower)

Scala

val lower_RDD = textFile.map(line => line.lower())
TRANSFORMATION

RDD Partitions -> map
TRANSFORMATION

Narrow Transformations

- flatMap(func)
- filter(func)
- sample(withReplacement, fraction, seed)
- coalesce(numPartitions)
**TRANSFORMATION**

**flatMap**

```scala
val words = textFile.flatMap(line => line.split(" "))
words.collect()
```

**line1:** “having fun in hadoop workshop”

**line2:** “sunny day”

**output:** [u‘having’, u’fun’, u’in’, u’hadoop’, u’workshop’, u’sunny’, u’day’]

Different from map, the output size is not 1 on 1
TRANFORMATION

filter(func): keep only elements where func is true

Scala

```scala
val count = sc.parallelize(1 to NUM_SAMPLES).filter { _ =>
  val x = math.random
  val y = math.random
  x*x + y*y < 1
}.count()
println(s"Pi is roughly \${4.0 * count / NUM_SAMPLES}\")
```

Python

```python
def inside(p):
    x, y = random.random(), random.random()
    return x**2 + y**2 < 1

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \n .filter(inside).count()
print "Pi is roughly \%f" % (4.0 * count / NUM_SAMPLES)
```
TRANSFORMATION

Coalesce(numPartitions): merge partitions to reduce them to numPartitions
TRANSFORMATION

Coalesce(numPartitions): merge partitions to reduce them to numPartitions

```python
c.sc.parallelize(range(10),4).glom().collect()
Out[]:[[0,1],[2,3],[4,5],[6,7,8,9]]
c.sc.parallelize(range(10),4).coalesce(2).glom().collect()
Out[]:[[0,1,2,3],[4,5,6,7,8,9]]
```
TRANSFORMATION

Wide Transformations

• groupByKey: (K, V)
• reduceByKey(func)
• repartition(numPartitions)
• join: (K, V) and (K, W) => (K, (V, W))
TRANSFORMATION

groupByKey: (K, V) pairs => (K, iterable of all V)
TRANSFORMATION

reduceByKey(func): (K,V) pairs => (K, result of func on all V)

```scala
val lines = sc.textFile("data.txt")
val pairs = lines.map(s => (s, 1))
val counts = pairs.reduceByKey((a, b) => a + b)
```

Different from groupByKey(no func applied)
SHUFFLE

Global redistribution of data
Great impact on performance
PERFORMANCE
ACTION

Final stage of workflow
Triggers execution of the DAG
Returns results to the Driver or Writes to HDFS
ACTIONS

collect(): return all elements …

take(n): return an array with the first n

first(): take(1)

count(): return the number of elements in the dataset

reduce(func): aggregate results

saveAsTextFile(filename): save results to file
**CACHING**

Mark RDD with `.cache()`

Proper cache to speed up the calculation

to where?

- Memory (most common)
- Disk (rare)
- Both (heavy calculations)
CACHING
BROADCAST

Allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks

Transfer just once per executor

Efficient peer-to-peer transfer
**BROADCAST**

```scala
c scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(0)

c scala> broadcastVar.value
res0: Array[Int] = Array(1, 2, 3)
```

```python
>>> broadcastVar = sc.broadcast([1, 2, 3])
< pyspark.broadcast.Broadcast object at 0x102789f10>

>>> broadcastVar.value
[1, 2, 3]
```
**ACCUMULATOR**

shared variable over all clusters

```scala
scala> val accum = sc.longAccumulator("My Accumulator")
accum: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 0, name: Some(My Accumulator), value: 0)

scala> sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum.add(x))
...
10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s

scala> accum.value
res2: Long = 10
```
ACCUMULATOR

shared variable over all clusters

```python
>>> accum = sc.accumulator(0)
>>> accum
Accumulator<id=0, value=0>

>>> sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
...
10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s

>>> accum.value
10
```
LIBRARIES

Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)

Apache Spark

MLLIB EXAMPLE

```scala
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

// Load and parse the data
val data = sc.textFile("data/mllib/kmeans_data.txt")
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()

// Cluster the data into two classes using KMeans
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
println("Within Set Sum of Squared Errors = " + WSSSE)

// Save and load model
clusters.save(sc, "target/org/apache/spark/KMeansExample/KMeansModel")
val sameModel = KMeansModel.load(sc, "target/org/apache/spark/KMeansExample/KMeansModel")
```
SCHEDULE

Spark Architecture
Spark Operation
Hands on Challenge
QUESTIONS

bible+shakes.nopunc dataset

1. Word count
2. How many lines with the word ‘God’
3. PMI

\[ pmi(x; y) = \log \frac{p(x, y)}{p(x)p(y)} \]

<table>
<thead>
<tr>
<th>word 1</th>
<th>word 2</th>
<th>count word 1</th>
<th>count word 2</th>
<th>count of co-occurrences</th>
<th>PMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>puerto</td>
<td>rico</td>
<td>1938</td>
<td>1311</td>
<td>1159</td>
<td>10.0349081703</td>
</tr>
<tr>
<td>hong</td>
<td>kong</td>
<td>2438</td>
<td>2694</td>
<td>2205</td>
<td>9.72831972408</td>
</tr>
<tr>
<td>los</td>
<td>angeles</td>
<td>3501</td>
<td>2808</td>
<td>2791</td>
<td>9.56067615065</td>
</tr>
</tbody>
</table>
QUESTIONS

Ferguson JSON file
1. Parse JSON file
2. Data cleaning
3. Generate vector space
4. K-means clustering