Apache Spark : Fast and Easy Data Processing

Sujeet Maniyam
Elephant Scale LLC
sujee@elephantscale.com
http://elephantscale.com
Spark

- Fast & Expressive Cluster computing engine
- Compatible with Hadoop
- Came out of Berkeley AMP Lab
- Now Apache project
- Version 1.1 just released (Sep 2014)
## Comparison With Hadoop

<table>
<thead>
<tr>
<th>Hadoop</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Storage + Distributed Compute</td>
<td>Distributed Compute Only</td>
</tr>
<tr>
<td>MapReduce framework</td>
<td>Generalized computation</td>
</tr>
<tr>
<td>Usually data on disk (HDFS)</td>
<td>On disk / in memory</td>
</tr>
<tr>
<td>Not ideal for iterative work</td>
<td>Great at Iterative workloads</td>
</tr>
<tr>
<td></td>
<td>(machine learning ..etc)</td>
</tr>
<tr>
<td>Batch process</td>
<td>- Upto 10x faster for data on disk</td>
</tr>
<tr>
<td></td>
<td>- Upto 100x faster for data in memory</td>
</tr>
<tr>
<td></td>
<td>Compact code</td>
</tr>
<tr>
<td></td>
<td>Java, Python, Scala supported</td>
</tr>
<tr>
<td></td>
<td>Shell for ad-hoc exploration</td>
</tr>
</tbody>
</table>
Spark Vs Hadoop

- Spark is ‘easier’ than Hadoop
- ‘friendlier’ for data scientists
- Interactive shell (adhoc exploration)
- API supports multiple languages
  - Java, Scala, Python
- Great for small (Gigs) to medium (100s of Gigs) data
Spark Vs. Hadoop

Hadoop Map Reduce

Map → Reduce → Map → Reduce → HDFS

Spark

compute 1 → cached → compute 2 → HDFS or S3
Is Spark Replacing Hadoop?

- Right now, Spark runs on Hadoop / YARN
  - Complimentary
- Can be seen as generic MapReduce
- Spark is really great if data fits in memory (few hundred gigs),
- People are starting to use Spark as their only compute platform
- Future ???
Hadoop + Yarn: Universal OS for Cluster Computing

- Batch (mapreduce)
- Streaming (storm, S4)
- In-memory (spark)

YARN

HDFS
Bit of History of Spark

Sep 2012
: Spark 0.6

Feb 2014
: Apache Top Level Project

Jun 2013
: Apache incubator project

May 2014
: v1.0

Sep 2014
: v1.1
Hypo-meter 😊

Technology

Hype-o-meter
Spark Job Trends

spark Job Trends

Scale: Absolute - Relative

Job Trends from Indeed.com

Percentage of Matching Job Postings

2015 SNIA Analytics and Big Data Summit. © Elephant Scale LLC. All Rights Reserved.
Spark Eco-System

- Schema / sql
  - Spark SQL
- Real Time
  - Spark Streaming
- Machine Learning
  - ML lib

Spark Core
Spark Core

- Distributed compute engine
- Sort / shuffle algorithms
- In case of node failures -> re-computes missing pieces
Spark Streaming

- Process data streams in **real time**
- Stock ticks / click streams …etc
Machine Learning (ML Lib)

- Out of the box ML capabilities!
- Lots of common algorithms are supported
- Classification / Regressions
  - Linear models (linear R, logistic regression, SVM)
  - Decision trees
- Collaborative filtering (recommendations)
- K-Means clustering
- ...
- More to come
Spark Architecture

1) Standalone
2) Yarn
3) Mesos
Spark Architecture

- Multiple ‘applications’ can run at the same time
- Driver (or ‘main’) launches an application
- Each application gets its own ‘executor’
  - Isolated (runs in different JVMs)
  - Also means data can not be shared across applications
- Cluster Managers:
  - multiple cluster managers are supported
  - 1) Standalone : simple to setup
  - 2) YARN : on top of Hadoop
  - 3) Mesos : General cluster manager (AMP lab)
Spark Data Model : RDD

- Resilient Distributed Dataset (RDD)
- Can live in
  - Memory (best case scenario)
  - Or on disk (FS, HDFS, S3 …etc)
- Each RDD is split into multiple partitions
- Partitions may live on different nodes
- Partitions can be computed in parallel on different nodes
RDD : Loading

- Use Spark context to load RDDs from disk / external storage

```scala
val sc = new SparkContext(...) 
val f = sc.textFile("/data/input1.txt") // single file 
sc.textFile("/data/") // load all files under dir 
sc.textFile("/data/*.log") // wild card matching
```
RDD Operations

- Two kinds of operations on RDDs
  - 1) Transformations
    - Create a new RDD from existing ones (e.g. Map)
  - 2) Actions
    - E.g. Returns the results to clients (e.g. Reduce)
- Transformations are lazy. Actions force transformations
Transformations / Actions

RDD 1 → Transformation 1 (map) → Action 1 (collect) → RDD 3

Client
# RDD Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>Filters through each record (aka grep)</td>
<td>f.filter( line =&gt; line.contains(&quot;ERROR&quot;))</td>
</tr>
<tr>
<td>union</td>
<td>Merges two RDDs</td>
<td>rdd1.union(rdd2)</td>
</tr>
<tr>
<td></td>
<td>...see docs ...</td>
<td></td>
</tr>
</tbody>
</table>
### RDD Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>count()</td>
<td>Counts all records in an rdd</td>
<td>f.count()</td>
</tr>
<tr>
<td>first()</td>
<td>Extract the first record</td>
<td>f.first()</td>
</tr>
<tr>
<td>take(n)</td>
<td>Take first N lines</td>
<td>f.take(10)</td>
</tr>
<tr>
<td>collect()</td>
<td>Gathers all records for RDD. All data has to fit in memory of ONE machine (don’t use for big data sets)</td>
<td>f.collect()</td>
</tr>
</tbody>
</table>

.... See documentation ..
Partitions Explained

1G file

32M 32M 32M 32M

Task Task Task Task

Result
RDD : Saving

- `saveAsTextFile()` and `saveAsSequenceFile()`
- `f.saveAsTextFile("/output/directory")`  // a directory
- Output usually is a directory
  - RDDs will be saved as multiple files in the dir
  - Each partition → one output file
Caching of RDDs

- RDDs can be loaded from disk and computed
  - Hadoop mapreduce model
- Also RDDs can be cached in memory
- Subsequent operations are much faster
  - f.persist() // on disk or memory
  - f.cache() // memory only
- In memory RDDs are great for iterative workloads
  - Machine learning algorithms

![Graph showing running time comparison between Hadoop and Spark]
Demo Time!
Demo : Spark-shell

- Invoke spark shell
- Load a data set
- Do basic operations (count / filter)
Demo: RDD Caching

- From Spark shell
- Load an RDD
- Demonstrate the difference between cached and non-cached
Quick Word count

val input = sc.textFile("...")
val counts = input.flatMap(
    line => line.split(" ").
    map(word => (word, 1)).
    reduceByKey(_+_)
)
Thanks !

Sujee Maniyam
sujee@elephantscale.com
http://elephantscale.com

Expert consulting & training in Big Data
Credits

- [http://www.strategictechplanning.com](http://www.strategictechplanning.com)