Best Weather for Goals?

Willem Hendriks
Data Scientist IBM

willem.hendriks@nl.ibm.com
https://github.com/willemhendriks
“More data usually beats better algorithms”
Anand Rajaraman (when teaching at Stanford)
http://anand.typepad.com/datawocky/2008/03/more-data-usual.html
In The News: No more Silent Movies!

Artificial intelligence produces realistic sounds to fool humans.
Creative Jobs are at risk, not engineers!

Movie written by algorithm turns out to be hilarious and intense

For Sunspring's exclusive debut on Ars, we talked to the filmmakers about collaborating with an AI.

Sunspring | A Sci-Fi Short Film Starring Thomas Middleditch
Apache Spark™ is a fast and general engine for large-scale data processing.

**Speed**
Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

**Ease of Use**
Write applications quickly in Java, Scala, Python, or R.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.
Spark 1.6.1 Released (9 March 2016)

Documentation

- [SPARK-12351] - Add documentation of submitting Mesos jobs with cluster mode
- [SPARK-12507] - Expose closeFileAfterWrite and allowBatching configurations for Streaming
- [SPARK-12722] - Typo in Spark Pipeline example
- [SPARK-12758] - Add note to Spark SQL Migration section about SPARK-11724
- [SPARK-12814] - Add deploy instructions for Python in Flume integration doc
- [SPARK-12894] - Add deploy instructions for Python in Kinesis integration doc
- [SPARK-13214] - Fix dynamic allocation docs
- [SPARK-13274] - Fix Aggregator Links on GroupedDataset Scala API
- [SPARK-13350] - Configuration documentation incorrectly states that PYSPARK_PYTHON's default is "python"
- [SPARK-13439] - Document that spark.mesos.uris is comma-separated

Improvement

- [SPARK-6273] - Improve documentation examples for LinearRegression
- [SPARK-11790] - Provide type aliases in org.apache.spark.sql.types for backwards compatibility
- [SPARK-12120] - Improve exception message when failing to initialize HiveContext in PySpark
- [SPARK-12411] - Refactor executor heartbeat rpc timeout
- [SPARK-12450] - Un-persist broadcasted variables in KMeans
- [SPARK-12701] - Logging FileAppender should use join to ensure thread is finished
- [SPARK-12834] - Use type conversion instead of SerDe of Pickle to transform JavaArray and JavaList
- [SPARK-12932] - Bad error message with trying to create Dataset from RDD of Java objects that are not bean-compliant
- [SPARK-13094] - No encoder implicit for Seq[Primitive]
- [SPARK-13279] - Scheduler does O(N^2) operation when adding a new task set (making it prohibitively slow for scheduling 200K tasks)

New Feature

- [SPARK-10359] - Enumerate Spark's dependencies in a file and diff against it for new pull requests

Task

- [SPARK-13474] - Update packaging scripts to stage artifacts to home apache.org
Why People love Spark

1 - Nice short snappy code
(Python / Scala / R)

2 - Universal Connectors
(Databases / Hadoop / Files / Messaging Systems)

3 - Interactive Shell

4 - Beautiful Library
(Linear Algebra & ML)
Why Madrid loves Spark?

1 - Nice short snappy code
   (Python / Scala / R)

2 - Universal Connectors
   (Databases / Hadoop / Files / Messaging Systems)

3 - Interactive Shell

4 - Beautiful Library
   (Linear Algebra & ML)
What can we do now, We could not do some years ago?

Why would a person/company invest in Spark Technology?
<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yin Huai</td>
<td>Databricks</td>
</tr>
<tr>
<td>Shane Huang</td>
<td>Intel</td>
</tr>
<tr>
<td>Andrew Or</td>
<td>Databricks</td>
</tr>
<tr>
<td>Kay Ousterhout</td>
<td>UC Berkeley</td>
</tr>
<tr>
<td>Sean Owen</td>
<td>CloudBees</td>
</tr>
<tr>
<td>Nick Pentreath</td>
<td>Mxit</td>
</tr>
<tr>
<td>Imran Rashid</td>
<td>CloudBees</td>
</tr>
<tr>
<td>Charles Reiss</td>
<td>UC Berkeley</td>
</tr>
<tr>
<td>Josh Rosen</td>
<td>Databricks</td>
</tr>
<tr>
<td>Sandy Ryza</td>
<td>Cloudera</td>
</tr>
<tr>
<td>Kousuke Saruta</td>
<td>NTT Data</td>
</tr>
<tr>
<td>Prashant Sharma</td>
<td>IBM</td>
</tr>
<tr>
<td>Ram Sriharsha</td>
<td>Hortonworks</td>
</tr>
<tr>
<td>DB Tsai</td>
<td>Netflix</td>
</tr>
<tr>
<td>Marcelo Vanzin</td>
<td>Cloudera</td>
</tr>
</tbody>
</table>

**spark-netezza**

A data source library to load data into Spark SQL DataFrames from IBM® Netezza® database. This data source library is implemented using Netezza external table mechanism to transfer data from the Netezza host system to the Spark system optimally.

**Binary download:**

You can download spark-netezza assembly jars from here:

<table>
<thead>
<tr>
<th>Spark Version</th>
<th>Release #</th>
<th>Binary Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.2+</td>
<td>v0.1.1</td>
<td>spark-netezza-assembly-0.1.1.jar</td>
</tr>
</tbody>
</table>
Key Components Spark

It doesn't matter how you access Spark, Command line, Spark-submit Or notebook
Quiz!

With Apache Spark, Hadoop will be useless (soon)

True -or- False?
Quiz!

Different Data-Sources (MySQL / Hadoop / Files / Cassandra) Requires Different Spark Programming Approaches

True -or- False?
Quiz!

**Scala** is the Best Way to Spark, **Python** is for hipsters.... **R** is for pirates.... **Java** for old folks....

True -or- False ?
More than 1 way to Spark

<table>
<thead>
<tr>
<th>RDDs</th>
<th>DataFrames</th>
</tr>
</thead>
<tbody>
<tr>
<td>little slower</td>
<td>little faster</td>
</tr>
<tr>
<td></td>
<td>(language independent)</td>
</tr>
<tr>
<td>MLlib</td>
<td>sparkML</td>
</tr>
<tr>
<td>chain of RDDs</td>
<td>pipelines!</td>
</tr>
<tr>
<td></td>
<td>Nice chain of transformations &amp; fitters</td>
</tr>
<tr>
<td>Freedom (map/reduce)</td>
<td>structured</td>
</tr>
<tr>
<td>Programmers Start</td>
<td>SQL'er / R start</td>
</tr>
</tbody>
</table>

![Pipeline Diagram](chart.png)
Thinking in Spark

Don't think as Variables & Functions, but in Transformations & Actions

A **RDD** is a pointer to your data, not the data itself (Hadoop / File / Database)

A transformation on a **RDD** will create a new **RDD**, by creating a new pointer (again no data)

It adds tubes & flasks, but still no data/fluid is flowing.

Not until an **action** is called upon an RDD, spark will not flow data through your chain of RDD's... (lazy evaluation)

After an **action**, Spark lets the fluid/data flow.
How can we 'hold' some fluid/data inside a flask, to later quickly use it?

rdd 1
rdd 2
rdd 3
rdd 4
rdd 5
How can we 'hold' some fluid/data inside a flask, to later quickly use it?

```
persist() or 
cache()
```
Actions & Transformations

Transformations:

`map(f, preservesPartitioning=False)`
Return a new RDD by applying a function to each element of this RDD.

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

`filter(f)`
Return a new RDD containing only the elements that satisfy a predicate.

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

Actions:

`count()`
Return the number of elements in this RDD.

```python
>>> sc.parallelize([2, 3, 4]).count()
3
```

`first()`
Return the first element in this RDD.

```python
>>> sc.parallelize([2, 3, 4]).first()
2
>>> sc.parallelize([]).first()
Traceback (most recent call last):
...
ValueError: RDD is empty
```
RDD's

- Contains Elements
- Transformations & Actions
  .persist()
- Mllib to create models (Machine Learning)
- Key/Value, a special case, needed for joins
- Freedom (as a opposite to Data-Frames)
### Partition A

- [1, 4, 6]
- [5, 7, 3]
- [7, 6, 5]
- [8, 8, 8]
- [3, 8, 5]

### Partition B

- [2, 4, 6]
- [5, 4, 3]
- [7, 6, 5]
- [8, 7, 8]
- [3, 4, 5]

### Partition C

- [2, 4, 6]
- [5, 4, 3]
- [7, 1, 3]
- [1, 1, 3]
- [3, 4, 3]

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double, Float</td>
<td>1.3</td>
</tr>
<tr>
<td>List</td>
<td>[12, “Blue”, 3.03]</td>
</tr>
<tr>
<td>String</td>
<td>“23;”AEFED”</td>
</tr>
<tr>
<td>Numpy Array</td>
<td>Array(1, 4, 3, 3, 5, ...)</td>
</tr>
<tr>
<td>Spark’s Labelled Points</td>
<td>LabelledPoint( “GO OD”, “Chablis”, 12.3 )</td>
</tr>
<tr>
<td>Any (basic) python</td>
<td>......</td>
</tr>
<tr>
<td>(Key, Value)</td>
<td>Special Case Key = everything Value = everything</td>
</tr>
</tbody>
</table>

**getNumPartitions(self)**

Returns the number of partitions in RDD.

```python
>>> rdd = sc.parallelize([1, 2, 3, 4], 2)
>>> rdd.getNumPartitions()
2
```
## Transformations

The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc (Scala, Java, Python, R) and pair RDD functions doc (Scala, Java) for details.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map(func)</code></td>
<td>Return a new distributed dataset formed by passing each element of the source through a function <code>func</code>.</td>
</tr>
<tr>
<td><code>filter(func)</code></td>
<td>Return a new dataset formed by selecting those elements of the source on which <code>func</code> returns true.</td>
</tr>
<tr>
<td><code>flatMap(func)</code></td>
<td>Similar to <code>map</code>, but each input item can be mapped to 0 or more output items (so <code>func</code> should return a <code>Seq</code> rather than a single item).</td>
</tr>
<tr>
<td><code>mapPartitions(func)</code></td>
<td>Similar to <code>map</code>, but runs separately on each partition (block) of the RDD, so <code>func</code> must be of type <code>Iterator&lt;T&gt; =&gt; Iterator&lt;U&gt;</code> when running on an RDD of type <code>T</code>.</td>
</tr>
<tr>
<td><code>mapPartitionsWithIndex(func)</code></td>
<td>Similar to <code>mapPartitions</code>, but also provides <code>func</code> with an integer value representing the index of the partition, so <code>func</code> must be of type <code>(Int, Iterator&lt;T&gt;) =&gt; Iterator&lt;U&gt;</code> when running on an RDD of type <code>T</code>.</td>
</tr>
<tr>
<td><code>sample(withReplacement, fraction, seed)</code></td>
<td>Sample a fraction <code>fraction</code> of the data, with or without replacement, using a given random number generator <code>seed</code>.</td>
</tr>
<tr>
<td><code>union(otherDataset)</code></td>
<td>Return a new dataset that contains the union of the elements in the source dataset and the argument.</td>
</tr>
<tr>
<td><code>intersection(otherDataset)</code></td>
<td>Return a new RDD that contains the intersection of elements in the source dataset and the argument.</td>
</tr>
<tr>
<td><code>distinct([numTasks])</code></td>
<td>Return a new dataset that contains the distinct elements of the source dataset.</td>
</tr>
<tr>
<td><code>groupByKey([numTasks])</code></td>
<td>When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable&lt;V&gt;) pairs. <strong>Note:</strong> If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. <strong>Note:</strong> By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numTasks</code> argument to set a different number of tasks.</td>
</tr>
<tr>
<td><code>reduceByKey(func, [numTasks])</code></td>
<td>When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <code>func</code>, which must be of type <code>(V, V) =&gt; V</code>. Like in <code>groupByKey</code>, the number of reduce tasks is configurable through an optional second argument <code>numTasks</code>.</td>
</tr>
</tbody>
</table>
### Actions


<table>
<thead>
<tr>
<th>Action</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reduce(func)</code></td>
<td>Aggregate the elements of the dataset using a function <code>func</code> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.</td>
</tr>
<tr>
<td><code>collect()</code></td>
<td>Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.</td>
</tr>
<tr>
<td><code>count()</code></td>
<td>Return the number of elements in the dataset.</td>
</tr>
<tr>
<td><code>first()</code></td>
<td>Return the first element of the dataset (similar to <code>take(1)</code>).</td>
</tr>
<tr>
<td><code>take(n)</code></td>
<td>Return an array with the first <code>n</code> elements of the dataset.</td>
</tr>
<tr>
<td><code>takeSample(withReplacement, num, [seed])</code></td>
<td>Return an array with a random sample of <code>num</code> elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.</td>
</tr>
<tr>
<td><code>takeOrdered(n, [ordering])</code></td>
<td>Return the first <code>n</code> elements of the RDD using either their natural order or a custom comparator.</td>
</tr>
<tr>
<td><code>saveAsTextFile(path)</code></td>
<td>Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.</td>
</tr>
<tr>
<td><code>saveAsSequenceFile(path)</code></td>
<td>Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDS of key-value pairs that implement Hadoop’s Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc.).</td>
</tr>
<tr>
<td><code>saveAsObjectFile(path)</code></td>
<td>Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using SparkContext::objectFile().</td>
</tr>
<tr>
<td><code>countByKey()</code></td>
<td>Only available on RDDS of type <code>(K, V)</code>. Returns a hashmap of <code>(K, Int)</code> pairs with the count of each key.</td>
</tr>
<tr>
<td><code>foreach(func)</code></td>
<td>Run a function <code>func</code> on each element of the dataset. This is usually done for side effects such as updating an <code>Accumulator</code> or interacting with external storage systems.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td><strong>Note:</strong> modifying variables other than Accumulators outside of the <code>foreach()</code> may result in undefined behavior. See Understanding closures for more details.</td>
</tr>
</tbody>
</table>
Let's take some Data!
Let's take some Data!
What is are the best weather conditions for Spanish teams?

Does the weather have influence on the number of goals in UEFA Champions League Matches?