Introduction to Spark

Seminar -236826, 2017-2018
What is Spark?

• Platform for distributed computing
  – In memory, efficient, Map-Reduce inspired
  – Supports interactive and repetitive applications
  – Simple programming model
  – One “universal” data abstraction: RDD

• Supports multiple applications
  – Can be accessed from Java, Scala, Python
  – Libraries: Machine Learning, Graph Analytics, SQL,..
# Cluster Concepts

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>User program built on Spark. Consists of a driver program and executors on the cluster.</td>
</tr>
<tr>
<td>Application jar</td>
<td>A jar containing the user's Spark application. In some cases users will want to create an &quot;uber jar&quot; containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.</td>
</tr>
<tr>
<td>Driver program</td>
<td>The process running the main() function of the application and creating the SparkContext</td>
</tr>
<tr>
<td>Cluster manager</td>
<td>An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)</td>
</tr>
<tr>
<td>Deploy mode</td>
<td>Distinguishes where the driver process runs. In &quot;cluster&quot; mode, the framework launches the driver inside of the cluster. In &quot;client&quot; mode, the submitter launches the driver outside of the cluster.</td>
</tr>
<tr>
<td>Worker node</td>
<td>Any node that can run application code in the cluster</td>
</tr>
<tr>
<td>Executor</td>
<td>A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.</td>
</tr>
<tr>
<td>Task</td>
<td>A unit of work that will be sent to one executor</td>
</tr>
<tr>
<td>Job</td>
<td>A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. save, collect); you'll see this term used in the driver's logs.</td>
</tr>
<tr>
<td>Stage</td>
<td>Each job gets divided into smaller sets of tasks called stages that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.</td>
</tr>
</tbody>
</table>
Standalone mode: [https://spark.apache.org/docs/1.1.0/spark-standalone.html](https://spark.apache.org/docs/1.1.0/spark-standalone.html)

[https://spark.apache.org/docs/1.1.0/cluster-overview.html](https://spark.apache.org/docs/1.1.0/cluster-overview.html)
RDD - Resilient Distributed Dataset

• Distributed immutable collection of objects
• Creation from:
  – Files
  – In-memory objects
• Two kinds of operations:
  – Transformations: lazy, evaluated only when needed
  – Actions: return an answer or write data out, force evaluation
RDD Creation

• In pyspark:
  • allLines = sc.textDile(“enwik8a”)
  • sc is the sparkContex object
  • enwik8a is a file with text lines in the directory from which pyspark is invoked
  • The result is an RDD called allLines
  • No actual activity took place, only a definition
RDD Transformation examples

• isLines = allLines.filter(lambda line: "Israeli" in line)
  - A new RDD is defined by retaining only lines in allLines that contain the string “Israeli”

• Again, only a definition.

• frLines = allLines.filter(lambda line: "French" in line)

• isOrfrLines = frLines.union(isLines)
RDD - Lineage Graph

lines

filter

lines

filter

union

lines

lines
Running in the pyspark shell

```python
>>> import sys
>>> reload(sys)
<module 'sys' (built-in)>
>>> sys.setdefaultencoding('utf8')
>>> allLines = sc.textFile("enwik8a")
>>> isLines = allLines.filter(lambda line: "Israeli" in line)
>>> frLines = allLines.filter(lambda line: "French" in line)
>>> isfrLines = frLines.intersection(isLines)
```
RDD Action examples

• Actions force evaluation and production of a result or an external effect
>>> isfrLines.count()
18

>>> isfrLines.first()
• u"On [[July 20]] [[2004]], Sharon called on [[France | French]] [[Jew]]s to emigrate from France to Israel immediately, in light of an increase in French [[anti-Semitism]] (94 anti-Semitic assaults reported in the first six months of 2004 compared to 47 in 2003). France has the third largest Jewish population (about 600,000 people), after the [[United States]] and [[Israel]]. Sharon claimed that an &quot;unfettered anti-Semitism&quot; reigned in France. The French government responded by describing his comments as &quot;unacceptable&quot;, as did the French representative Jewish organization [[CRIF]], which denied Sharon's claim of intense anti-Semitism in French society. An Israeli spokesperson later claimed that Sharon had been misunderstood. France then postponed a visit by Sharon. Upon his visit, both Sharon and Chirac were described as showing a willingness to put the issue behind them."
RDD Transformations

• Map: can map a RDD of a certain type to a RDD of another type
• Operates one element at a time
• Is carried out in parallel on all the RDD partitions
• Some examples...
Map example

```python
>>> nums = sc.parallelize([1,2,3,4])
>>> squared = nums.map(lambda x: x*x).collect()

>>> for m in squared:
...     print("%i " % (m))
...
1
4
9
16
>>> 
```
Another Map example

```python
>>> nums = sc.parallelize([1,2,3,4])
>>> squared1 = nums.map(lambda x: [x, x*x]).collect()
>>> squared1
[[1, 1], [2, 4], [3, 9], [4, 16]]

>>> for m in squared1:
...     print("%i %i " % (m[0], m[1]) )
...
1 1
2 4
3 9
4 16
>>> ```
flatMap: Multiple elements per mapped element

- Generate multiple elements per each mapped element
- “ Flatten” it all into a single collection
flatMap example

```python
>>> flatm = nums.flatMap(lambda x: [x, x*x]).collect()
>>> flatm
[1, 1, 2, 4, 3, 9, 4, 16]
```
Another flatMap example

```python
>>> lines = sc.parallelize(['"hello world", "again hi", "how are you"'])
>>> words = lines.flatMap(lambda line: line.split(" "))
>>> words.take(5)
['hello', 'world', 'again', 'hi', 'how']
```
RDD Transformations

- Map(), flatMap(), filter(), distinct(), sample(withReplacement, frac, [seed])
- union(), intersection(), subtract(), cartesian()
Examples

```python
>>> a = sc.parallelize([1,1,2,3,4])
>>> b = sc.parallelize([2,2,3,3,3,5])
>>> c = a.union(b)
>>> c
UnionRDD[2] at union at NativeMethodAccessorImpl.java:-2
>>> c1 = c.collect()
>>> c1
[1, 1, 2, 3, 4, 2, 2, 3, 3, 3, 5]
```
Examples

```python
a = sc.parallelize([1,1,2,2,3,4])
b = sc.parallelize([2,2,3,3,3,5])
c = a.subtract(b)
print("a.subtract(b) \n")
c1 = c.collect()
for x in c1:
    print("%i " % (x))

d = a.intersection(b)
d1 = d.collect()
print("a.intersection(b) \n")
for x in d1:
    print("%i " % (x))
```

```
a.subtract(b)
4
1
1

a.intersection(b)

2
3
```
An Application Example

from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("My App0")
sc = SparkContext(conf = conf)

print("Beeginning of program \n")

# encoding=utf8
import sys
reload(sys)
sys.setdefaultencoding('utf8')

lines = sc.textFile("enwik8a")
israelLines = lines.filter(lambda line: "Israel" in line)
collected = israelLines.collect()

f = open("israelMentions", "w")
for index, entry in enumerate(collected):
    f.write(str(index))
    f.write(": ")
    f.write(entry)
    f.write("\n")
f.close()
print("End of program \n")

spark-submit script0.py
At end:
ls -lat
-rw-r--r-- 1 cs236826 users 951205 Sep 7 2015 israelMentions
wc -l israelMentions
2166 israelMentions
Example Actions: reduce()

• Takes an operator on two RDD members and produces one of the same type

```python
>>> a = sc.parallelize([1,3,2,4,5])
>>> sum = a.reduce(lambda x,y: x + y)
>>> sum
15
>>> sum = a.reduce(lambda x,y: x * y)
>>> sum
120
```
Example Actions: fold()

- Takes an operator and a zero value on two RDD members and produces one of the same type

```python
>>> a = sc.parallelize([1,3,2,4,5])
>>> sum = a.fold(1, lambda x,y: x * y)
>>> sum
120
>>> a.fold(1, lambda x,y: x + y)
22
>>> a.fold(0, lambda x,y: x + y)
15
>>> a.fold(-1, lambda x,y: x + y)
8
>>> a = sc.parallelize([1,3,2,4,5],3)
>>> a.fold(1, lambda x,y: x + y)
19
>>> a = sc.parallelize([1,3,2,4,5])
>>> a.fold(1, lambda x,y: x + y)
22
>>> a = sc.parallelize([1,3,2,4,5], 6)
>>> a.fold(1, lambda x,y: x + y)
22
```

Zero value per partition.
If n is specified n+1

The 22 is not very logical, it has to do with some default when nothing is specified.
Example Actions: aggregate()

- Takes a zero value of type we want to return, a function to combine RDD elements with the accumulator, and a second function to combine accumulators

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

>>> sum = a.aggregate((0,0),
...   (lambda acc, value: (acc[0] + value, acc[1] + 1)),
...   (lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])))

>>> sum
(15, 5)
```

```python
>>> sum[0]/float(sum[1])
3.0
```
Example Actions: `countByValue()`

```python
>>> b = sc.parallelize([1,3,2,1,4,1,4,2])
>>> cbv = b.countByValue()
>>> cbv
defaultdict(<type 'int'>, {1: 3, 2: 2, 3: 1, 4: 2})
```
Example Actions: take(), top()

```python
>>> b = sc.parallelize([1,3,2,1,4,1,4,2])
>>> a = b.take(2)
>>> a
[1, 3]
>>> a = b.take(4)
>>> a
[1, 3, 2, 1]

>>> c = b.top(2)
>>> c
[4, 4]
>>> d = b.top(3)
>>> d
[4, 4, 3]
```
Example Actions: takeSample()

```python
>>> a = b.takeSample(False, 3)

>>> a
[2, 1, 2]

>>> a = b.takeSample(False, 3)

>>> a
[2, 4, 1]

>>> a = b.takeSample(False, 3, 5)

>>> a
[3, 1, 1]

>>> a = b.takeSample(False, 3, 5)

>>> a
[3, 1, 1]
```

- Same command, different answers
- 5 is seed
- Same seed
Example Actions: foreach()

```python
colleceed()
[1, 3, 2, 4, 5]
def fun1(x):
    print(str(x*x))
    return True
...
>>> b = a.foreach(fun1)
9
16
4
1
25
```
>>> israelLines = lines.filter(lambda line: "Israel" in line)
>>> print(israelLines.count())
2166
>>> print(israelLines.top(2))
[u"\xc7evik Bir became the Turkish army's deputy chief of general staff shortly after the Somali operation and played a vital role in establishing a Turkish-[[Israel]]i entente against the emerging fundamentalism in the Middle East. Many people in Turkey accused him of actually being a [[Jewish]] [[donmeh]]. For being a staunch defender of secularism in the region, he was awarded the Secularism and Democracy Award of [[1997]] in [[Washington, DC]].", u'|states=[[Armenia]], [[Azerbaijan]], [[Iran]], [[Iraq]], [[Israel]], [[Georgia (country)|Georgia]], [[Lebanon]], [[Palestine (region)|Palestine]], [[Russia]], [[Syria]] and [[Turkey]]']

• Actions ➔ compute the RDD twice
Persistence Levels

• We can persist the data (and re-compute) upon failure
• Different treatment in Python and Scala, Python pickles
• If we run out of storage, persisted data is discarded
• A few levels: MEMORY_ONLY, MEMORY_ONLY_SER, MEMORY_AND_DISK, MEMORY_AND_DISK_SER, DISK_ONLY
• `israelLines = lines.filter(lambda line: "Israel" in line).persist(StorageLevel.MEMORY_ONLY)"`
Persistence Issues

```python
>>> israelLines1 = lines.filter(lambda line: "Israel" in line)
>>> israelLines1.persist(StorageLevel.MEMORY_ONLY)
PythonRDD[63] at RDD at PythonRDD.scala:43
>>> israelLines1.persist(StorageLevel.DISK_ONLY)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/spark/python/pyspark/rdd.py", line 241, in persist
    self._jrdd.persist(javaStorageLevel)
  File "/usr/local/spark/python/lib/py4j-0.8.2.1-src.zip/py4j/java_gateway.py", line 538, in __call__
  File "/usr/local/spark/python/lib/py4j-0.8.2.1-src.zip/py4j/protocol.py", line 300, in get_return_value
py4j.protocol.Py4JJavaError: An error occurred while calling o422.persist.: java.lang.UnsupportedOperationException: Cannot change storage level of an RDD after it was already assigned a level
```
An iterative algorithm that performs many joins, so it is a good use case for RDD partitioning.

The algorithm maintains two datasets, containing:
- (pageID, linkList) elements containing the list of neighbors of each page, and
- (pageID, rank) elements containing the current rank for each page.

It proceeds as follows:
1. Initialize each page’s rank to 1.0.
2. On each iteration, have page p send a contribution of rank(p)/numNeighbors(p) to its neighbors (the pages it has links to).
3. Set each page’s rank to 0.15 + 0.85 * contributionsReceived.
PageRank Computation (Scala)

// Assume that our neighbor list was saved as a Spark objectFile
val links = sc.objectFile[(String, Seq[String])]("links")
  .partitionBy(new HashPartitioner(100))
  .persist()

// Initialize each page's rank to 1.0; since we use mapValues, the resulting RDD will have the same partitioner as links
var ranks = links.mapValues(v => 1.0)
PageRank Computation (Scala)

// Run 10 iterations of PageRank
for (i <- 0 until 10) {
    val contributions = links.join(ranks).flatMap {
        case (pageId, (links, rank)) =>
            links.map(dest => (dest, rank / links.size))
    }
    ranks = contributions.reduceByKey((x, y) => x + y).mapValues(v => 0.15 + 0.85*v)
}

Note: Scala map command

// Write out the final ranks
ranks.saveAsTextFile("ranks")
The join is on the page in the 2-tuples, such as \([p, [p_1, p_2, p_3]]\) and \([p, rankVal]\) \(\rightarrow\) \([p, [[p_1, p_2, p_3], rankVal]]\) \(\rightarrow\) \([[[p_1, rankVal/3], [p_2, rankVal/3], [p_3, rankVal/3]]]\).

In fact each page gives rise to such a blue list of pairs and they are all made into one big list (flat).

Then, reduceByKey accumulates the values for each page, and then maps each such value into 0.85 of this value + 0.15 (total preserved).
SERVERLESS
Severless Spark:
a different take on the Serverless idea

https://www.youtube.com/watch?v=oFOgKB4OlBM