Programming for Big Data

Lecture 3

Data Processing with Spark

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH

DUBLIN

TECHNOLOGICAL UNIVERSITY DUBLIN

Dr. Bojan Božić
Technological University Dublin

Agenda

Preliminaries

1. RECAP
   - Architectures
   - Programmings RDDs (Map/Reduce)
   - Machine Learning with Spark (Clustering)
2. Summary

PRELIMINARIES

Books and Resources

Mastering Spark

www.packtpub.com/big-data-and-business-intelligence/mastering-apache-spark

Apache Spark From Inception to Production

http://info.mapr.com/rs/mapr/images/Getting_Started_With_Apache_Spark.pdf

Spark Programming Guide

http://spark.apache.org/docs/latest/programming-guide.html

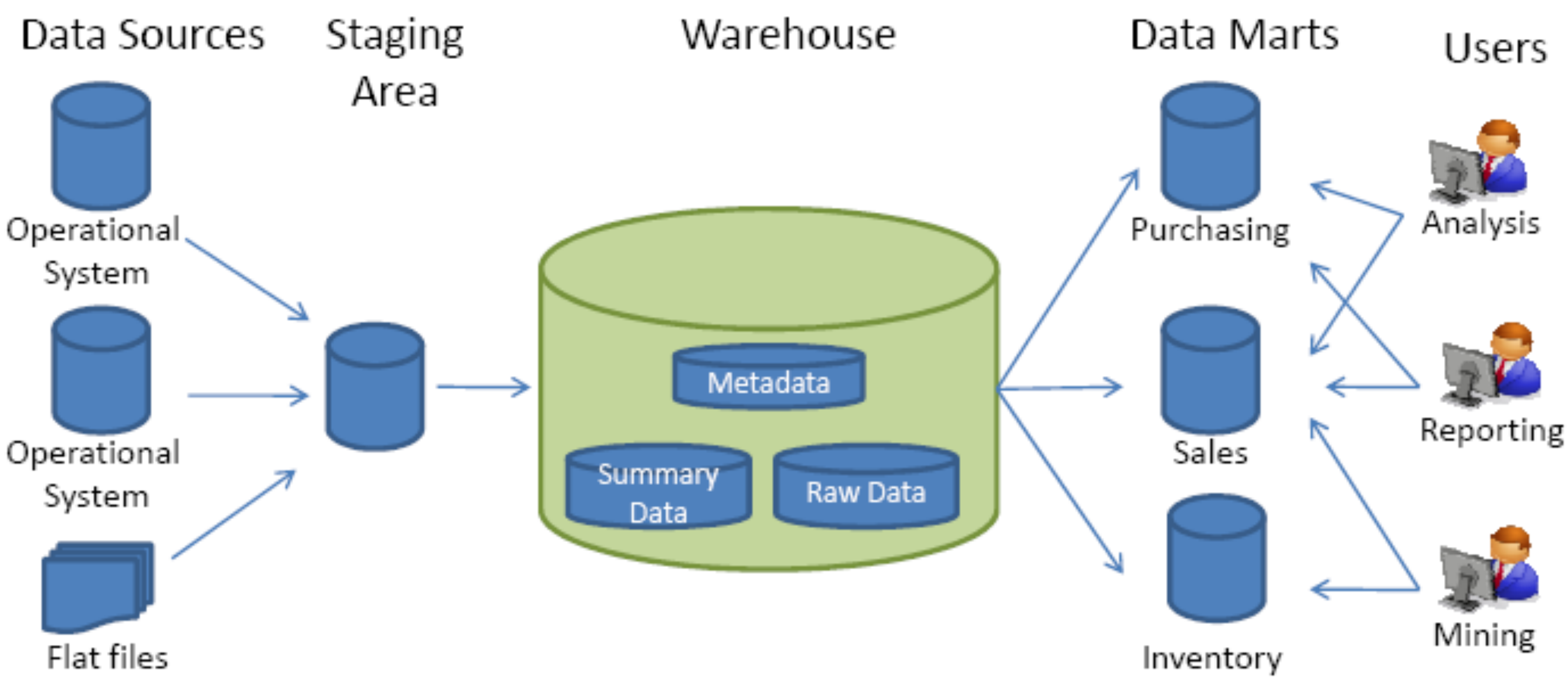
Software

http://spark.apache.org/

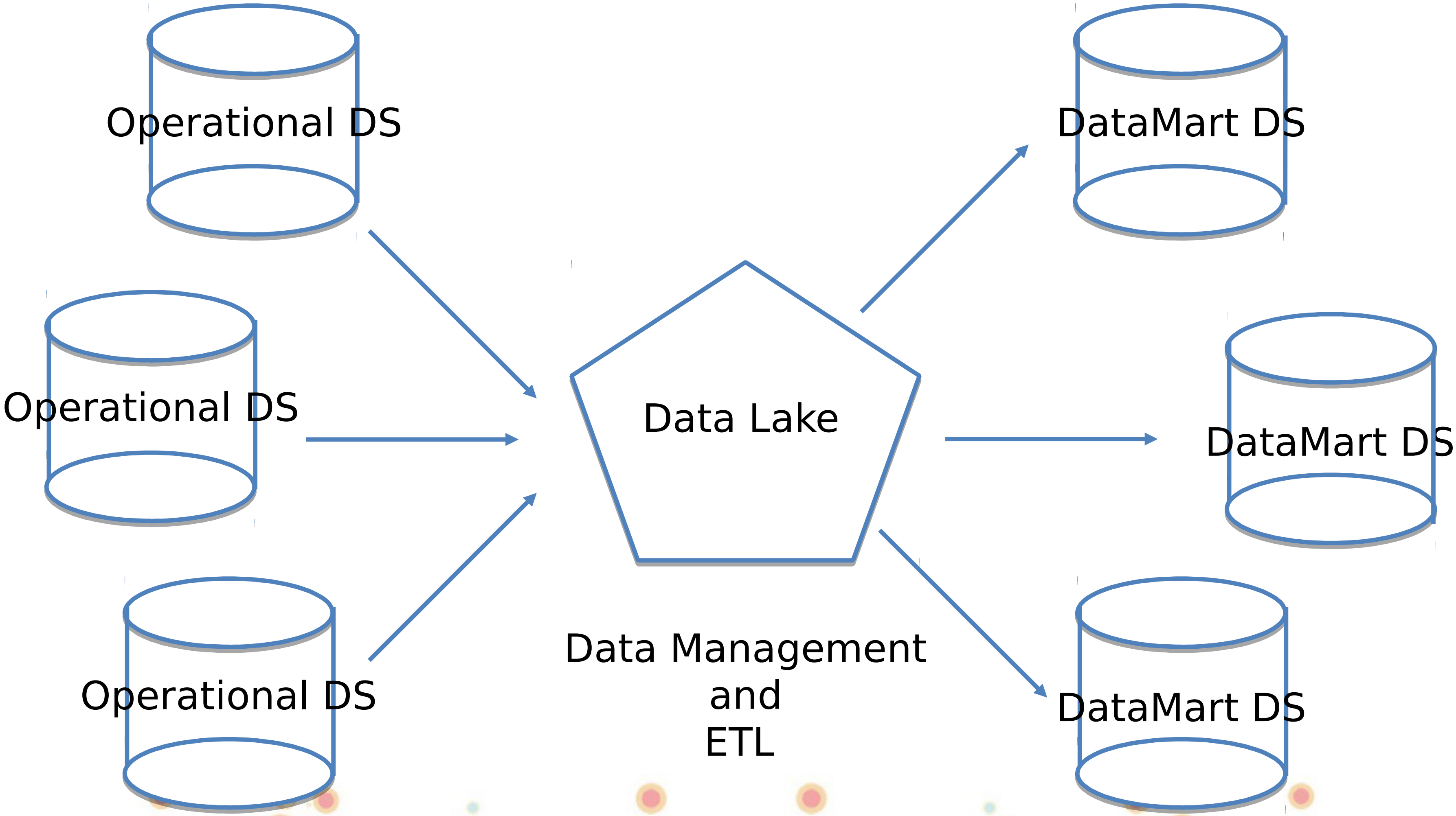
RECAP

ARCHITECTURES

Data Warehousing



Data Lakes

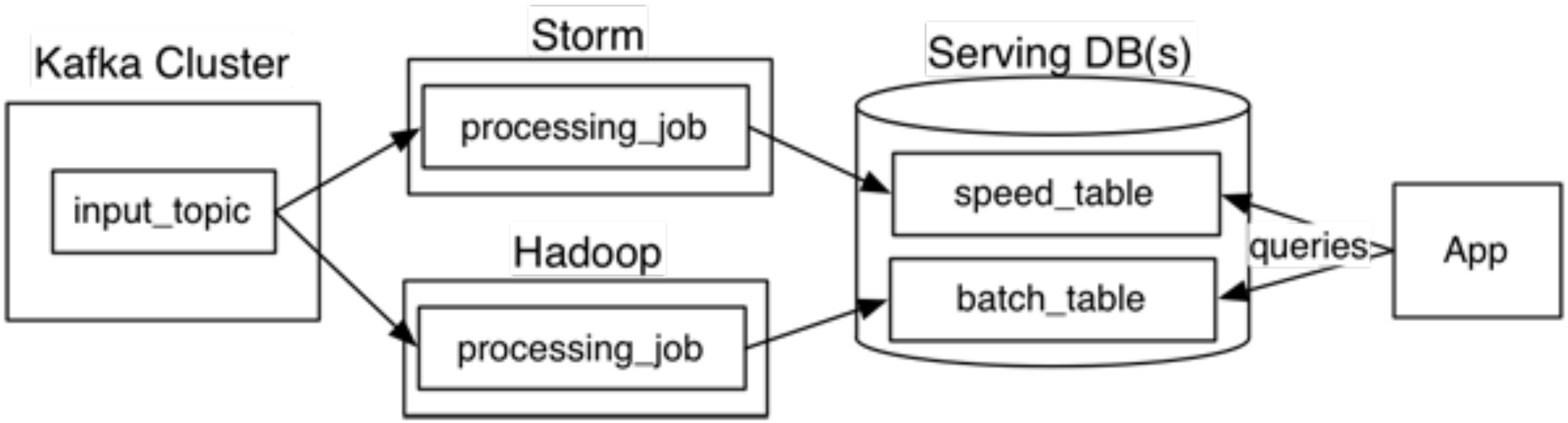


Operational DS
Operational DS
Operational DS
Data Lake
Data Management
and
ETL
DataMart DS
DataMart DS
DataMart DS

Data Lakes

1. ***Retain all Data*** *- don't throw away anything, everything is kept in the original form, no upstream transformation*
2. ***Support all data types*** *- Structured (csv, tsv, rdf, sql) and un-structured (text, numeric), no homogenisation required, use HDFS or Key-Value Store*
3. ***Supports all users*** *- data scientists, business users etc*
4. ***Easily adapt to change***
5. ***Provide faster insights*** *- Schema-on-read so there is no need to define complex ETL and homogenisation efforts*
6. ***Requires strict governance and data engineering efforts***
7. *Supports Batch Processing*

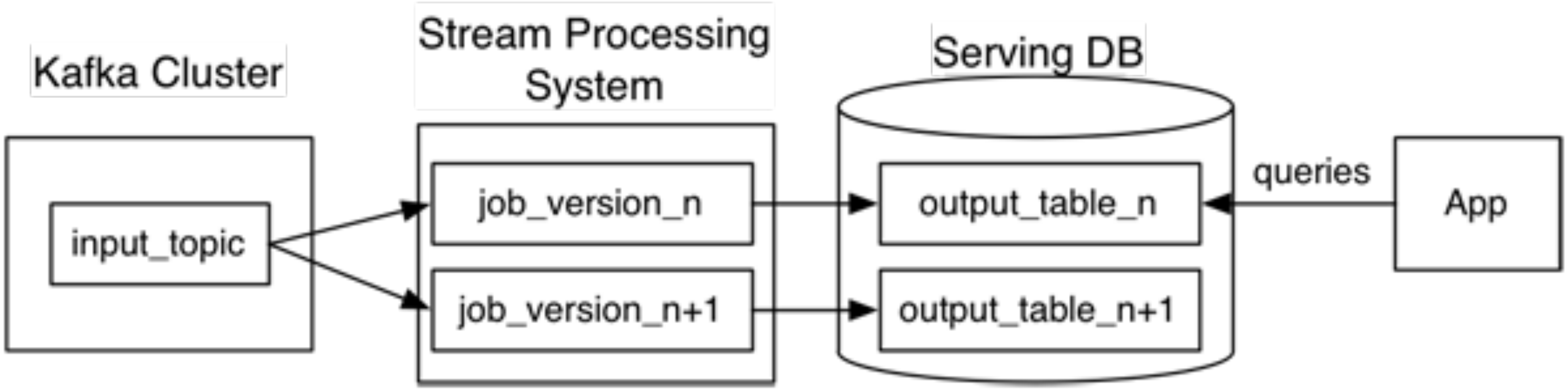
Lambda Architecture



Lambda Architecture

A *complex architecture that requires two code bases to be maintained, each applying functions independently on data. Given the stress of software production, each code base can fall out of step resulting in different functions being applied to the same data at different stages.*

Kappa Architecture



Big Data Architecture

Data Processing has developed in a way that requires new systems that support a variety of mechanisms to stream, manage, process and interrogate data.

The ecosystem was divided between a variety of Apache Incubator projects until the arrival of Spark - a general purpose data processing engine that has gained momentum in the last several years.

PROGRAMMINGS RDDS

Programmings RDDs

RDD operations are divided between:

1) **Transformations** result in a new RDD
2) **Actions** return a result (computation) to the driver

Programmings RDDs

Union(RDD) (join two RDDs):

__val__ badLinesRDD __=__ errorsRDD.union(warningsRDD)

Intersection(RDD) (return the common values of two RDDs):

```
val lines1 = sc.parallelize(List("hello", "world", "tennis", "Japan"))
val lines2 = sc.parallelize(List("hello", "Japan"))
val common = lines1.intersection(lines2)
common.collect().foreach(println)
```

Distinct (return distinct values from and RDD):

```
val lines = sc.parallelize(List(“hello”, “hello”,“tennis”, “Japan”))
lines.distinct().collect().foreach(println)
```

Subtract (take one RDD from another and return):

```
val common = lines1.subtract(lines2)
common.distinct().collect().foreach(println)
```

Programmings RDDs

Transformations (Key, Value Pairs):

ReduceByKey (combines values with the same key):

```
val data = Seq(("a", 3), ("b", 4), ("a", 1))
val result = sc.parallelize(data).reduceByKey((x, y) => x + y))
result.collect().foreach(println)
```

GroupByKey (returns a dataset of (K, Iterable<V>) pairs):

```
result.groupByKey.collect.foreach(println))
```

MapValues (apply to each value):

```
val result = sc.parallelize(Seq(("a", 3), ("b", 4), ("a", 1)))
result.mapValues(x => x+1).foreach(println)
```

keys (returns the keys from a key.value pair):

```
val result = sc.parallelize(Seq(("a", 3), ("b", 4), ("a", 1)))
result.keys.foreach(println)
```

Programmings RDDs

Transformations contd... (Key, Value Pairs):

Values (return an RDD of just the values):

```
val data = Seq(("a", 3), ("b", 4), ("a", 1))
data.values.collect().foreach(println)
```

SortByKey (returns a sorted by key RDD):

```
data.sortByKey().collect().foreach(println)
```

flatMapValues (returns an RDD of just the values):

```
data.sortByKey().collect().foreach(println)
```

subtractByKey (return an RDD of just the values):

```
data.sortByKey().collect().foreach(println)
```

Programmings RDDs

Actions:

Reduce (reduces two elements to one):

```
val input = sc.parallelize(List(1, 2, 3, 4))
val result = input.map(x => x * x)
val sumInput = input.reduce((x,y) => x + y)
val sumResult = result.reduce((x,y) => x + y)
```

Collect (returns all the elements of a dataset as an array):

```
val lines = sc.parallelize(List("hello", "hello","tennis", "Japan"))
lines.collect().foreach(println)
```

Count (take the first element):

```
val common = lines.first
```

Take (return an array of the first n elements):

```
val common = lines.take(10)
```

Programmings RDDs

foreach (apply a function to every element in an array):

```
val input = sc.parallelize(List(1, 2, 3, 4))
input.collect().foreach(println)
```

takeOrdered (takes an ordered set of elements from an array):

```
val input = sc.parallelize(List(1, 2, 3, 4))
input.collect().foreach(println)
```

Worked Example

LETS DO AN EXAMPLE TOGETHER

Population vs. Median Home Prices

Linear Regression with Single Variable

Worked Example

```
# Load and parse the data
# Use the Spark CSV datasource with options specifying:
#  - First line of file is a header
#  - Automatically infer the schema of the data
data = sqlContext.read.format("com.databricks.spark.csv")\
  .option("header","true")\
  .option("inferSchema","true")\
  .load("/databricks-datasets/samples/population-vs-price/data_geo.csv")
data.cache() # cache data for faster reuse
data.count()

display(data)

data = data.dropna() # drop rows with missing values
data.count()
```

Worked Example

```
# This will let us access the table from our SQL notebook!
data.createOrReplaceTempView("data_geo")

# Limit data to population vs price
df = spark.sql("select `2014 Population estimate`, `2015
median sales price` as label from data_geo")
display(df)

# Transform the output dataframe
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
assembler = VectorAssembler(inputCols=["2014
Population estimate"],outputCol="features")
outputdf = assembler.transform(df)

display(outputdf.select("features", "label"))
```

Worked Example

```
# Scatterplot of the date using ggplot
import numpy as np
import matplotlib.pyplot as plt

x = outputdf.rdd.map(lambda p: (p.features[0])).collect()
y = outputdf.rdd.map(lambda p: (p.label)).collect()

from pandas import *
from ggplot import *
pydf = DataFrame({'pop': x, 'price': y})
p = ggplot(pydf, aes('pop','price'))+\
  geom_point(color='blue')
display(p)
```

Worked Example

```
# Linear Regression
# Goal: predict y = 2015 Median Housing Price using feature x = 2014 Population
Estimate
from pyspark.ml.regression import LinearRegression # Import LinearRegression class
lr = LinearRegression() # Define LinearRegression algorithm
# Fit 2 models, using different regularization parameters
modelA = lr.fit(outputdf,{lr.regParam:0.0})
modelB = lr.fit(outputdf,{lr.regParam:100.0})

print ">>>> ModelA intercept: %r, coefficient: %r" % (modelA.intercept,
modelA.coefficients[0])
print ">>>> ModelB intercept: %r, coefficient: %r" % (modelB.intercept,
modelB.coefficients[0])

# Make predictions
# calling transform adds a new column of predictions
predictionsA = modelA.transform(outputdf)
display(predictionsA)
```

Worked Example

```
# Evaluate the model
# Predicted vs true label
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(metricName="rmse")
RMSE = evaluator.evaluate(predictionsA)
print("ModelA: Root Mean Squared Error = " + str(RMSE))

predictionsB = modelB.transform(outputdf)
RMSE = evaluator.evaluate(predictionsB)
print("ModelB: Root Mean Squared Error = " + str(RMSE))
```

Worked Example

```
# Linear regression plots
import numpy as np
from pandas import *
from ggplot import *

pop = outputdf.rdd.map(lambda p: (p.features[0])).collect()
price = outputdf.rdd.map(lambda p: (p.label)).collect()
predA = predictionsA.select("prediction").rdd.map(lambda r: r[0]).collect()
predB = predictionsB.select("prediction").rdd.map(lambda r: r[0]).collect()

pydf = DataFrame({'pop': pop, 'price': price, 'predA': predA, 'predB': predB})

# View the python pandas dataframe (pydf)
pydf

# ggplot figure (display the scatterplot and two regression models)
p = ggplot(pydf,aes('pop','price')) + geom_point(color='blue') + geom_line(pydf,
aes('pop','predA'),color='red') + geom_line(pydf,aes('pop','predB'),color='green') +
scale_x_log10() + scale_y_log10()
display(p)
```

MACHINE LEARNING

Machine Learning with Spark

- Basic Data Types
- Basic Statistics
- Supervised
  - Classification and Regression
- Unsupervised
  - Collaborative Filtering (Matrix Factorisation)
  - Clustering
  - Dimensionality Reduction
- Frequent Pattern Matching
- Evaluation
- PMML - Publishing and Sharing

Machine Learning with Spark

- Data Types
  - Local Vector
    - Vector on a single machine
    - Sparse and Dense Vector types
  - Labelled point
    - Vector associated with a label/response
  - Local Matrix
    - Dense and Sparse Matrices supported
  - Distributed Matrix
    - RowMatrix, IndexedRow Matrix, Coordinate Matrix

Machine Learning with Spark

- Data Statistics
  - Summary Statistics

val summary: MultivariateStatisticalSummary = Statistics.colStats(observations)
println(summary.mean)

  - Correlations

val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")

  - Stratified Sampling - splits automatically on a given level

val approxSample = data.sampleByKey(withReplacement = false, fractions)

  - Hypothesis Testing
  - Random data generation

val u = normalRDD(sc, 1000000L, 10)

Machine Learning with Spark

- Supervised
  - Classification (Binary and Multinomial)
    - Linear SVM
    - Logistic Regression
    - Decision trees (random forests)
  - Regression
    - Linear Regression (Linear least squares)
    - Lasso (Support regularisation and feature selection)
    - Ridge Regression (Addresses Co-linearity)
    - Streaming linear regression (A version of online learning)
    - Decision Tree Regression

Machine Learning with Spark

- Unsupervised
  - Collaborative Filtering (Matrix Factorisation)
    - Supports user-item recommendation
    - Supports Implicit and Explicit Feedback
  - Clustering
    - K-means
    - Gaussian Mixture Models
    - Latent Dirichlet allocation (Topic Modelling)
    - Bisecting k-means, Streaming K-means
  - Dimensionality Reduction
    - PCA
    - SVD

Machine Learning with Spark

- Evaluation
  - Classification
    - Confusion Matrix
    - Precision (Positive Predictive Value)
    - Recall (True Positive Rate)
    - F-measure
    - Receiver Operating Characteristic (ROC)
    - Area Under ROC Curve
  - Regression
    - Mean Squared Error (MSE)
    - Root Mean Squared Error & Mean Absolute Error
    - Coefficient of Determination (R2)

Summary

Preliminaries

1. RECAP
   - Architectures
   - Programmings RDDs (Map/Reduce)
   - Machine Learning with Spark (Clustering)
2. Summary

Further Reading

MLIB Release:
https://databricks.com/blog/2014/07/16/new-features-in-mllib-in-spark-1-0.html

Lasso Explained:
https://www.youtube.com/watch?v=qU1_cj4LfLY

Ridge Regression Explained:
http://www.ncss.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf

Netflix Prise using Matrix Factorisation:
http://dx.doi.org/10.1007/978-3-540-68880-8_32

Gaussian Mixture Models
https://www.youtube.com/watch?v=Rkl30Fr2S38

Questions

?