Introduction to Big Data with Apache Spark
This Lecture

The Big Data Problem
Hardware for Big Data
Distributing Work
Handling Failures and Slow Machines
Map Reduce and Complex Jobs
Apache Spark
Some Traditional Analysis Tools

- Unix shell commands, Pandas, R

All run on a single machine!
The Big Data Problem

• Data growing faster than computation speeds

• Growing data sources
  » Web, mobile, scientific, …

• Storage getting cheaper
  » Size doubling every 18 months

• But, stalling CPU speeds and storage bottlenecks
Big Data Examples

- Facebook’s daily logs: 60 TB
- 1,000 genomes project: 200 TB
- Google web index: 10+ PB
- Cost of 1 TB of disk: ~$35
- Time to read 1 TB from disk: 3 hours (100 MB/s)
The Big Data Problem

- A single machine can no longer process or even store all the data!
- Only solution is to distribute data over large clusters
Google Datacenter

How do we program this thing?
Hardware for Big Data

Lots of hard drives … and CPUs
Hardware for Big Data

One big box?
(1990’s solution)

But, expensive
  » Low volume
  » All “premium” hardware

And, still not big enough!

Image: Wikimedia Commons / User:Tonusamuel
Hardware for Big Data

**Consumer-grade** hardware
Not “gold plated”

Many desktop-like servers
**Easy to add capacity**
**Cheaper** per CPU/disk

**Complexity in software**
Problems with Cheap Hardware

Failures, Google’s numbers:
1-5% hard drives/year
0.2% DIMMs/year

Network speeds versus shared memory
Much more latency
Network slower than storage

Uneven performance
What’s Hard About Cluster Computing?

• How do we split work across machines?
How do you count the number of occurrences of each word in a document?

“\[I \text{ am } \text{Sam} \\
I \text{ am } \text{Sam} \\
\text{Sam} \text{ I am} \\
\text{Do you like} \\
\text{Green eggs and ham?}\]”

\[
\begin{align*}
I: & 3 \\
am: & 3 \\
\text{Sam}: & 3 \\
do: & 1 \\
you: & 1 \\
\text{like}: & 1 \\
\ldots
\end{align*}
\]
One Approach: Use a Hash Table

“I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?”
One Approach: Use a Hash Table

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?"

\{l : 1\}
One Approach: Use a Hash Table

“I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?”

{I: 1,
am: 1}
One Approach: Use a Hash Table

“\text{I am } \textbf{Sam} \text{ I am Sam Sam I am Do you like Green eggs and ham?}”

\{I: 1, \text{am: 1, Sam: 1}\}
One Approach: Use a Hash Table

“I am Sam
Sam I am
Do you like Green eggs and ham?”

{I: 2,
am: 1,
Sam: 1}
What if the Document is Really Big?

“I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
…”
What if the Document is Really Big?

Machines 1-4

{I: 3, am: 3, Sam: 3}
{do: 2, ...
{Sam: 1, 
{Would:1, 

Machine 5

{l: 6, am: 4, Sam: 4, do: 3 ...

What’s the problem with this approach?
What if the Document is Really Big?

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
…"

Machines 1 - 4

{I: 3,
am: 3,
Sam: 3}

{do: 2,
...
}

{Sam: 1,
...
}

{Would: 1,
...
}

Machine 5

{l: 6,
am: 4,
Sam: 4,
do: 3
...
}

Results have to fit on one machine
What if the Document is Really Big?

“"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
…”"
What if the Document is Really Big?

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
..."

Use Divide and Conquer!!
What if the Document is Really Big?

“...I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
...”

Machines 1 - 4

Use Divide and Conquer!!

Machines 1 - 4
What if the Document is Really Big?

“I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
…”

Use Divide and Conquer!!
What if the Document is Really Big?

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
..."

---

Google
Map Reduce 2004

http://research.google.com/archive/mapreduce.html
Map Reduce for Sorting

I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
...

"What word is used most?"
What’s Hard About Cluster Computing?

• How to divide work across machines?
  » Must consider network, data locality
  » Moving data may be very expensive

• How to deal with failures?
  » 1 server fails every 3 years ➔ with 10,000 nodes see 10 faults/day
  » Even worse: stragglers (not failed, but slow nodes)
How Do We Deal with Failures?

“I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
…”

{do: 1,
 you: 1,
…}

{Would: 1,
 you: 1,
…}

{Would: 1,
 you: 1,
…}
How Do We Deal with Machine Failures?

“...I am Sam
I am Sam
Sam I am

Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
...”

{I: 1,
am: 1,
...}

{Would: 1,
you: 1,
...}

{Would: 1,
you: 1,
...}

Launch another task!
How Do We Deal with Slow Tasks?

“I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
…”
How Do We Deal with Slow Tasks?

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and ham?
I do not like them
Sam I am
I do not like
Green eggs and ham
Would you like them
Here or there?
..."

{I: 1,
am: 1,
...

{do: 1,
you: 1,
...

{Would: 1,
you: 1,
...

{Would: 1,
you: 1,
...

Launch another task!
Map Reduce: Distributed Execution

Each stage passes through the hard drives
Map Reduce: Iterative Jobs

- Iterative jobs involve a lot of disk I/O for each repetition

Disk I/O is very slow!
Apache Spark Motivation

- Using Map Reduce for complex jobs, interactive queries and online processing involves *lots of disk I/O*

Interactive mining

Stream processing

Also, iterative jobs

Disk I/O is very slow
Tech Trend: Cost of Memory


Lower cost means can put more memory in each server

Memory

2010: 1 ¢/MB

disk

flash
Hardware for Big Data

Lots of hard drives … and CPUs

… and memory!
Opportunity

• Keep more data *in-memory*

• Create new distributed execution engine:

Use Memory Instead of Disk

1. Input
2. HDFS read
3. iteration 1
4. HDFS write
5. iteration 2
6. HDFS read
7. HDFS write
8. . .

9. Input
10. HDFS read
11. query 1
12. result 1
13. query 2
14. result 2
15. query 3
16. result 3
17. . .
In-Memory Data Sharing

10-100x faster than network and disk
Resilient Distributed Datasets (RDDs)

- Write programs in terms of operations on distributed datasets
- Partitioned collections of objects spread across a cluster, stored in memory or on disk
- RDDs built and manipulated through a diverse set of parallel transformations (map, filter, join) and actions (count, collect, save)
- RDDs automatically rebuilt on machine failure
The Spark Computing Framework

• Provides programming abstraction and parallel runtime to hide complexities of fault-tolerance and slow machines

• “Here’s an operation, run it on all of the data”
  » I don’t care where it runs (you schedule that)
  » In fact, feel free to run it twice on different nodes
Spark Tools

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

Apache Spark
## Spark and Map Reduce Differences

<table>
<thead>
<tr>
<th></th>
<th>Hadoop Map Reduce</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage</strong></td>
<td>Disk only</td>
<td>In-memory or on disk</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>Map and Reduce</td>
<td>Map, Reduce, Join, Sample, etc…</td>
</tr>
<tr>
<td><strong>Execution model</strong></td>
<td>Batch</td>
<td>Batch, interactive, streaming</td>
</tr>
<tr>
<td><strong>Programming</strong></td>
<td>Java</td>
<td>Scala, Java, R, and Python</td>
</tr>
</tbody>
</table>
Other Spark and Map Reduce Differences

- Generalized patterns
  - unified engine for many use cases
- Lazy evaluation of the lineage graph
  - reduces wait states, better pipelining
- Lower overhead for starting jobs
- Less expensive shuffles
In-Memory Can Make a Big Difference

- Two iterative Machine Learning algorithms:

**K-means Clustering**

- Hadoop MR: 121 sec
- Spark: 4.1 sec

**Logistic Regression**

- Hadoop MR: 80 sec
- Spark: 0.96 sec
First Public Cloud Petabyte Sort

<table>
<thead>
<tr>
<th></th>
<th>Hadoop MR Record</th>
<th>Spark Record</th>
<th>Spark 1 PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size</td>
<td>102.5 TB</td>
<td>100 TB</td>
<td>1000 TB</td>
</tr>
<tr>
<td>Elapsed Time</td>
<td>72 mins</td>
<td>23 mins</td>
<td>234 mins</td>
</tr>
<tr>
<td># Nodes</td>
<td>2100</td>
<td>206</td>
<td>190</td>
</tr>
<tr>
<td># Cores</td>
<td>50400 physical</td>
<td>6592 virtualized</td>
<td>6080 virtualized</td>
</tr>
<tr>
<td>Cluster disk throughput</td>
<td>3150 GB/s (est.)</td>
<td>618 GB/s</td>
<td>570 GB/s</td>
</tr>
<tr>
<td>Sort Benchmark Daytona Rules</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Network</td>
<td>dedicated data center, 10Gbps</td>
<td>virtualized (EC2) 10Gbps network</td>
<td>virtualized (EC2) 10Gbps network</td>
</tr>
<tr>
<td>Sort rate</td>
<td>1.42 TB/min</td>
<td>4.27 TB/min</td>
<td>4.27 TB/min</td>
</tr>
<tr>
<td>Sort rate/node</td>
<td>0.67 GB/min</td>
<td>20.7 GB/min</td>
<td>22.5 GB/min</td>
</tr>
</tbody>
</table>

Daytona Gray 100 TB sort benchmark record (tied for 1st place)

Spark Expertise Tops Big Data Median Salaries

Over 800 respondents across 53 countries and 41 U.S. states

History Review

- 2002: MapReduce @ Google
- 2004: MapReduce paper
- 2006: Hadoop Summit
- 2006: Hadoop @ Yahoo!
- 2008: Hadoop Summit
- 2010: Spark paper
- 2014: Apache Spark top-level
Historical References

• circa 1979 – Stanford, MIT, CMU, etc.: set/list operations in LISP, Prolog, etc., for parallel processing
  http://www-formal.stanford.edu/jmc/history/lisp/lisp.htm

• circa 2004 – Google: MapReduce: Simplified Data Processing on Large Clusters
  Jeffrey Dean and Sanjay Ghemawat
  http://research.google.com/archive/mapreduce.html

• circa 2006 – Apache Hadoop, originating from the Yahoo!’s Nutch Project
  Doug Cutting

• circa 2008 – Yahoo!: web scale search indexing
  Hadoop Summit, HUG, etc.
  http://developer.yahoo.com/hadoop/

• circa 2009 – Amazon AWS: Elastic MapReduce
  Hadoop modified for EC2/S3, plus support for Hive, Pig, Cascading, etc.
  http://aws.amazon.com/elasticmapreduce/
Spark Research Papers

- *Spark: Cluster Computing with Working Sets*
  Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
  USENIX HotCloud (2010)

- *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*
  Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
  NSDI (2012)