Mike Carey
UC Irvine
mjcarey@ics.uci.edu

NOTE:
Many thanks to Databricks and Profs. Michael Franklin (past UC Berkeley AMPLab Director) and Matei Zaharia (MIT/Stanford, a.k.a. Dr. Spark) for providing some of this material and granting us permission to use it!
Announcements

• We are in Big Data Analytics territory!

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>W 5/19</td>
<td>Big Data Analytics: Google, MapReduce, HDFS</td>
<td>Big Data Platforms paper (skim)</td>
</tr>
<tr>
<td>M 5/24</td>
<td>Big Data Analytics: Spark &amp; SparkSQL</td>
<td>Spark Overview paper (skim)</td>
</tr>
<tr>
<td>W 5/26</td>
<td>Big Data Analytics: Spark &amp; DataFrames</td>
<td>Databricks and Spark materials (as needed)</td>
</tr>
<tr>
<td>W 6/02</td>
<td>Data Stream Systems: Spark Structured Streaming</td>
<td>Databricks and Spark materials (as needed)</td>
</tr>
<tr>
<td>Overflow</td>
<td>Column stores, search, message streams, timeseries/IoT, ...</td>
<td>Google to find out more...</td>
</tr>
<tr>
<td>M 6/07</td>
<td>Final Exam (Cumulative)</td>
<td>4:00-6:00 PM -- be there!!!</td>
</tr>
</tbody>
</table>

• HW #5 is in flight – and due on Thursday evening

<table>
<thead>
<tr>
<th>HW5</th>
<th>Mo 5/17</th>
<th>Th 5/27 (11:59 PM)</th>
<th>Neo4J</th>
<th>HW5 Setup</th>
<th>HW5 Details Template</th>
<th>HW5 Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW6</td>
<td>Th 5/27</td>
<td>Mo 6/07 (11:59 PM)</td>
<td>Spark</td>
<td>HW6 Setup</td>
<td>HW6 Details</td>
<td>HW6 Solution</td>
</tr>
</tbody>
</table>

• Course evaluations open: One of your 5 participation points!

• Today: *Apache Spark*...
More to Life than MapReduce?

- **Ex:** Bulk Synchronous Programming (BSP) platforms for Big Graph analytics:
  
  "Think Like a Vertex"
  - Receive messages
  - Update state
  - Send messages

("Big" is the platform's concern)

- Quite a few BSP-based platforms have been developed
  - Pregel (Google)
  - Giraph (Facebook, LinkedIn, Twitter, Yahoo!, ...)
  - Distributed GraphLab (CMU, Washington)
  - ...

- Machine learning (ML) has similar iterative requirements
Subsequent Platforms (cont.)

- Iteration (e.g., BSP or ML) on Map/Reduce*

ML Processing in Map/Reduce

Initial Model $w^{(0)}$

Training Data

Map

Reduce

Learned Model

$w^{(1)}$

$w^{(2)}$

$w^{(3)}$

M. Carey, Spring 2021: CS122D
ML Overheads in Map/Reduce

Initial Model

Training Data

Map

Reduce

Learned Model

Repeatedly load same data

M. Carey, Spring 2021: CS122D
**ML Overheads in Map/Reduce**

Redundantly save output between stages (jobs)
The Reaction from Berkeley

(My favorite scene from Dreamworks’ *Penguins of Madagascar*...)
The Reaction from Berkeley (cont.)

- Spark, for *in-memory* cluster computing – for doing repetitive data analyses, iterative machine learning tasks, ...

---

**Word Count**

In this example, we use a few more transformations to build a dataset of (String, Int) pairs called counts and then save it to a file.

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```
The Reaction from Berkeley (cont.)

- **Spark**, for *in-memory* cluster computing – for doing repetitive data analyses, iterative machine learning tasks, ...

(Spark specially gained traction for scaling ML)
Berkeley Data Analytics Stack (BDAS)

• Specializing MapReduce leads to stovepiped systems
• Instead, **generalize** MapReduce
  1. Richer programming model
     ➔ Fewer systems to master
  2. Data sharing
     ➔ Less data movement leads to better performance

• Spark initially showed 10x performance improvement on existing HDFS data with no migration
• **Another MR generalization**: Apache Flink
Spark Dataflow Operators

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save
- ...
Spark = Memory + Dataflow
Spark = Memory + Dataflow

Training Data (HDFS) → Map → Reduce → Map → Reduce → Map → Reduce → Storage

Cached load
Spark = Memory + Dataflow

Result: 10-100× speedup over Hadoop MapReduce
Fault-Tolerance in Spark

- **RDDs**: Immutable collections of objects that can be stored in memory (or disk) across a cluster
  - Built via parallel transformations (map, filter, ...)
  - Automatically rebuilt on (partial) failure (→ lineage)

```
messages = textFile(...).filter(_.contains("error"))
  .map(_.split('t')(2))
```

- HadoopRDD
  - path = hdfs://...

- FilteredRDD
  - func = _.contains(...)

- MappedRDD
  - func = _.split(...)

Apache Spark: User’s Perspective

• Overview
  • Platform basics
  • Languages and interfaces

• SparkSQL (today)
  • SparkSQL basics (~SQL...!)
  • Nested data in SparkSQL

• Spark Dataframes (next time)
  • What are Dataframes? (Python/Pandas)
  • Dataframes in the Spark world

(Q: Prior Pandas experience?)
What is Apache Spark?

1. **Unified platform**: Same engine for simple data loading, SQL queries, ML, and streaming computation – including possible cross-optimization.
What is Apache Spark (cont.)?

2. **Computing engine**: “…Spark carefully limits its scope to a computing engine ... Spark only handles loading data from storage systems and performing computation on it, not permanent storage as the end itself.”
What is Apache Spark *(cont.)*?

3. **Libraries**: SQL and structured data (SparkSQL), machine learning (MLib), stream processing (Spark Streaming and Structured Streaming), graph analytics (GraphX), and a variety of open source libraries.

![Diagram showing libraries and technologies related to Apache Spark](image-url)
Basic Architecture

- Cluster manager controls the physical machines and allocates resources to Spark Applications
- Driver process runs the main() function and oversees the Executors
- Executors perform the actual computation

- Proxy object (spark) for the app
- Supported languages/APIs include
  - Scala (Spark’s native language)
  - Java (since Scala is JVM-based)
  - Python
  - SQL
  - R (libraries)
Spark Data Abstractions

- Core (partitioned) data abstractions and APIs
  - **RDD**: A sequence of data objects (with caching and lineage) that consist of one or more types that are located across a cluster of machines
  - **DataFrame**: Similar in concept to the DataFrame notion found in the Pandas Python library and the R language (*more next time*)
  - **Dataset**: A combination of DataFrame and RDD that provides the typed interface available in RDDs along with the convenience of DataFrames
  - **SQL Table**: Similar to a SQL table (SparkSQL handle for a DataFrame)
SparkSQL

• Spark’s module for “structured data processing”
  • Higher level API than RDDs
  • Relies on schema (inferred or provided)
  • Then able to perform extra optimizations

• Supports both SQL and Dataset (and Dataframe) APIs
  • Same execution engine involved in both cases
  • Allows developers to switch between them as desired, based on what’s most natural for the task at hand

• We’ll be exploring SparkSQL using their Community Edition’s “notebook in the cloud” capabilities
  • Two “Big Data” examples: Hoofers sailing club (relational) and SQL++ order management (JSON)
  • Notebook and little data files will appear on the wiki
  • Let’s go....
Questions?