Cloud, Big Data & Linear Algebra

Shelly Garion

IBM Research -- Haifa
What is Big Data?
What is Big Data?

Global Data Volume in Exabytes

- 2005
- 2012
- 2017

- Sensors (Internet of Things)
- Social Media (video, audio, and text)
- VoIP
- Enterprise Data
What is Big Data?

![Diagram showing the increase in global data volume from 2005 to 2017](image)

- **Volume**
- **Velocity**
- **Variety**
- **Veracity**
- **Value**

*Global Data Volume in Exabytes*

- 2005
- 2012
- 2017
Big Data in the Cloud
How to Analyze Big Data?

KEEP CALM AND ANALYZE BIG DATA
How to Analyze Big Data?
Basic Example: Word Count (Spark & Python)

```python
>lines = sc.textFile("hamlet.txt")
>counts = lines.flatMap(lambda line: line.split(" "))
.map(lambda word => (word, 1))
.reduceByKey(lambda x, y: x + y)
```

Basic Example: Word Count (Spark & Scala)

```scala
> val lines = sc.textFile("hamlet.txt")
> val counts = lines.flatMap(_.split(" "))
    .map((_, 1))
    .reduceByKey(_ + _)  

"to be or"  "to"  "be"  "or"  
           |     |    |     |
          (to, 1) (be, 1) (or, 1) (be, 2) (not, 1)

"not to be"  "not"  "to"  "be"  
            |      |    |      |
           (not, 1) (to, 1) (be, 1) (or, 1) (to, 2)
```

Some History…

- **Map/Reduce** was invented by **Google**:  
  - Inspired by functional programming languages map and reduce functions  
  - Seminal paper: Jeffrey Dean and Sanjay Ghemawat (OSDI 2004), "MapReduce: Simplified Data Processing on Large Clusters"  
  - Used at Google to completely regenerate Google’s index of the World Wide Web

- **Hadoop** – open source implementation matches Google’s specifications

- **Amazon EMR** (Elastic MapReduce) running on Amazon EC2

- **Spark** started in 2009 as a research project of UC Berkley

- **Spark** is now an open source Apache project  
  - Built by a wide set of developers from over 200 companies  
  - more than 1000 developers have contributed to Spark  
  - IBM created Spark Technology Center (STC) - [http://www.spark.tc/](http://www.spark.tc/)
Why Spark?

- **Apache Spark™** is a fast and general open-source cluster computing engine for big data processing
- **Speed**: Spark is capable to run programs up to 100x faster than Hadoop Map/Reduce in memory, or 10x faster on disk
- **Ease of use**: Write applications quickly in Java, Scala, Python and R, also with notebooks
- **Generality**: Combine streaming, SQL and complex analytics – machine learning, graph processing
- **Runs everywhere**: on Apache Mesos, Hadoop YARN cluster manager, standalone, or in the cloud, and can read any existing Hadoop data, and data from HDFS, object store, databases etc.

https://spark.apache.org/
Combined Analytics of Data with Spark

Analyze tabular data with SQL
Analyze graph data using GraphX graph analytics engine
Use same machine learning Infrastructure
Use same solution for streaming data

Joseph Gonzalez, Reynold Xin, Ankur Dave, Daniel Crankshaw, Michael Franklin, and Ion Stoica, “GRAPHX: UNIFIED GRAPH ANALYTICS ON SPARK”, spark summit July 2014
Spark Example

Goal:
Find number of distinct names per "first letter".

| AHIR | Pat | Andy |
---|---|---|

Goal:
Find number of distinct names per "first letter".

res0 = [(A, 2), (P, 1)]
Spark Example

Goal: Find number of distinct names per “first letter”

```
sc.textFile("hdfs:/names")
 .map(name => (name.charAt(0), name))
 .groupByKey()
 .mapValues(names => names.toSet.size)
 .collect()
```

```
res0 = [(A, 2), (P, 1)]
```
PageRank Example
\[ A = \begin{bmatrix} 0 & 0.5 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} \]

\[ V = [1] \]

\[ B = 0.85 \cdot A \]

\[ U = 0.15 \cdot V \]

\[ B \cdot V + U = ? \]
\[
A = \begin{bmatrix}
0 & 0.5 & 1 & 0.5 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 \\
0 & 0.5 & 0 & 0
\end{bmatrix}
\]

\[
V = [1] \\
[1] \\
[1] \\
[1]
\]

\[
B = 0.85 \cdot A \\
U = 0.15 \cdot V
\]

\[
B \cdot V + U = \begin{bmatrix}
1.85 \\
1.0 \\
0.575 \\
0.575
\end{bmatrix}
\]
PageRank Example

\[
\begin{bmatrix}
0 & 0.5 & 1 & 0.5 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 \\
0 & 0.5 & 0 & 0 \\
\end{bmatrix}
\]

\[
V = [1] \\
B = 0.85 \times A \\
U = 0.15 \times V
\]

\[
B \times (B \times V + U) + U = ?
\]
PageRank Example

\[
A = \begin{bmatrix}
0 & 0.5 & 1 & 0.5 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 \\
0 & 0.5 & 0 & 0 \\
\end{bmatrix}
\]

\[
V = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\]

\[
B = 0.85A
\]

\[
U = 0.15V
\]

\[
B^* (B^* V + U) + U = \begin{bmatrix}
1.31 \\
1.72 \\
0.39 \\
0.58 \\
\end{bmatrix}
\]

\[
B^* (B^* (B^* V + U) + U) + U = \ldots
\]
\[
A = \begin{bmatrix}
0 & 0.5 & 1 & 0.5 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 \\
0 & 0.5 & 0 & 0 \\
\end{bmatrix}
\]

\[
V = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\]

\[
B = 0.85 \cdot A \\
U = 0.15 \cdot V \\
\]

At the \( k \)-th step:

\[
B^k V + (B^{k-1} + B^{k-2} + \ldots + B^2 + B + I)U = B^k V + (I - B^k)(I - B)^{-1}U
\]

For \( k = 10 \):

\[
\begin{bmatrix}
1.43 \\
1.37 \\
0.46 \\
0.73 \\
\end{bmatrix}
\]
\[ A = \begin{bmatrix} 0 & 0.5 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} \]

\[ V = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \]

\[ B = 0.85 \times A \]

\[ U = 0.15 \times V \]

Where \( k \) goes to infinity:

\[ B^k_v \rightarrow 0 \]

\[ B^k_v + (I-B^k)(I-B)^{-1}U \rightarrow (I-B)^{-1}U \]
A is a stochastic matrix,

\[
A = \begin{bmatrix}
0 & 0.5 & 1 & 0.5 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 \\
0 & 0.5 & 0 & 0
\end{bmatrix}
\]

\[
B = 0.85 \times A
\]

- Characteristic polynomial of A:
  \[x^4 - 0.5x^3 - 0.25x - 0.25\]
- A is a stochastic matrix,
- 1 is the largest eigen value of A (in its absolute value),
- 1 corresponds to the eigen vector:
  \[
E = \begin{bmatrix}
1.0 \\
1.0 \\
0.25 \\
0.25
\end{bmatrix}
\]

Where k goes to infinity:

\[A^k v \rightarrow cE\]

\[B^k v \rightarrow 0\]
PageRank

PageRank Algorithm

- Start each page with a rank of 1
- On each iteration:
  
  A. \( contrib = \frac{cur\text{Rank}}{|\text{neighbors}|} \)

  B. \( cur\text{Rank} = 0.15 + 0.85 \sum \text{contrib}_i \)

PageRank

- Rank of each page is the probability of landing on that page for a random surfer on the web

- Probability of visiting all pages after $k$ steps is

$$V_k = A^k \times V^t$$

$V$: the initial rank vector

$A$: the link structure (sparse matrix)

- Each page is identified by its unique URL rather than an index

- Ranks vectors ($V$): RDD[(URL, Double)]

- Links matrix ($A$): RDD[(URL, List(URL))]
PageRank in Spark

```scala
val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
    .mapValues(0.15 + 0.85 * _)
}

ranks.saveAsTextFile(...)
```

// Load the edges as a graph
val graph = GraphLoader.edgeListFile(sc, "graphx/data/followers.txt")
// Run PageRank
val ranks = graph$pageRank(0.0001).vertices
Machine Learning: K-Means Clustering

Goal:
Segment tweets into clusters by geolocation using Spark MLLib K-means clustering

Machine Learning: K-Means Clustering

Machine Learning: K-Means Clustering

1. \( k \) initial "means" (in this case \( k=3 \)) are randomly generated within the data domain (shown in color).

2. \( k \) clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the \( k \) clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

(from Wikipedia)
K-Means Clustering with Spark MLLib

To run the k-means algorithm in Spark, we need to first read the csv file:

```scala
val sc = new SparkContext("local[4]", "kmeans")
// Load and parse the data, we only extract the latitude and longitude of each line
val data = sc.textFile(arg)
val parsedData = data.map {
  line =>
    Vectors.dense(line.split(',').slice(0, 2).map(_.toDouble))
}
```

Then we can run the spark kmeans algorithm:

```scala
val iterationCount = 100
val clusterCount = 10
val model = KMeans.train(parsedData, clusterCount, iterationCount)
```

From the model we can get the cluster centers and group the tweets by cluster:

```scala
val clusterCenters = model.clusterCenters map (_.toArray)
val cost = model.computeCost(parsedData)
print("Cost: " + cost)
val tweetsByGoup = data
  .map (_.split(',').slice(0, 2).map(_.toDouble))
  .groupBy(rdd => model.predict(Vectors.dense(rdd)))
  .collect()
sc.stop()
```

Combined Analytics of Data with Spark

Analyze tabular data with SQL
Analyze graph data using GraphX graph analytics engine
Use same machine learning Infrastructure
Use same solution for streaming data

Joseph Gonzalez, Reynold Xin, Ankur Dave, Daniel Crankshaw, Michael Franklin, and Ion Stoica,
“GRAPHX: UNIFIED GRAPH ANALYTICS ON SPARK”, spark summit July 2014
How Does Spark Work?
Spark RDD (Resilient Distributed Dataset)

- Immutable, partitioned collections of objects spread across a cluster, stored in RAM or on Disk
- Built through lazy parallel transformations
- **Fault tolerance** – automatically built at failure

```scala
var myRDD = sc.sequenceFile("hdfs://…")
```

- We can apply **Transformations** or **Actions** on RDD
Spark Cluster

- **Driver program** – The process running the main() function of the application and creating the SparkContext
- **Cluster manager** – External service for acquiring resources on the cluster (e.g. standalone, Mesos, YARN)
- **Worker node** - Any node that can run application code in the cluster
- **Executor** – A process launched for an application on a worker node
Spark Scheduler

- **Task** - A unit of work that will be sent to one executor
- **Job** - A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action
- **Stage** - Each job gets divided into smaller sets of tasks called *stages* that depend on each other