Diving into Hadoop and Big Data
Michael Carducci
@MichaelCarducci
Agenda

• What is Big Data
• Introducing Hadoop
• Hadoop Ecosystem & Architecture
• HDFS Overview
• Programming Hadoop
  • MapReduce
  • Hive (SQL* OLAP)
  • Drill (Interactive Queries)
  • Pig (Fast Scripting)
• Spark
  • Spark SQL
  • Machine Learning
What is Big Data
What is Big Data

“90% of the data in the world today has been created in the last two years alone, at 2.5 quintillion bytes of data a day!”

–IBM Marketing Cloud, 2017
What is Big Data

• Large quantity of data
• Many sources/formats
• Unstructured
• Large Processing Load (needs to be distributed)
• Streaming/real-time processing
• Big Deployment footprint
Key Trends

- **Device Explosion**: 23.14 Billion Connected devices
- **Ubiquitous Connection**: 3.3 ZB by 2021, 278 EB/month
- **Social Networks**: 2.77 billion social media users by 2019
- **Sensor Networks**: Millions of new sensors go online every hour
- **Cheap Storage**: $0.019 Avg Cost of 1GB in 2016, 15.5 Million Times Cheaper
- **Inexpensive Computing**: $0.03 Cost/GFLOPS in 2018
Big Data
Big Opportunities
Letters in Winning Word of Scripps National Spelling Bee correlates with Number of people killed by venomous spiders

Correlation: 80.57% (r=0.8057)

Data sources: National Spelling Bee and Centers for Disease Control & Prevention

A New Way of Thinking

What, not Why
Big Data - Challenges

Volume

Variety

Velocity
“an open source software platform for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware”

–Hortonworks
Why Hadoop?

- Low Cost
- Linear Scaling
- Fault Tolerant
- Flexibility
Hadoop History

2002
Nutch created by Doug Cutting and Mike Caferella

2003-04
GFS and MapReduce From Google

2004
Nutch’s use of MapReduce and distributed filesystems take shape

2006
Doug Cutting joins Yahoo and brings Nutch with him

2008
Hadoop is born from Nutch

2008
Yahoo releases Hadoop as Open Source Project
We Learn by DOING

• Download Virtualbox
  https://www.virtualbox.org/

• Download Hortonworks sandbox
  https://hortonworks.com/products/sandbox/

• Download Example Data
  https://grouplens.org/datasets/movielens/

• Explore Examples offline!
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Zookeeper

- Tracks state across the cluster
  - Which node is master
  - What tasks are assigned to which workers
  - Which workers are current available?
- Failure Recovery
- HA witness
- Integral to many hadoop technologies
Failure Modes

• Master Crashes

• Worker crashes

• Network Issues
Zookeeper Under the Hood

• A distributed filesystem

• Supports basic general commands
  • Create, Delete, Exists, SetData, GetData, GetChildren

/  
/master  “master1.xxxx:2223”
  /workers
    /worker1  “worker1.xxxx:2225”
    /worker2  “worker2.xxxx:2225”
Push Notifications

• Clients can register for notifications on a znode
  • Avoids continuous polling
Persistent and Ephemeral Znodes

- Persistent zones persist until explicitly deleted
- Ephemeral Znodes go away when the client that created it disconnects.
Hadoop Ecosystem
Hadoop Ecosystem
Concepts

- Handle very large files
- Write Once
- Commodity Hardware
- Low Latency Data Access
- Small Files
- Arbitrary File Modifications
Architecture

Name Node

Data Nodes
Architecture

Master

Name Node

Data Nodes

Workers
Architecture

Master

Name Node

Data Nodes

Workers
Reading a file

Client → Name Node → Data Nodes
Writing a file

Client → Name Node → Data Nodes
Name Node Resiliency

- Backup Metadata
- Secondary Namenode*
- HDFS Federation
- HDFS High Availability

Name Node

SPOF
Using HDFS

- CLI
- HTTP/HDFS Proxies
- Java Client
- NFS Gateway
- UI (Ambari)
Let’s throw some data onto HDFS!
Hadoop Ecosystem
Apache YARN

- Introduced in Hadoop 2
- Manages Resources (Yet Another Resource Negotiator)
- Originally part of MapReduce
- Paved the way for more powerful/Performant MapReduce Alternatives
How YARN Works

• Application coordinates with YARN to distribute work across your cluster

• YARN attempts to process data on nodes that contain the relevant data blocks

• Scheduling options can be configured
  • FIFO, Capacity, Fair Schedulers
Hadoop Ecosystem

- **MapReduce** (Processing using different languages)
- **Hive & Drill** (Analytical SQL on Hadoop)
- **Mahout & Spark MLlib** (Machine learning)
- **PIG** (Scripting)
- **HBase** (NoSQL Database)
- **Zookeeper & Ambari** (Management & Coordination)

**SPARK** (In-Memory Data Flow Engine)

**Kafka & Storm** (Streaming)

**YARN**

**Storage**

**Resource Management**

**Flume**

Unstructured/Semi-structured Data

**Sqoop**

Structured Data

Apache Solr

Lucene

Apache Drill

Hive
MapReduce

- Distributes the processing of data on your cluster
- Resilient to failure
- Divides data into partitions
- Mappers transform data in parallel
-Reducers aggregate data together
How MapReduce Works: Mapping

• The Mapper transforms source rows into key/value pairs
Who rated the most jokes?
Example: Jester Data (jester_ratings.dat)

<table>
<thead>
<tr>
<th>UserId</th>
<th>JokeId</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>63978</td>
<td>147</td>
<td>8.281</td>
</tr>
<tr>
<td>63978</td>
<td>113</td>
<td>7.781</td>
</tr>
<tr>
<td>63978</td>
<td>112</td>
<td>9.438</td>
</tr>
<tr>
<td>63978</td>
<td>119</td>
<td>8.906</td>
</tr>
<tr>
<td>63978</td>
<td>121</td>
<td>8.438</td>
</tr>
<tr>
<td>63978</td>
<td>130</td>
<td>8.781</td>
</tr>
</tbody>
</table>
Example: Jester Data (jester_ratings.dat)

<table>
<thead>
<tr>
<th>UserId</th>
<th>JokeId</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>63978</td>
<td>147</td>
<td>8.281</td>
</tr>
<tr>
<td>63978</td>
<td>113</td>
<td>7.781</td>
</tr>
<tr>
<td>63978</td>
<td>112</td>
<td>9.438</td>
</tr>
<tr>
<td>63979</td>
<td>119</td>
<td>8.906</td>
</tr>
<tr>
<td>63979</td>
<td>121</td>
<td>8.438</td>
</tr>
<tr>
<td>63979</td>
<td>130</td>
<td>8.781</td>
</tr>
</tbody>
</table>

Mapper

Extract and organize the data we care about

MapReduce “Shuffle and Sort”


63978:147, 113, 112    63979:119, 121, 130

63978:147, 113, 112    63979:119, 121, 130
The REDUCER Processes Each Key’s Values

63978:147,113,112  63979:119,121,130

\[ \text{Len(jokes)} \]

\[ 63978:3 \quad 63979:3 \]
# The Big Picture

<table>
<thead>
<tr>
<th>UserId</th>
<th>JokeId</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>63978</td>
<td>147</td>
<td>8.281</td>
</tr>
<tr>
<td>63978</td>
<td>113</td>
<td>7.781</td>
</tr>
<tr>
<td>63978</td>
<td>112</td>
<td>9.438</td>
</tr>
<tr>
<td>63979</td>
<td>119</td>
<td>8.906</td>
</tr>
<tr>
<td>63979</td>
<td>121</td>
<td>8.438</td>
</tr>
<tr>
<td>63979</td>
<td>130</td>
<td>8.781</td>
</tr>
</tbody>
</table>

Mapper


Shuffle and Sort

63978:147,113,112, 63979:119,121,130

Reducer

63978:3, 63979:3
How MapReduce Scales
The Big Picture

<table>
<thead>
<tr>
<th>UserId</th>
<th>JokeId</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>978</td>
<td>147</td>
<td>8.281</td>
</tr>
<tr>
<td>976</td>
<td>113</td>
<td>7.781</td>
</tr>
<tr>
<td>978</td>
<td>112</td>
<td>9.438</td>
</tr>
<tr>
<td>979</td>
<td>119</td>
<td>8.906</td>
</tr>
<tr>
<td>974</td>
<td>121</td>
<td>8.438</td>
</tr>
<tr>
<td>978</td>
<td>130</td>
<td>8.781</td>
</tr>
</tbody>
</table>

Mapper

Shuffle and Sort

Reducer
Coordinating Distributed Map/Reduce Tasks

- Client Node
- YARN
- MapReduce Application Master
- NodeManager Node
- HDFS
  - MapTask / ReduceTask
  - MapTask / ReduceTask
  - MapTask / ReduceTask
- NodeManager Node
How Mappers & Reducers are Written

- MapReduce is Java Native
- STREAMING allows interfacing to other languages
Handling Failure

- Application Master monitors worker tasks
  - Restarts as needed
  - Reassigns to a different node
- YARN Monitors Application Master
  - Restarts as needed
- YARN Monitors Nodes
  - Restarts as needed
- In HA configuration, Zookeeper monitors YARN and switches to hot standby if needed
MapReduce in Action:
Which is the most rated joke?
from mrjob.job import MRJob
from mrjob.step import MRStep
class JokeRatings(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
            reducer=self.reducer_count_ratings),
            MRStep(mapper=self.mapper_get_ratings,
            reducer=self.reducer_count_ratings)
        ]
def mapper_get_ratings(self, _, line):
    (userID, JokeID, rating) = line.split('|')
    yield JokeID, 1

def reducer_count_ratings(self, key, values):
    yield sum(values), key
from mrjob.job import MRJob
from mrjob.step import MRStep

class JokeRatings(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, JokeID, rating) = line.split('|')
        yield JokeID, 1

    def reducer_count_ratings(self, key, values):
        yield sum(values), key

if __name__ == '__main__':
    JokeRatings.run()
Hadoop Ecosystem
Introducing Hive

- Abstracts away from Map/Reduce
- Use SQL Syntax
- Interactive
- Scalable (query across cluster)
- Extensible
- Highly Optimized
- Support JDBC/ODBC
- Leverage DB/OLAP Skillset
Introducing Hive

- Abstracts away from Map/Reduce
- Use SQL Syntax
- Interactive
- Scalable (query across cluster)
- Extensible
- Highly Optimized
- Support JDBC/ODBC
- Leverage DB/OLAP Skillset
Write SQL*, Get MapReduce
SQL* == HiveQL

• Basically MySQL

  • Specify data structure and partitions

  • Chain views together and use as tables
Using HIVE

• CLI
• Query Files
• Oozie
• Web/UI (Ambari)
• JDBC/ODBC Server
• Thrift Service
Most Rated Joke (in Hive)
from mrjob.job import MRJob
from mrjob.step import MRStep

class JokeRatings(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, JokeID, rating) = line.split('|')
        yield JokeID, 1

    def reducer_count_ratings(self, key, values):
        yield sum(values), key

if __name__ == '__main__':
    JokeRatings.run()
SELECT joke_id, count(*) as ratingCount
FROM jester_ratings
GROUP by rating;
Schema on Read vs Schema on Write
CREATE TABLE JokeRatings(
    UserID INT,
    JokeID INT,
    Rating INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘|’
STORED AS TEXTFILE

LOAD DATA LOCAL INPATH ‘$(env:HOME)/jester/jester_ratings.dat’
OVERWRITE INTO TABLE JokeRatings

Schema on Read
Storing Data

• LOAD DATA
  • MOVES data from HDFS into Hive

• LOAD DATA LOCAL
  • COPIES data from local filesystem into HIVE

• Managed vs External
Managed vs External Tables

- Data MOVED into Hive is Managed by hive
- Create External Table leaves data available in HDFS
Partitioning

• Store data in partitioned subdirectories

CREATE TABLE JokeRatings(
    UserID INT,
    JokeID INT,
    Rating INT
)
PARTITIONED BY UserID
Introducing Drill

- External SQLQuery Engine
- Unify/query across multiple data sources
  - HDFS
  - MongoDB
  - S3
  - Azure Blob Storage
  - Hive
  - HBase
  - etc…
Introducing Drill

- External SQLQuery Engine
- Unify/query across multiple data sources
  - HDFS
  - MongoDB
  - S3
  - Azure Blob Storage
  - Hive
  - HBase
  - etc…
Introducing Drill

• External SQLQuery Engine

• Unify/query across multiple data sources
  • HDFS
  • MongoDB
  • S3
  • Azure Blob Storage
  • Hive
  • HBase
  • etc…
Hadoop Ecosystem
Introducing Apache Pig
Introducing Apache Pig

• Abstracts Map/Reduce Complexity

• Uses PIG LATIN scripting language

• Highly extensible with UDFs

• Can perform faster than MapReduce
Running Pig Scripts

- Grunt
- Script
- Browser (Ambari)
Pig Latin Crash Course
movieRatings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userId: int, movieId:int, rating:int, ratingTime:int)
Working With Relations

- LOAD, STORE, DUMP
- FILTER, DISTINCT, FOREACH/GENERATE, MAPREDUCE, STREAM, SAMPLE
- JOIN, COGROUP, GROUP, CROSS, CUBE
- ORDER, RANK, LIMIT
- UNION, SPLIT
Working With Relations

• LOAD, STORE, DUMP

• FILTER, DISTINCT, FOREACH/GENERATE, MAPREDUCE, STREAM, SAMPLE

• JOIN, COGROUP, GROUP, CROSS, CUBE

• ORDER, RANK, LIMIT

• UNION, SPLIT
Working With Relations

• LOAD, STORE, DUMP

• FILTER, DISTINCT, FOREACH/GENERATE, MAPREDUCE, STREAM, SAMPLE

• JOIN, COGROUP, GROUP, CROSS, CUBE

• ORDER, RANK, LIMIT

• UNION, SPLIT
Diagnostics

• DESCRIBE

• EXPLAIN

• ILLUSTRATE
UDFs

- REGISTER
- DEFINE
- IMPORT
Aggregate Functions

- AVG
- COUNT
- CONCAT
- MAX
- MIN
- SIZE
- SUM
Loaders

• PigStorage
• TextLoader
• JsonLoader
• HBaseStorage
Hands on Example: Mining Movie Rating Data
LOAD ratings data

movieRatings = LOAD ‘/user/maria_dev/ml-100k/u.data’ AS (userId: int, movieId:int, rating:int, ratingTime:int)
Specify non-default delimiter: PigStorage

movieData = LOAD 'user/maria_dev/ml-100k/u.item'
  PigStorage('|')
  AS (MovieId: int, movieTitle:chararray,
      releaseDate:chararray, videoRelease:chararray,
      imdbLink:chararray);
Create a new relation from another relation

```java
movieData = LOAD '/user/maria_dev/ml-100k/u.item'
    PigStorage('|')
    AS (MovieId: int, movieTitle: chararray,
        releaseDate: chararray, videoRelease: chararray,
        imdbLink: chararray);

nameLookup = FOREACH movieData GENERATE movieId,
    movieTitle, ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) As releaseTime
```
ratingsByMovie = GROUP movieRatings BY movieId;
Compute avgRatings

avgRatings = FOREACH ratingsByMovie GENERATE group AS movieId, AVG(ratings.rating) as avgRating;
Filtering Results

\[ \text{fiveStarMovies} = \text{FILTER avgRatings BY avgRating} > 4.0 \]
JOIN

fiveStarWithNames = JOIN fiveStarMovies BY movieID, nameLookup BY movieId;
oldestFiveStarMovies = ORDER fiveStarsWithData BY nameLookup::releaseTime;

DUMP oldestFiveStarMovies;
Let’s Run It
Hadoop Ecosystem
Introducing HBASE

• NoSQL DB build on HDFS

• Based on BigTable

• No Language - API

• Auto Sharding
HBase Data Model

- Fast access to any given row
- A row is referenced by a unique key
- Rows has a small number of **column families**
  - A column family may contain arbitrary columns
  - You can have a very large number of columns in a column family
- Each Cell can have many versions with given timestamps
- Sparse data is ok. Missing columns in a row consume no storage
Example: Web Table

Key
com.cnn.www

Contents Column Family
Contents:
- com.cnn.www
- Anchor Column Family
  Anchor cnnsi.com:
  - “CNN”
  Anchor my.look.ca:
  - “CNN.com”
Accessing HBase

- HBase shell
- API
  - Wrappers in many language
- Spark, Hive, Pig
- Rest Service
- Thrift Service
- Avro Service
Hadoop Ecosystem
Introducing Apache Spark

“A fast and general engine for large-scale data processing”
Spark Components

- Spark Streaming
- Spark SQL
- MLLib
- GraphX
Spark is Scalable

Driver Program - Spark Context

Cluster Manager (Spark, YARN)

Executor - Cache
- Tasks

Driver Program - Spark Context
Spark is Fast

- “Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.”

- Directed Acyclic Graph Engine optimizes workflows
Spark is Popular

- Amazon
- Yahoo
- Groupon
- TripAdvisor
- NASA JPL
- Ebay

*https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark*
Easy to Code

- Code in Java, Scala, Python
- Built around one primary concept: Resilient Distributed Dataset (RDD)
What is an RDD?
SparkContext

- Created by your driver program
- SparkContext Makes your RDDs Resilient and Distributed
- It creates the RDDs
- The spark shell creates an “sc” object for you
Creating RDDs

- `rdd = parallelize([1,2,3,4])`
- `sc.textFile("hdfs:///c:/users/michael/bigTextFile.txt")` or `file:// s3n:// etc`
- `hiveCtx = HiveContext(sc) rows = hiveCtx.sql("SELECT name, age FROM users")`
- Create from:
  - JDBC
  - Cassandra
  - HBase
  - JSON
  - CSV
  - etc…
Transforming RDDs

• map

• flatmap

• filter

• distinct

• sample

• union, intersection, subtract, cartesian
map Example

- rdd = sc.parallelize([1,2,3,4])

- squaredRDD = rdd.map(lambda x: x*x)

Yields 1,4,9,16
RDD actions

• collect
• count
• countByValue
• take
• top
• Reduce
• ...

…
RDD actions

- collect
- count
- countByValue
- take
- top
- Reduce
- ...
- ...
Example - Find the lowest average rating.
from pyspark import SparkConf, SparkContext

def loadMovieNames():
    movieNames = {}
    with open("ml-100k/u.item") as f:
        for line in f:
            fields = line.split('\\|\\')
            movieNames[int(fields[0])] = fields[1]
    return movieNames
def parseInput(line):
    fields = line.split()
    return (int(fields[1]), (float(fields[2]), 1.0))

if __name__ == '__main__':
    # The main script - create our SparkContext
    conf = SparkConf().setAppName("WorstMovies")
    sc = SparkContext(conf = conf)

    # Load up our movie ID -> movie name lookup table
    movieNames = loadMovieNames()

    # Load up the raw u.data file
    lines = sc.textFile("hdfs:///user/maria_dev/ml-100k/u.data")
# Convert to (movieID, (rating, 1.0))
   movieRatings = lines.map(parseInput)

# Reduce to (movieID, (sumOfRatings, totalRatings))
   ratingTotalsAndCount = movieRatings.reduceByKey(lambda movie1, movie2: (movie1[0] + movie2[0], movie1[1] + movie2[1]))

# Map to (rating, averageRating)
   averageRatings = ratingTotalsAndCount.mapValues(lambda totalAndCount: totalAndCount[0] / totalAndCount[1])
# Sort by average rating
    sortedMovies = averageRatings.sortBy(lambda x: x[1])

# Take the top 10 results
    results = sortedMovies.take(10)

# Print them out:
    for result in results:
        print(movieNames[result[0]], result[1])
DataFrames & DataSets

- Extend RDDs
- Contain Row Objects
- Can run SQL Queries
- Has a Schema
- Read/Write to JSON, HIVE, Parquet
- Supports JDBC/ODBC, Tableau
- Query using SparkSQL
Working with DataFrames without SparkSQL

- `myDataFrame.show()`
- `myDataFrame.select("fieldName")`
- `myDataFrame.filter(myDataFrame("count") > 52)`
- `myDataFrame.groupBy(myDataFrame("quantity")).mean()`
- `myDataFrame.rdd().map(mapperFunction)`
Find Lowest Avg Movie using Datasets
from pyspark import SparkConf, SparkContext

def loadMovieNames():
    movieNames = {}
    with open("ml-100k/u.item") as f:
        for line in f:
            fields = line.split(' | ')
            movieNames[int(fields[0])] = fields[1]
    return movieNames
def parseInput(line):
    fields = line.split()
    return (int(fields[1]),
            (float(fields[2]), 1.0))

if __name__ == '__main__':
    # The main script - create our SparkContext
    conf = SparkConf().setAppName("WorstMovies")
    sc = SparkContext(conf = conf)

    # Load up our movie ID -> movie name lookup table
    movieNames = loadMovieNames()
# Get the raw data
lines =
spark.sparkContext.textFile("hdfs://
/user/maria_dev/ml-100k/u.data")
    # Convert it to a RDD of Row
objects with (movieID, rating)
movies = lines.map(parseInput)
    # Convert that to a DataFrame
movieDataset =
spark.createDataFrame(movies)
# Compute average rating for each movieID
averageRatings =
movieDataset.groupBy("movieID").avg("rating")

# Compute count of ratings for each movieID
counts =
movieDataset.groupBy("movieID").count()

# Join the two together (We now have movieID, avg(rating), and count columns)
averagesAndCounts =
counts.join(averageRatings, "movieID")

# Filter movies rated 10 or fewer times
popularAveragesAndCounts =
averagesAndCounts.filter("count > 10")
# Pull the top 10 results

topTen =
popularAveragesAndCounts.orderBy("avg(rating)").take(10)

# Print them out, converting movie ID's to names as we go.
for movie in topTen:
    print (movieNames[movie[0]], movie[1], movie[2])

# Stop the session
spark.stop()
Hadoop Ecosystem
What is Storm

- A framework for stream processing on your cluster
- Works on individual events
- sub-second latency
Storm Terminology

- A stream consists of tuples that flow through
- Spout are sources of stream data (kafka etc)
- Bolts process stream data as it’s received
  - transform aggregate, write to databases
- A topology is a group of spouts and bolts that process your stream
Developing Storm Applications

• Typically written in java

• Storm core
  • Low level API for storm
  • “At least once”

• Trident
  • Higher level API for storm
  • “Exactly Once”

• Storm runs “forever” once submitted… Until you stop them
Hadoop Ecosystem
Using MLLib in Spark
from pyspark import SparkConf, SparkContext

def loadMovieNames():
    movieNames = {}
    with open("ml-100k/u.item") as f:
        for line in f:
            fields = line.split('|')
            movieNames[int(fields[0])] = fields[1]
    return movieNames
def parseInput(line):
    fields = line.split()
    return (int(fields[1]),
            (float(fields[2]), 1.0))

if __name__ == "__main__":
    # The main script - create our SparkContext
    conf = SparkConf().setAppName("WorstMovies")
    sc = SparkContext(conf = conf)

    # Load up our movie ID -> movie name lookup table
    movieNames = loadMovieNames()
# Get the raw data
lines = spark.sparkContext.textFile("hdfs:///user/maria_dev/ml-100k/u.data")

# Convert it to a RDD of Row objects with (movieID, rating)
movies = lines.map(parseInput)

# Convert it to a RDD of Row objects with (userID, movieID, rating)
ratingsRDD = lines.map(parseInput)

# Convert to a DataFrame and cache it
ratings = spark.createDataFrame(ratingsRDD).cache()
# Construct a "test" dataframe for user 0 with every movie rated more than 100 times

```python
popularMovies =
    ratingCounts.select("movieID").withColumn('userID', lit(0))
```

# Run our model on that list of popular movies for user ID 0

```python
recommendations = model.transform(popularMovies)
```

# Get the top 20 movies with the highest predicted rating for this user

```python
topRecommendations =
    recommendations.sort(recommendations.prediction.desc()).take(20)
```

for recommendation in topRecommendations:
    print (movieNames[recommendation['movieID']], recommendation['prediction'])

spark.stop()
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
SQOOP
Sqoop

RDBMS

Mapper

Mapper

Mapper

Mapper

HDFS
Sqoop Example: Import data from MySQL

```
```
Sqoop Example: Import data from MySQL to Hive

```
```
Incremental Imports

• You can keep your RDBMS in sync

• —check-column and —last-value
Sqoop Example: Export data to MySQL from Hive

```
sqoop export --connect jdbc:mysql://localhost/movielens
--driver com.mysql.jdbc.Driver
--table exported_movies
--export-dir /apps/hive/warehouse/movies
--input-filesystem-terminated-by '\0001'
```
Thank you

Michael Carducci

@MichaelCarducci
GREAT INDIAN DEVELOPER SUMMIT 2019
Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com  @greatindiandev  bit.ly/gidslinkedin