BENELEARN 2016

BELGIAN-DUTCH CONFERENCE ON MACHINE LEARNING

Big Data Learning in Practice

Isaac Triguero

School of Computer Science

University of Nottingham

United Kingdom

Isaac.Triguero@nottingham.ac.uk

http://www.cs.nott.ac.uk/~pszit/benelearn.html





12th September 2016

Outline

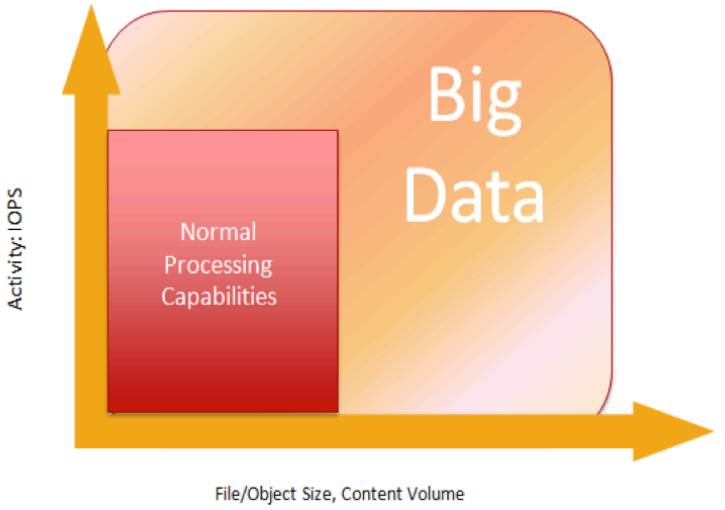
- What is Big data?
- How to deal with Data Intensive applications?
- Big Data Analytics
- A demo with MLlib
- Conclusions

What is Big Data?

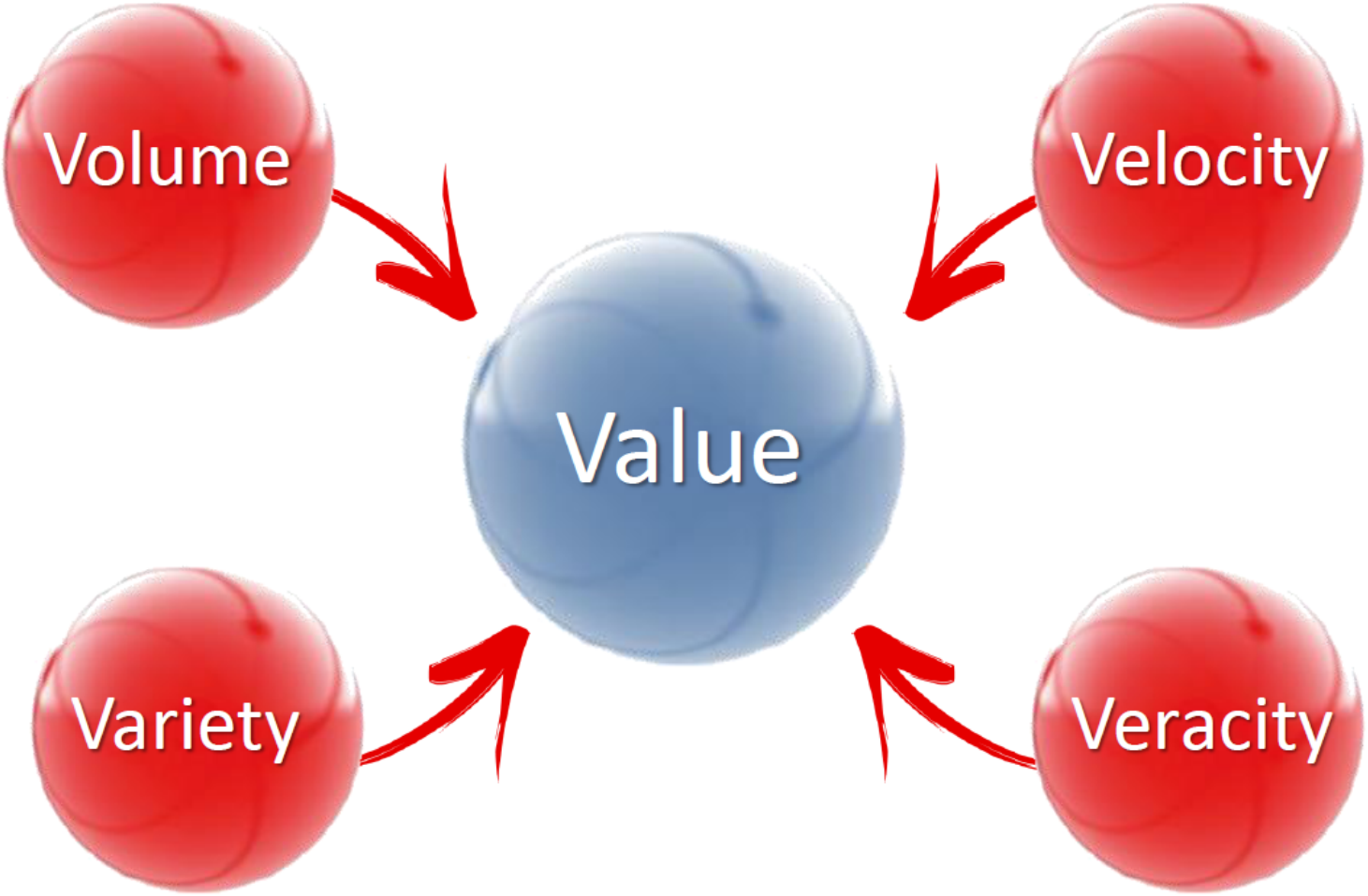
There is no a standard definition!

"*Big Data*" involves data whose volume, diversity and complexity requires new techniques, algorithms and analyses to extract valuable knowledge (hidden) .

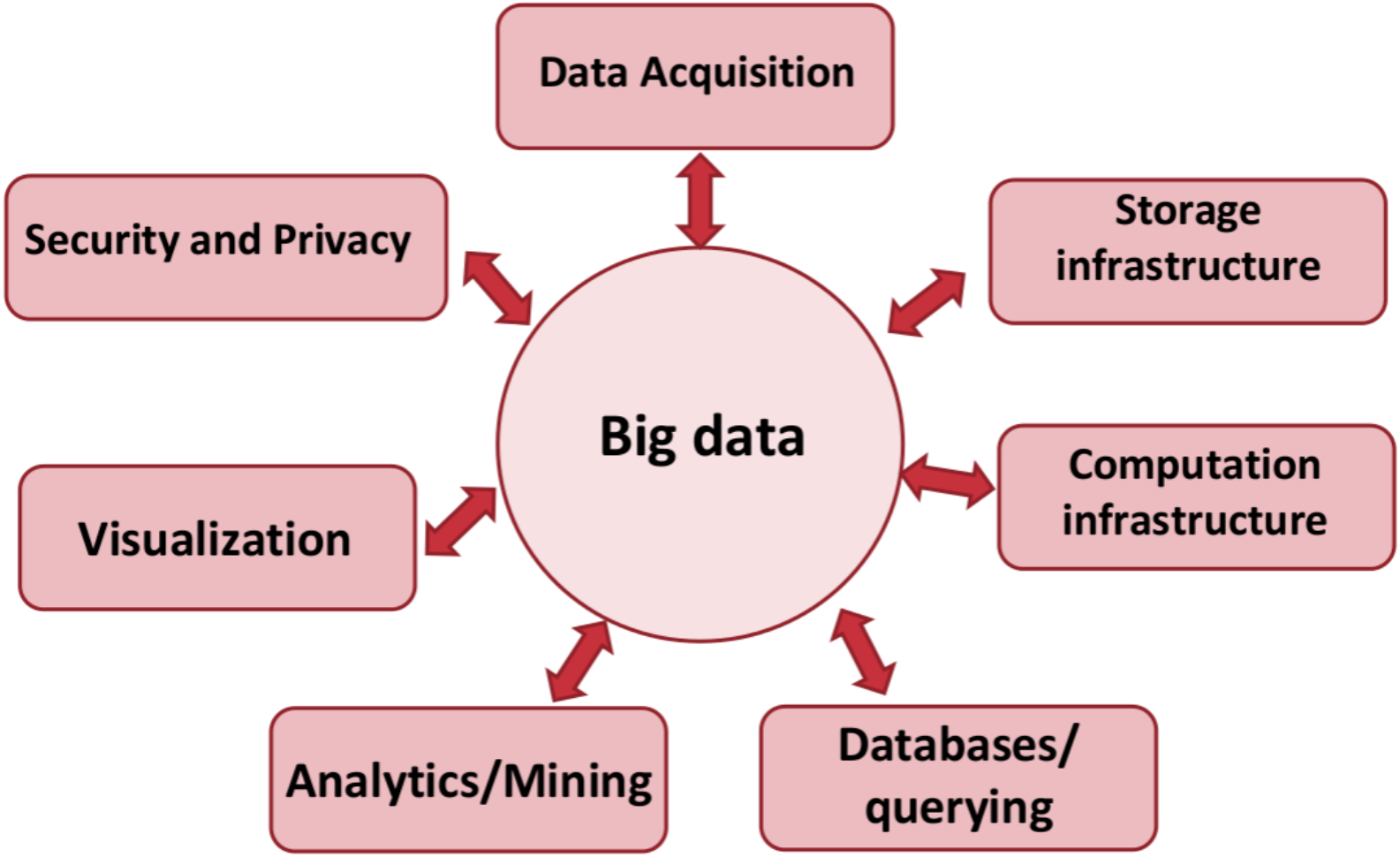
Data Intensive applications



What is Big Data? The 5V's definition



Big data has many faces



Outline

- ❑ What is Big data?
- ❑ How to deal with Data Intensive applications?
- ❑ Big Data Analytics
- ❑ A demo with MLlib
- ❑ Conclusions

How to deal with data intensive applications?

- **Problem statement:** scalability to big data sets.
- **Example:**
  - Explore 100 TB by 1 node @ 50 MB/sec = 23 days
  - Exploration with a cluster of 1000 nodes = 33 minutes
- **Solution➔ Divide-And-Conquer**

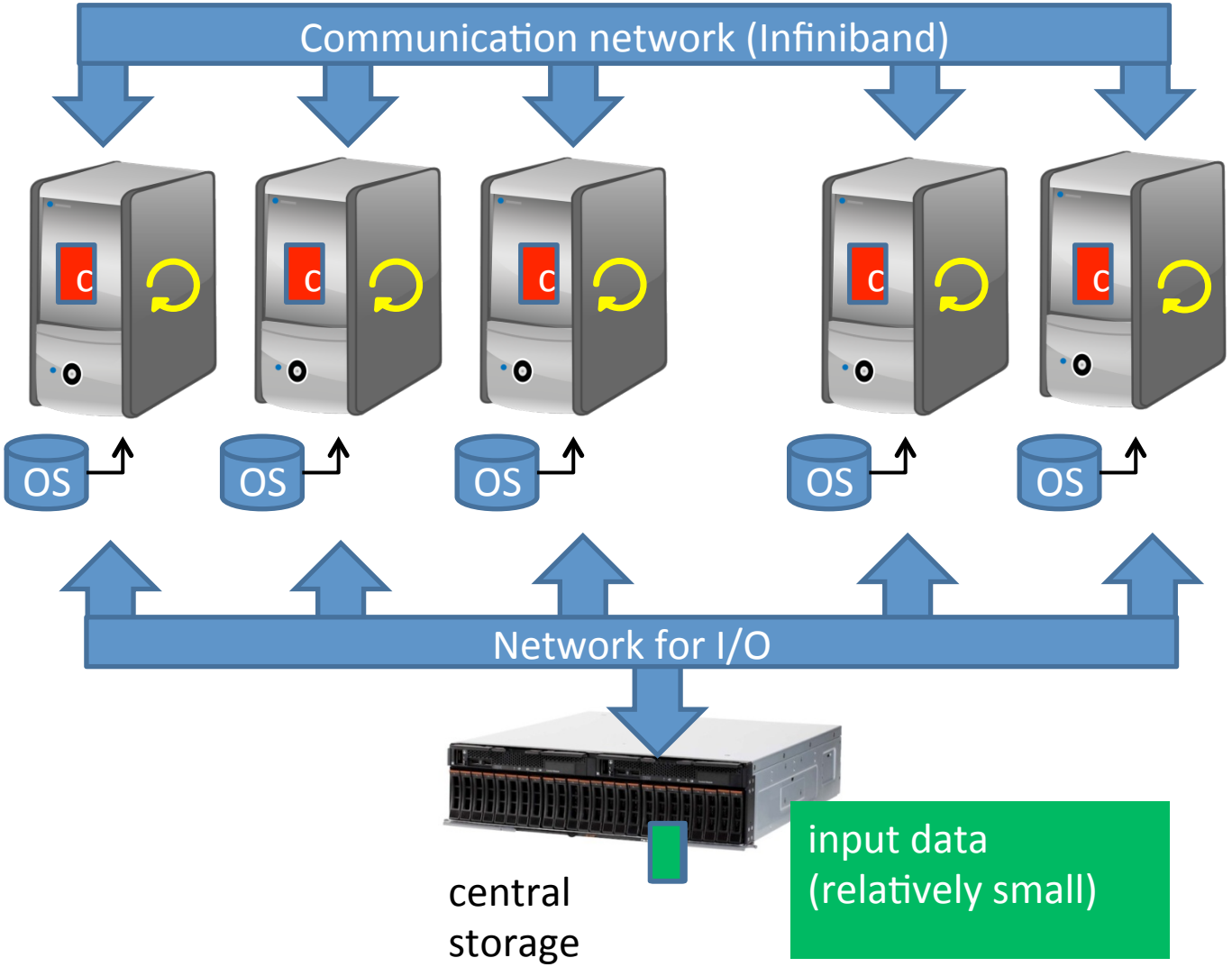
What happens if we have to manage 1000 or 10000 TB?

MapReduce



- Parallel Programming model
- **Divide & conquer strategy**
  - **divide**: partition dataset into smaller, independent chunks to be processed in parallel (*map*)
  - **conquer**: combine, merge or otherwise aggregate the results from the previous step (*reduce*)
- Based on **simplicity** and **transparency** to the programmers, and assumes **data locality**.
- Becomes popular thanks to the open-source project Hadoop! (Used by Google, Facebook, Amazon, …)

Traditional HPC way of doing things



Lots of communication

worker nodes
(lots of them)

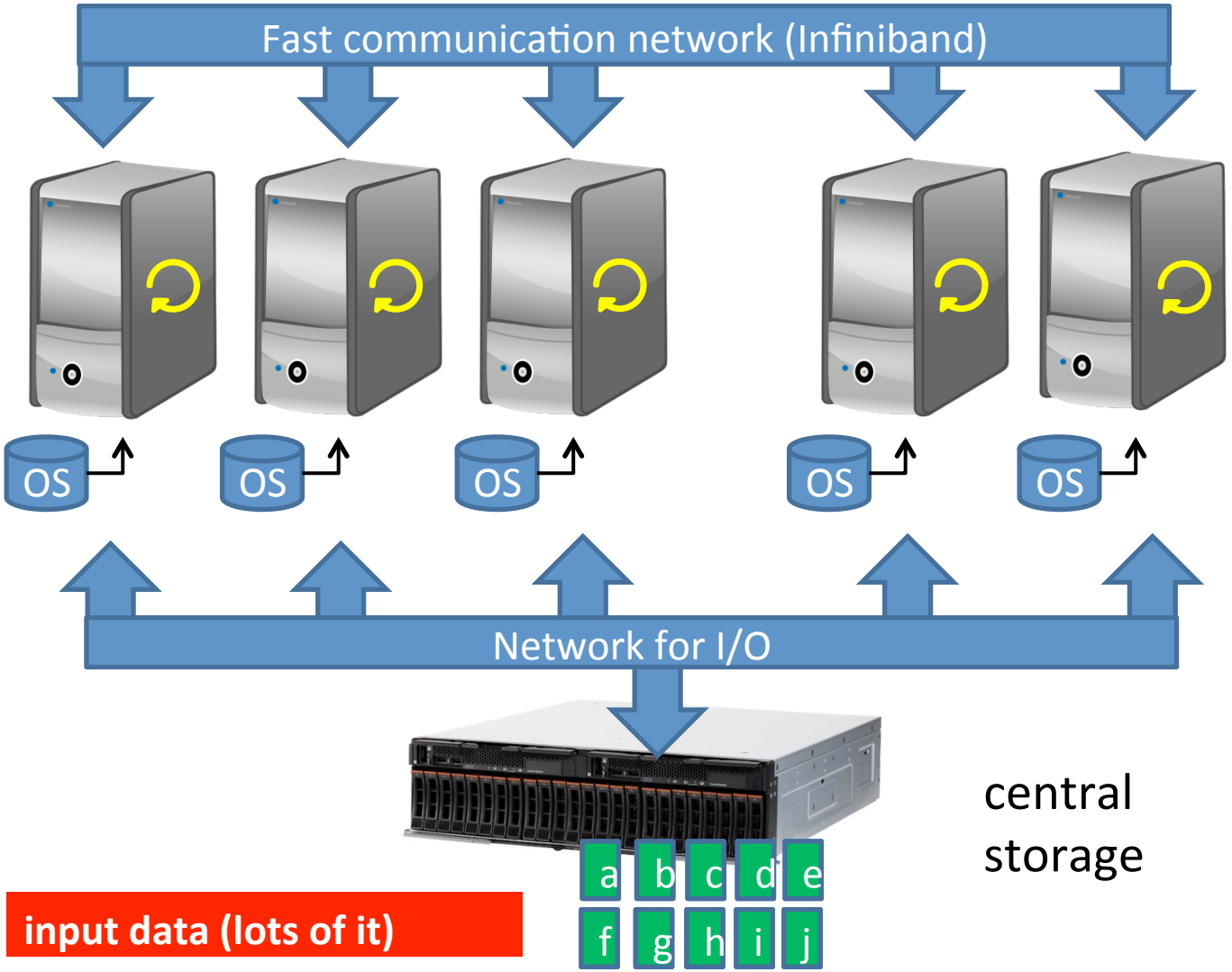
Lots of computations

Limited I/O

input data
(relatively small)

Source: Jan Fostier. Introduction to MapReduce and its Application to Post-Sequencing Analysis

Data-intensive jobs



Limited communication

Low compute intensity

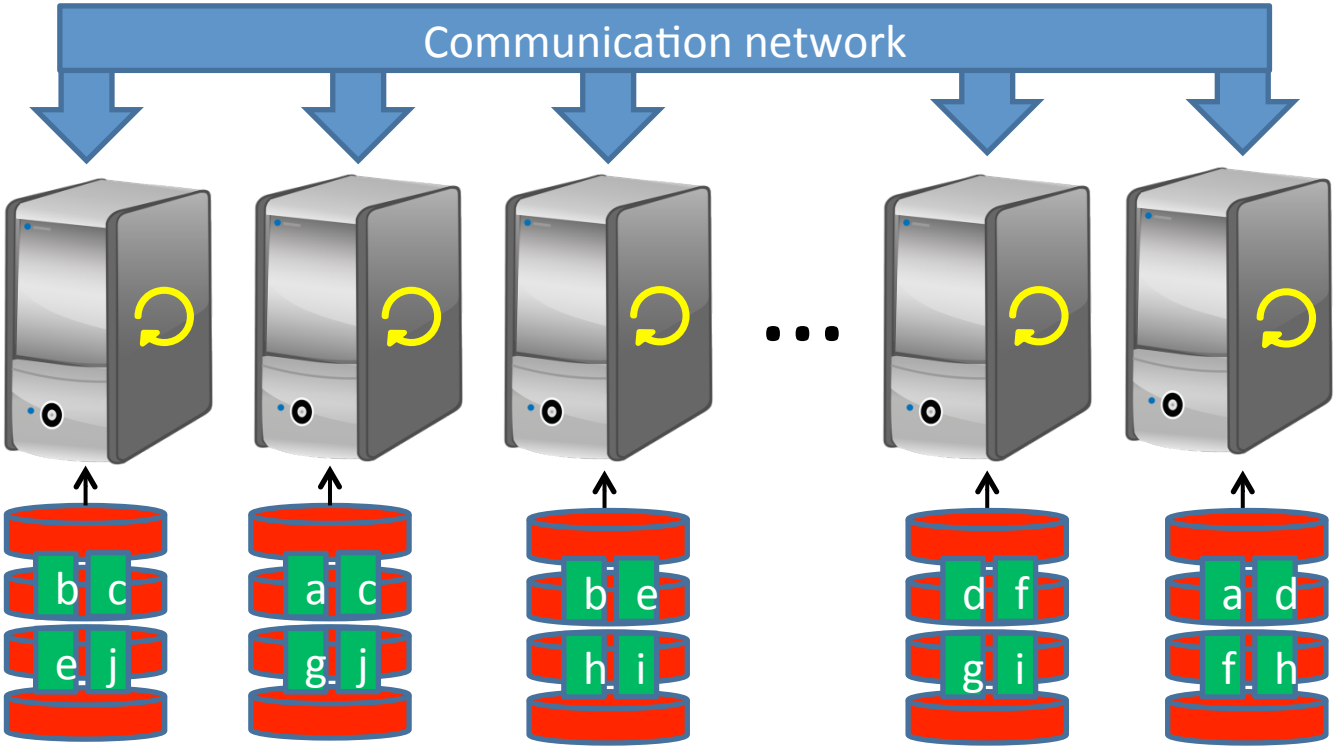
doesn't scale

Lots of I/O

central storage

input data (lots of it)

Data-intensive jobs



Limited communication

Low compute intensity

input data (lots of it)

Solution: store data on local disks of the nodes that perform computations on that data ("**data locality**")

Hadoop



- Hadoop is:
  - An **open-source** framework written in Java
  - Distributed storage of very large data sets (Big Data)
  - Distributed processing of very large data sets

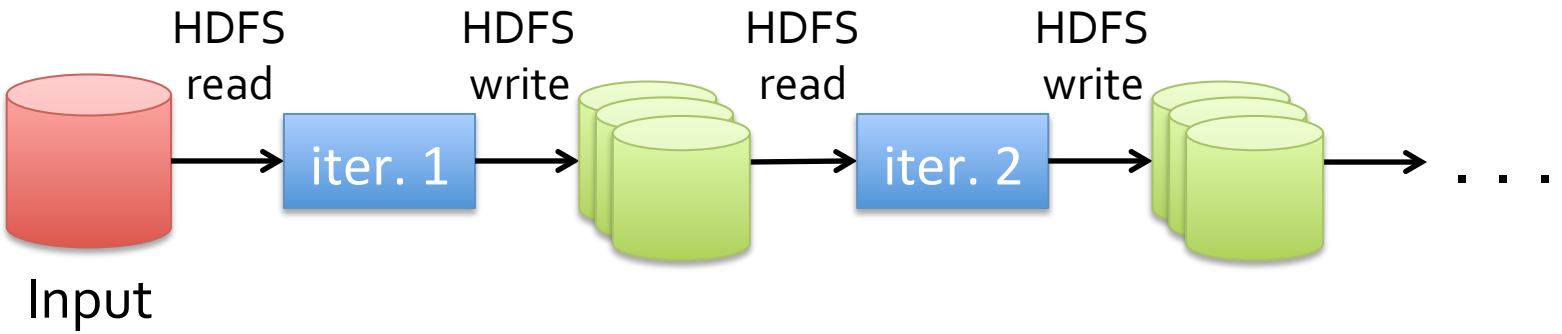
- This framework consists of a **number of modules**
  - *Hadoop Common*
  - ***Hadoop Distributed File System (HDFS)***
  - *Hadoop YARN* – resource manager
  - ***Hadoop MapReduce*** – programming model

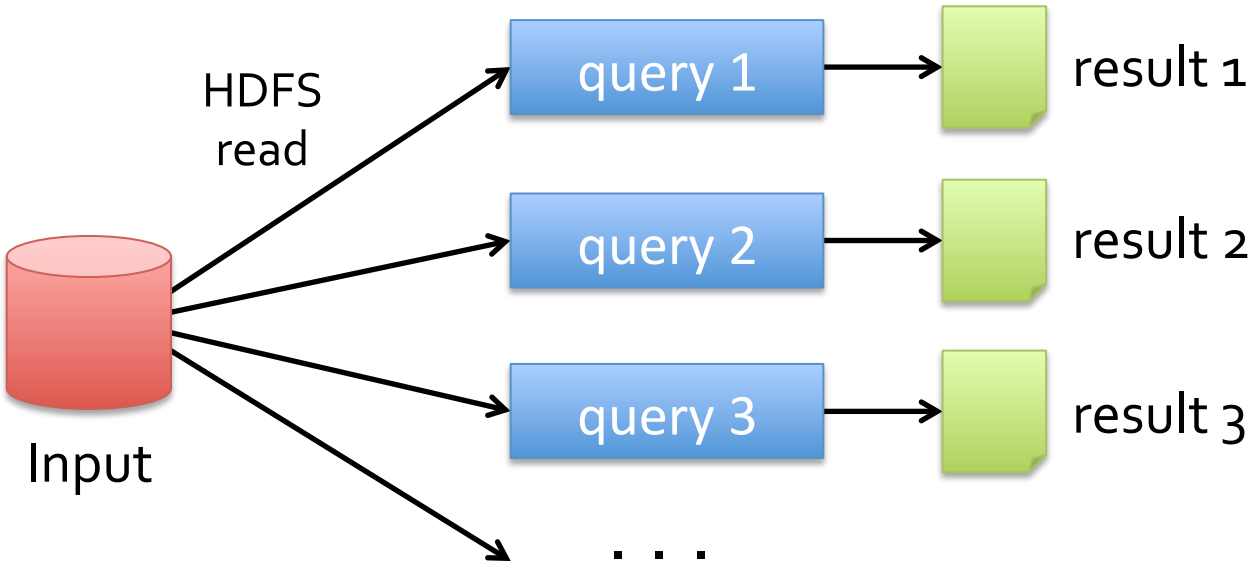
Hadoop MapReduce: Main Characteristics



- **Automatic parallelization:**
  - Depending on the size of the input data ➔ there will be multiple MAP tasks!
  - Depending on the number of Keys <k, value> ➔ there will be multiple REDUCE tasks!
- **Scalability:**
  - It may work over every data center or cluster of computers.
- **Transparent for the programmer**
  - Fault-tolerant mechanism.
  - Automatic communications among computers

Data Sharing in Hadoop MapReduce





Slow due to replication, serialization, and disk IO

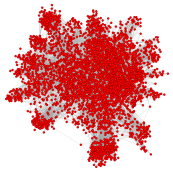
Paradigms that do not fit with Hadoop MapReduce

- **Directed Acyclic Graph (DAG) model:**
  - The DAG defines the dataflow of the application, and the vertices of the graph defines the operations on the data.
- **Graph model:**
  - More complex graph models that better represent the dataflow of the application.
  - Cyclic models -> Iterativity.
- **Iterative MapReduce model:**
  - An extented programming model that supports iterative MapReduce computations efficiently.

New platforms to overcome Hadoop's limitations



GIRAPH (APACHE Project)
(http://giraph.apache.org/)
Iterative graph processing



GPS - A Graph Processing System, (Stanford)
http://infolab.stanford.edu/gps/
Amazon's EC2



Distributed GraphLab
(Carnegie Mellon Univ.)
https://github.com/graphlab-code/graphlab
Amazon's EC2



Spark (UC Berkeley)
http://spark.incubator.apache.org/research.html



Twister (Indiana University)
http://www.iterativemapreduce.org/
Private Clusters



PrIter (University of Massachusetts Amherst, Northeastern University-China)
http://code.google.com/p/priter/
Private cluster and Amazon EC2 cloud

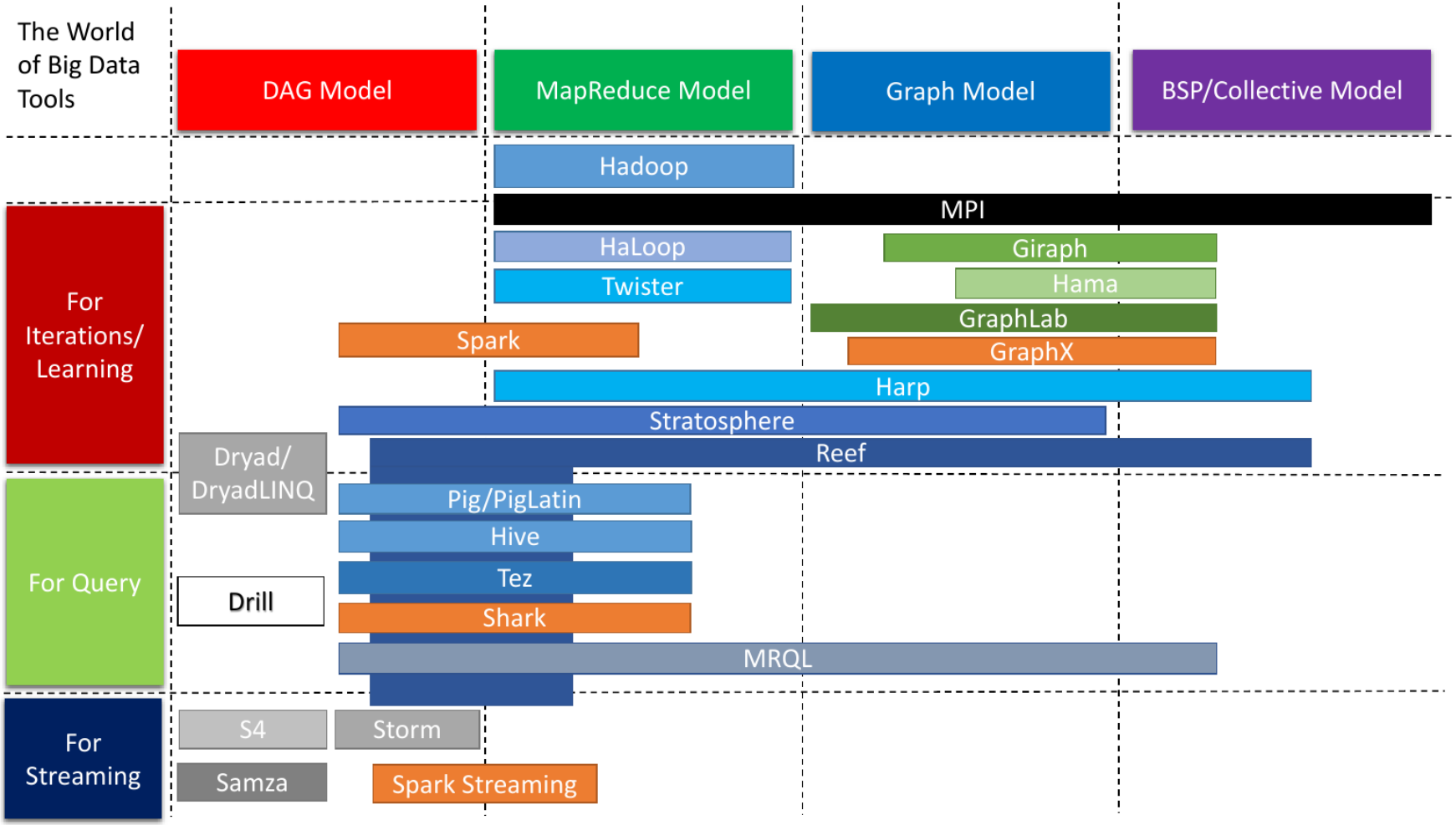


HaLoop
(University of Washington)
http://clue.cs.washington.edu/node/14
http://code.google.com/p/haloop/
Amazon's EC2

GPU based platforms
Mars
Grex



Big data technologies



What is Spark?

Fast and Expressive Cluster Computing Engine Compatible with Apache Hadoop

Up to **10×** faster on disk, **100×** in memory

2-5× less code

Efficient

- General execution graphs
- In-memory storage

Usable

- Rich APIs in Java, Scala, Python
- Interactive shell

Spark Goal

- Provide distributed memory abstractions for clusters to support apps with working sets
- **Retain the attractive properties of MapReduce**:
  - Fault tolerance (for crashes & stragglers)
  - Data locality
  - Scalability

Initial Solution: augment data flow model with "resilient distributed datasets" (RDDs)

RDDs in Detail

- An RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- There are two ways to create RDDs:
  - Parallelizing an existing collection in your driver program
  - Referencing a dataset in an external storage system, such as a shared filesystem, HDFS, Hbase.
- *Can be cached for future reuse*

Operations with RDDs

- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs
- Actions (e.g. count, collect, save)
  - Return a result or write it to storage

Transformations (define a new RDD)
map filter sample union groupByKey reduceByKey join cache …

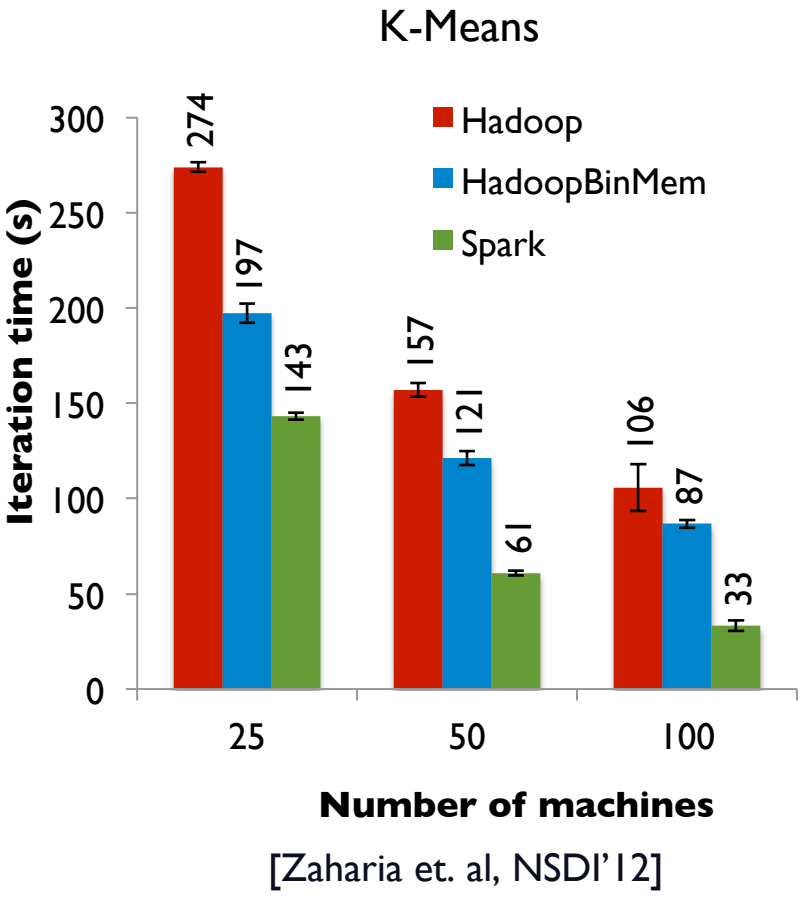
Parallel operations (return a result to driver)
reduce collect count save lookupKey …

Spark vs. hadoop

Lines of code for K-Means

Spark ~ 90 lines –

Hadoop ~ 4 files, > 300 lines



[Zaharia et. al, NSDI'12]

M. Zaharia et al. **Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing**. NSDI 2012.

Apache Spark – new collections

DataFrame (Spark 1.3+)

- Equivalent to a table in a relational database (data frame in R/ Python)

- Avoid Java serialization performed by RDDs.

- API natural for developers who are familiar with building query plans (e.g. SQL expressions).

Datasets (Spark 1.6+)

- Best of both DataFrame and RDDs.
- Functional transformations (map, flatMap, filter, etc)
- Spark SQL's optimised execution engine.

Flink

https://flink.apache.org/



Flink | Overview | Features | Downloads | FAQ | Quickstart | Documentation

Apache Flink is an open source platform for distributed stream and batch data processing.

Flink's core is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams.

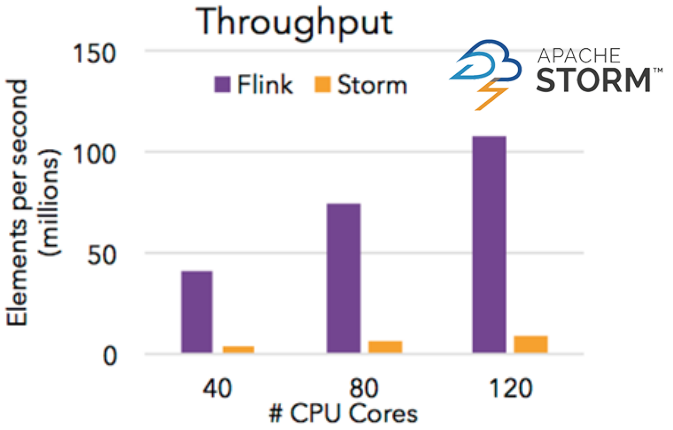
Flink includes **several APIs** for creating applications that use the Flink engine:

1. DataStream API for unbounded streams embedded in Java and Scala, and
2. DataSet API for static data embedded in Java, Scala, and Python,
3. Table API with a SQL-like expression language embedded in Java and Scala.

Flink also bundles **libraries for domain-specific use cases**:

1. CEP, a complex event processing library,
2. Machine Learning library, and
3. Gelly, a graph processing API and library.

You can **integrate** Flink easily with other well-known open source systems both for data input and output as well as deployment.



Batch on Streaming

Batch processing applications run efficiently as special cases of stream processing applications.

Streaming First

High throughput and low latency stream processing with exactly-once guarantees.

APIs, Libraries, and Ecosystem

DataSet, DataStream, and more. Integrated with the Apache Big Data stack.

2001-2010

2010-2016

Big Data: Technology and Chronology

The Apache Software Foundation



BIG DATA

2001
3V's Gartner
Doug Laney

Google



Hadoop Ecosystem



2009-2013 Flink
TU Berlin
Flink Apache (Dec. 2014) Volker Markl

2004
MapReduce
Google
Jeffrey Dean

Big Data

Flink

YAHOO!

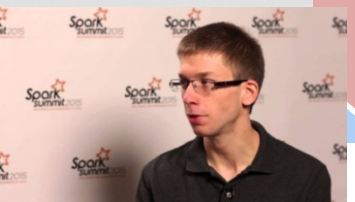
hadoop



2008
Hadoop
Yahoo!
Doug Cutting

2010 Spark
U Berckeley
Apache Spark
Feb. 2014
Matei Zaharia

Spark
Lightning-Fast Cluster Computing



2010-2016:

Big Data Analytics: Mahout, MLLib, …

Hadoop Ecosystem

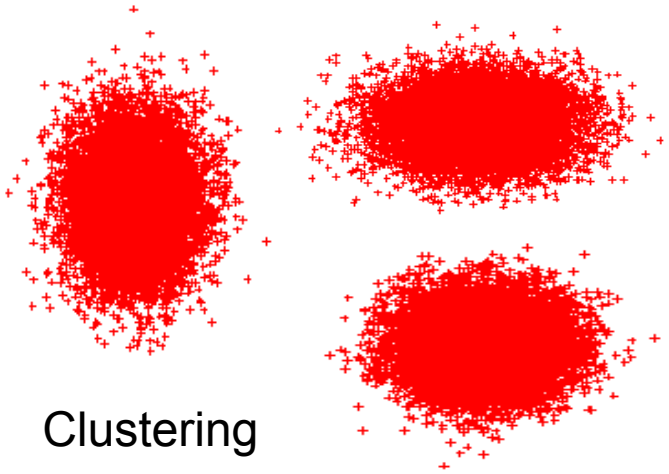
Applications New Technology

Outline

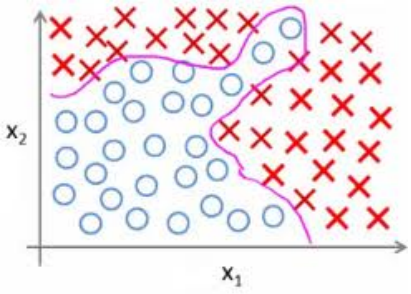
- ❑ What is Big data?
- ❑ How to deal with Data Intensive applications?
- ❑ Big Data Analytics
- ❑ A demo with MLlib
- ❑ Conclusions

Big Data Analytics

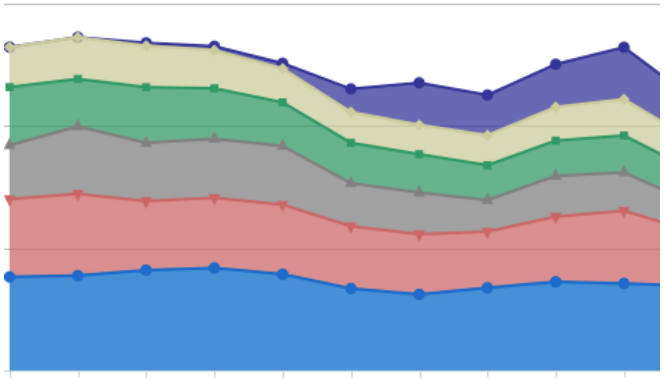
Potential scenarios



Clustering

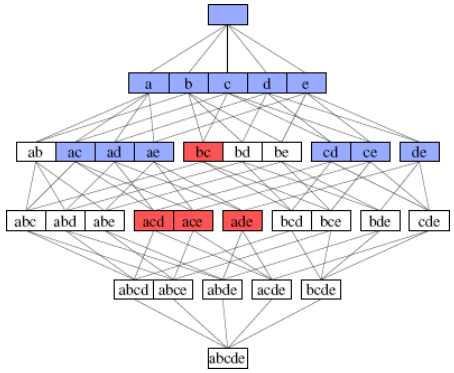


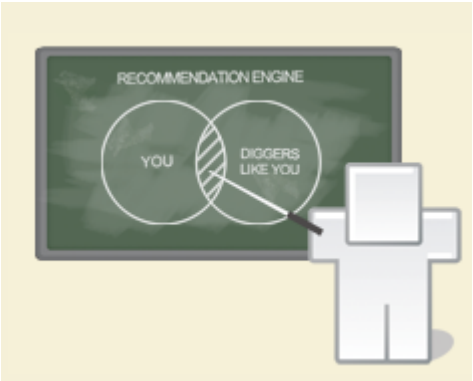
Classification



Real Time Analytics/
Big Data Streams

Association





Recommendation
Systems



Social Media Mining
Social Big Data

Big Data Analytics: A 3 generational view

Generation	1st Generation	2nd Generation	3rd Generation
Examples	SAS, R, Weka, SPSS, KEEL	Mahout, Pentaho, Cascading	Spark, Haloop, GraphLab, Pregel, Giraph, ML over Storm
Scalability	Vertical	Horizontal (over Hadoop)	Horizontal (Beyond Hadoop)
Algorithms Available	Huge collection of algorithms	Small subset: sequential logistic regression, linear SVMs, Stochastic Gradient Descendent, k-means clustering, Random forest, etc.	Much wider: CGD, ALS, collaborative filtering, kernel SVM, matrix factorization, Gibbs sampling, etc.
Algorithms Not Available	Practically nothing	Vast no.: Kernel SVMs, Multivariate Logistic Regression, Conjugate Gradient Descendent, ALS, etc.	Multivariate logistic regression in general form, k-means clustering, etc. – Work in progress to expand the set of available algorithms
Fault-Tolerance	Single point of failure	Most tools are FT, as they are built on top of Hadoop	FT: HaLoop, Spark Not FT: Pregel, GraphLab, Giraph

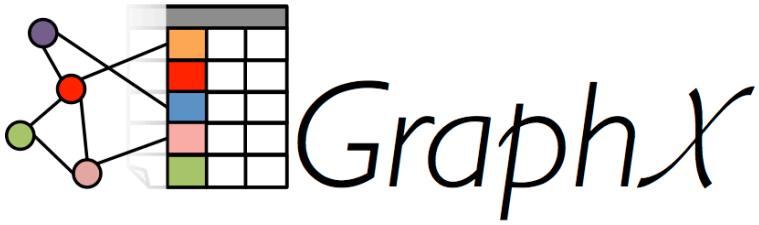
Mahout (Samsara)

- First ML library initially based on Hadoop MapReduce.
- Abandoned MapReduce implementations from version 0.9.
- Nowadays it is focused on a new **math environment** called **Samsara.**
- It is integrated with **Spark, Flink** and **H2O**
- Main algorithms:
  - Stochastic Singular Value Decomposition (ssvd, dssvd)
  - Stochastic Principal Component Analysis (spca, dspca)
  - Distributed Cholesky QR (thinQR)
  - Distributed regularized Alternating Least Squares (dals)
  - Collaborative Filtering: Item and Row Similarity
  - Naive Bayes Classification

http://mahout.apache.org/

Spark Libraries









MLlib types, algorithms and utilities

This lists functionality included in `spark.mllib`, the main MLlib API.

- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naive Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - isotonic regression
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
  - Gaussian mixture
  - power iteration clustering (PIC)
  - latent Dirichlet allocation (LDA)
  - streaming k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
  - FP-growth
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)
- PMML model export

https://spark.apache.org/mllib/

As of Spark 2.0

Announcement: DataFrame-based API is primary API

The MLlib RDD-based API is now in maintenance mode.

As of Spark 2.0, the RDD-based APIs in the `spark.mllib` package have entered maintenance mode. The primary Machine Learning API for Spark is now the DataFrame-based API in the `spark.ml` package.

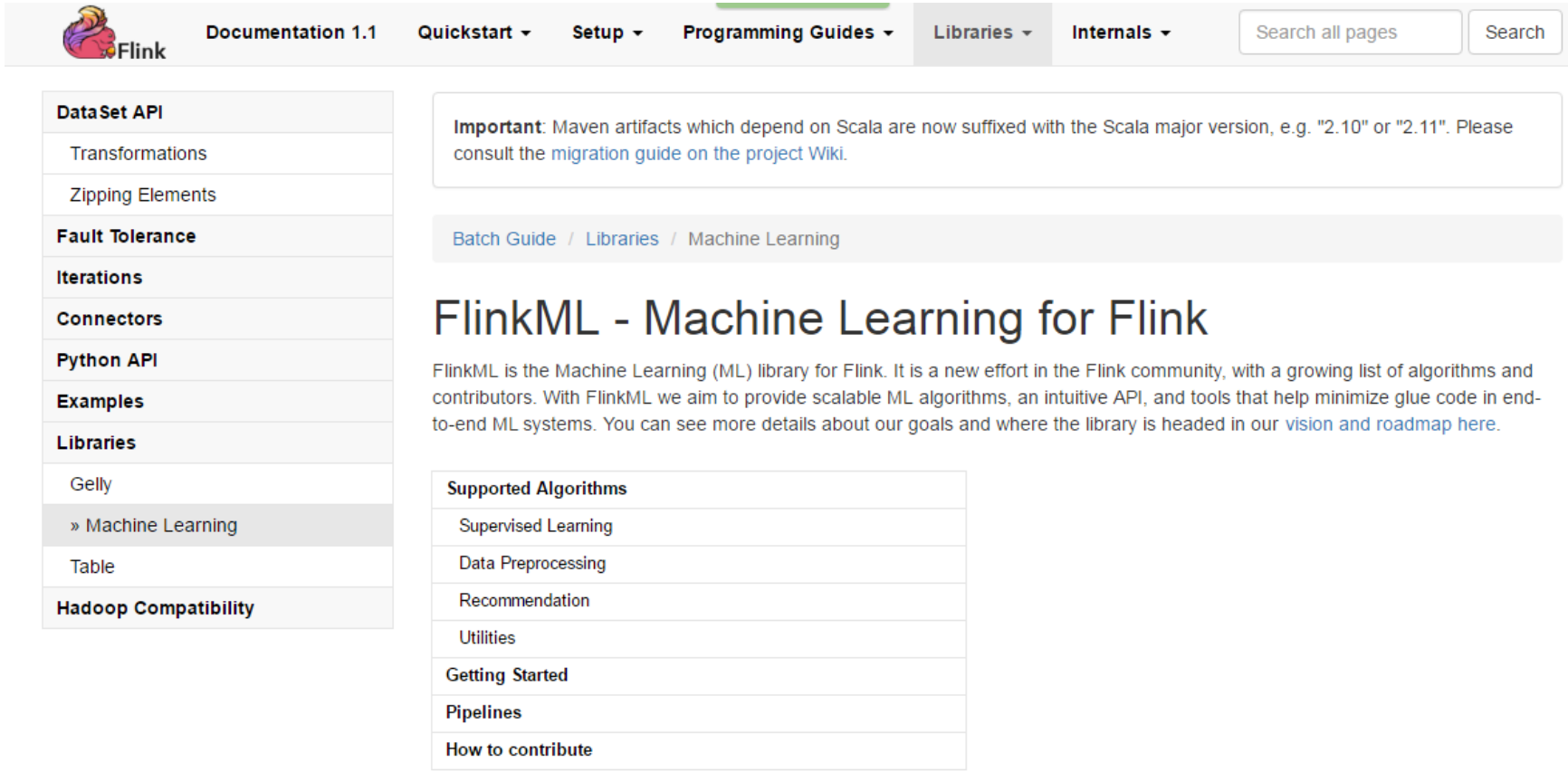
What are the implications?

- MLlib will still support the RDD-based API in `spark.mllib` with bug fixes.
- MLlib will not add new features to the RDD-based API.
- In the Spark 2.x releases, MLlib will add features to the DataFrames-based API to reach feature parity with the RDD-based API.
- After reaching feature parity (roughly estimated for Spark 2.2), the RDD-based API will be deprecated.
- The RDD-based API is expected to be removed in Spark 3.0.

Why is MLlib switching to the DataFrame-based API?

- DataFrames provide a more user-friendly API than RDDs. The many benefits of DataFrames include Spark Datasources, SQL/DataFrame queries, Tungsten and Catalyst optimizations, and uniform APIs across languages.
- The DataFrame-based API for MLlib provides a uniform API across ML algorithms and across multiple languages.
- DataFrames facilitate practical ML Pipelines, particularly feature transformations. See the Pipelines guide for details.

FlinkML



https://ci.apache.org/projects/flink/flink-docs-master/apis/batch/libs/ml/

Outline

- ❑ What is Big data?
- ❑ How to deal with Data Intensive applications?
- ❑ Big Data Analytics
- ❑ A demo with MLlib
- ❑ Conclusions

Demo

- In this demo I will show two ways of working with Apache Spark:
  - Interactive mode with Spark Notebook.
  - Standalone mode with Scala IDE.

- All the code used in this presentation is available at:

http://www.cs.nott.ac.uk/~pszit/benelearn.html

DEMO with Spark Notebook in local

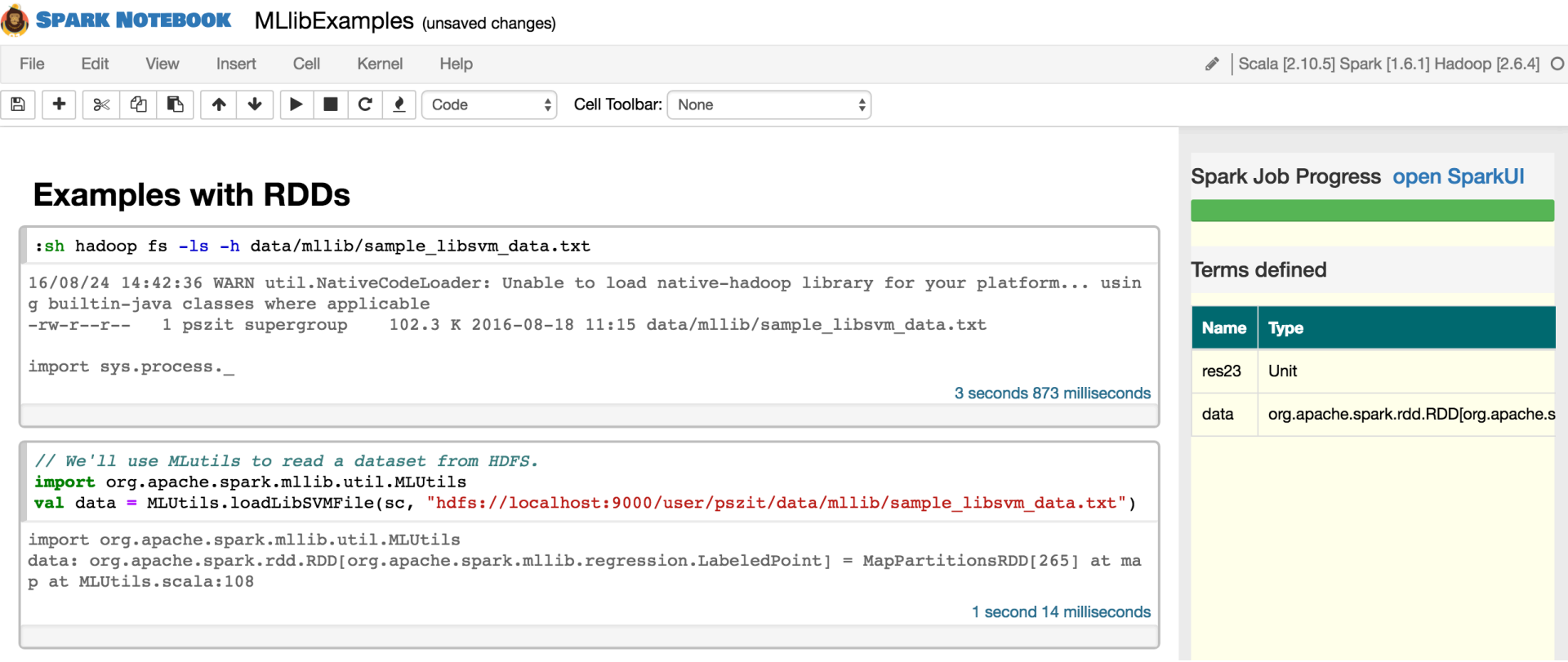


SPARK NOTEBOOK

http://spark-notebook.io/



DEMO with Spark Notebook in local



DEMO with Spark Notebook in local

```
val count = data.count
val sample = data.sample(false, 5d/count, 123456L) // no replacement.
sample.collect()
```

Show: 10

label	features
1	(692,[99,100,101,127,128,129,130,154,155,156,157,158,182,183,184,185,186,209,210,211,212,213,237,238,239,240,241,264,265,266,267,26 [166.0,222.0,55.0,197.0,254.0,218.0,5.0,29.0,249.0,254.0,254.0,9.0,45.0,254.0,254.0,174.0,2.0,4.0,164.0,254.0,254.0,85.0,146.0,254.0,254.0,
0	(692,[153,154,155,156,157,158,159,160,161,179,180,181,182,183,184,185,186,187,188,189,190,206,207,208,209,210,211,212,213,214,215,2 [46.0,105.0,254.0,254.0,254.0,254.0,255.0,239.0,41.0,37.0,118.0,222.0,254.0,253.0,253.0,253.0,253.0,253.0,211.0,54.0,14.0,200.0,253
0	(692,[152,153,154,155,156,157,158,159,179,180,181,182,183,184,185,186,187,188,206,207,208,209,210,211,212,213,214,215,216,233,234,2 [203.0,254.0,252.0,252.0,252.0,214.0,51.0,20.0,62.0,221.0,252.0,250.0,250.0,250.0,252.0,250.0,160.0,20.0,62.0,211.0,250.0,252.0,250.0,250
0	(692,[234,235,237,238,239,240,241,242,243,244,245,260,261,262,263,265,266,267,268,269,270,271,272,273,274,287,288,289,290,293,294,3
0	(692,[123,124,125,126,127,128,129,130,150,151,152,153,154,155,156,157,158,159,178,179,180,181,182,183,184,185,186,187,188,189,204,2 [66.0,254.0,254.0,58.0,60.0,59.0,59.0,50.0,73.0,233.0,253.0,253.0,148.0,254.0,253.0,253.0,232.0,73.0,156.0,253.0,253.0,253.0,117.0,255.0,2

DEMO with Spark Notebook in local

```
// Split the data into training and test sets (30% held out for testing)
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))
```

```
print(data.count() + ";"+ trainingData.count() + ";" + testData.count())
```

Decision Tree

```
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.tree.{impurity, DecisionTree, RandomForest}
import org.apache.spark.mllib.tree.configuration.{Algo, Strategy}
import org.apache.spark.mllib.tree.configuration.Algo._
```

```
val algo: Algo = Classification
val strategy = new Strategy(algo = algo,
          impurity = impurity.Entropy,   // Options: Gini, Entropy, Variance
          maxDepth = 5,  // Maximum allowed is 30
          maxBins = 1000,
          numClasses = 2,
          minInstancesPerNode = 2,
          minInfoGain = 0
          )
```

```
algo: org.apache.spark.mllib.tree.configuration.Algo.Algo = Classification
strategy: org.apache.spark.mllib.tree.configuration.Strategy = org.apache.spark.mllib.tree.configuration.Strategy@46937db5
```

DEMO with Spark Notebook in local

```
val modelDT = DecisionTree.train(trainingData,strategy)
```

```
modelDT: org.apache.spark.mllib.tree.model.DecisionTreeModel = DecisionTreeModel classifier of depth 2 with
 5 nodes
```

2 seconds 114 milliseconds

```
// Evaluate model on test instances and compute test error
val labelAndPreds = testData.map { point =>
      val prediction = modelDT.predict(point.features)
      (point.label, prediction)
    }
val testDT = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
println("Test Error (DT) = " + testDT)
```

```
Test Error (DT) = 0.06060606060606061
labelAndPreds: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[283] at map at <console>:99
testDT: Double = 0.06060606060606061
```

1 second 472 milliseconds

DEMO with Spark Notebook in local

Random Forest

```
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel

val modelRF= RandomForest.trainClassifier(trainingData,strategy,100,"sqrt",12345678)
val labelAndPreds = testData.map { point =>
      val prediction = modelRF.predict(point.features)
      (point.label, prediction)
    }
val testRF = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
println("Test Error (RF) = " + testRF)
```

```
Test Error (RF) = 0.030303030303030304
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel
modelRF: org.apache.spark.mllib.tree.model.RandomForestModel =
TreeEnsembleModel classifier with 100 trees

labelAndPreds: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[308] at map at <console>:99
testRF: Double = 0.030303030303030304
```

2 seconds 478 milliseconds

DEMO with Spark Notebook in local

SVM model

```
import org.apache.spark.mllib.classification.SVMModel
import org.apache.spark.mllib.classification.SVMWithSGD

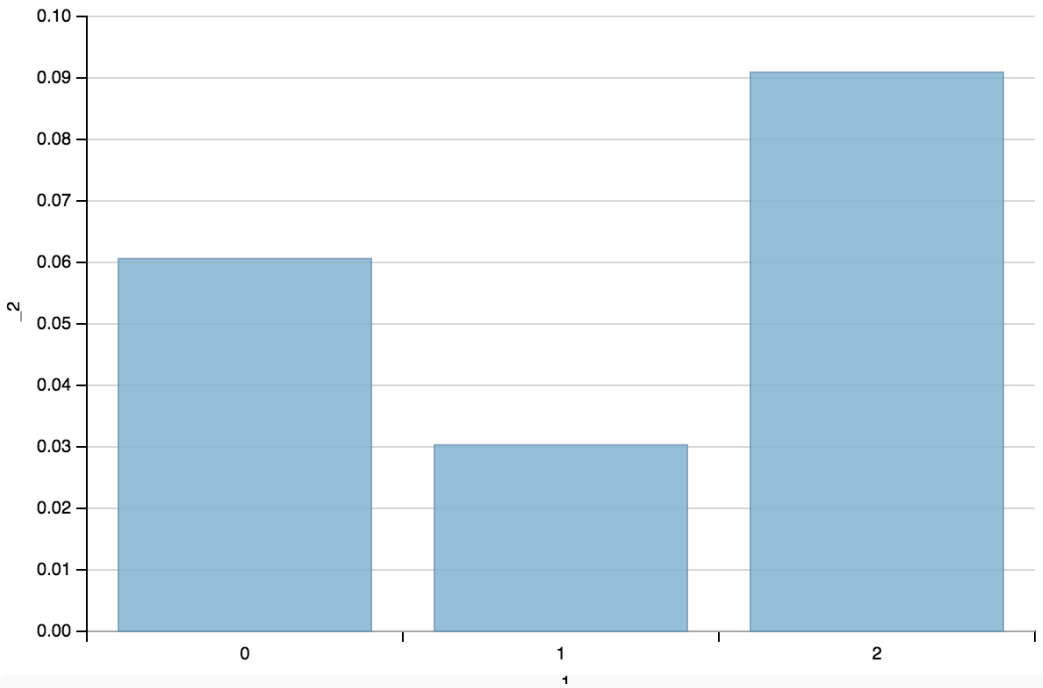
val modelSVM = SVMWithSGD.train(trainingData,100)
val labelAndPreds = testData.map { point =>
      val prediction = modelSVM.predict(point.features)
      (point.label, prediction)
    }
val testSVM = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
println("Test Error (SVM) = " + testSVM)
```

```
Test Error (SVM) = 0.09090909090909091
import org.apache.spark.mllib.classification.SVMModel
import org.apache.spark.mllib.classification.SVMWithSGD
modelSVM: org.apache.spark.mllib.classification.SVMModel = org.apache.spark.mllib.classification.SVMModel: i
ntercept = 0.0, numFeatures = 692, numClasses = 2, threshold = 0.0
labelAndPreds: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[513] at map at <console>:101
testSVM: Double = 0.09090909090909091
```

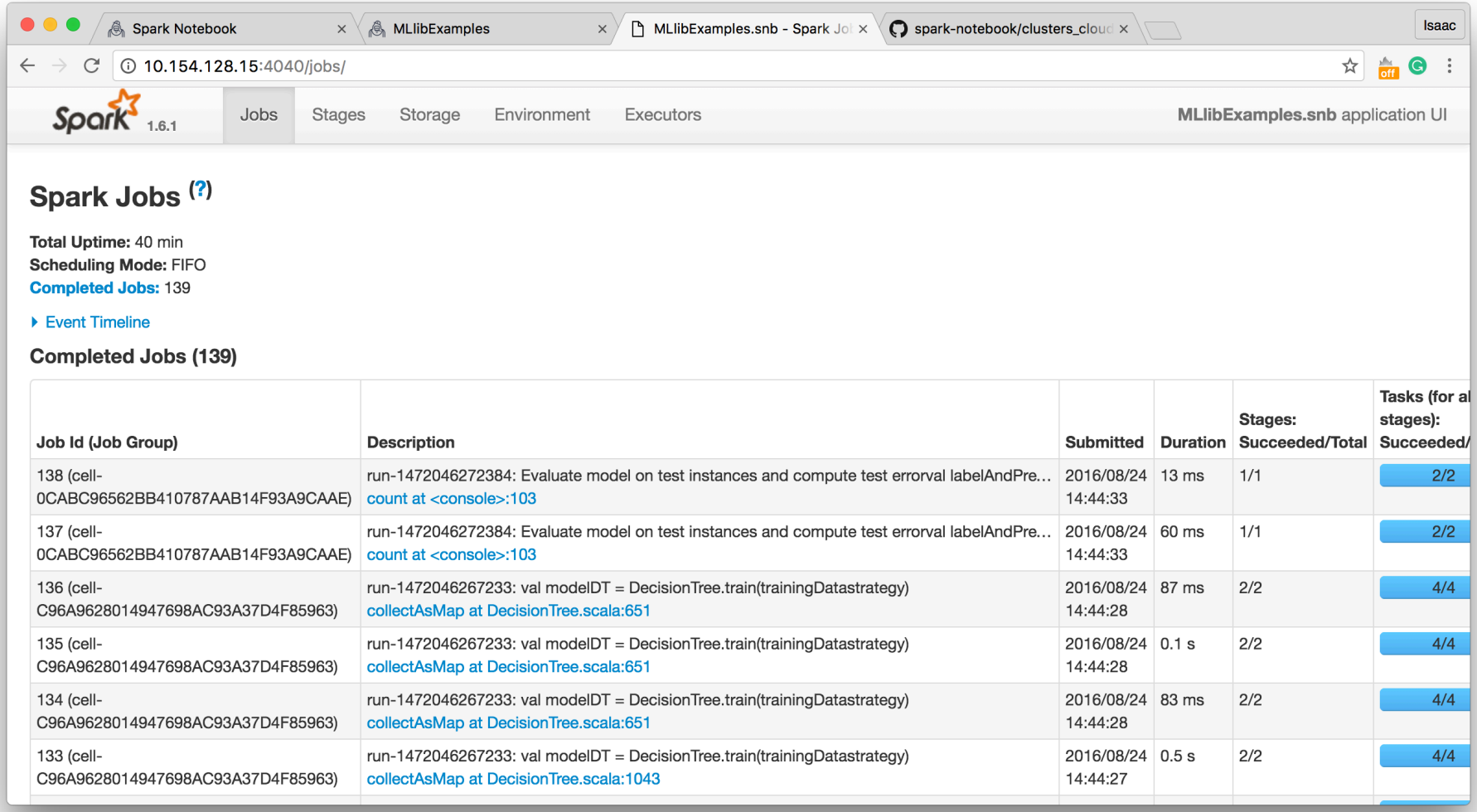
4 seconds 652 milliseconds

DEMO with Spark Notebook in local

```
List(testDT,testRF,testSVM)
```



DEMO with Spark Notebook in local



DEMO with Spark Notebook in local



Advantages:

- ✓ Interactive.
- ✓ Automatic plots.
- ✓ It allows connection with a cluster.
- ✓ Tab completion

Disadvantages:

- ❑ Built-in for specific spark versions.
- ❑ Difficult to integrate your own code.

DEMO with Scala IDE



http://scala-ide.org/

Example:
An Imbalanced Big Data problem

- Two main approaches to tackle this problem:
  - **Data sampling:**
    - **Undersampling,**
    - Oversampling
    - Hybrid approaches
  - Algorithmic modifications

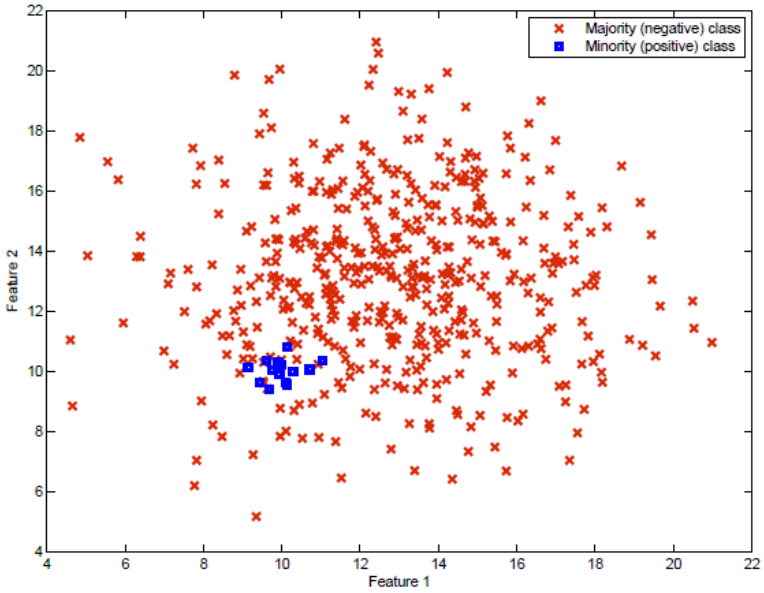
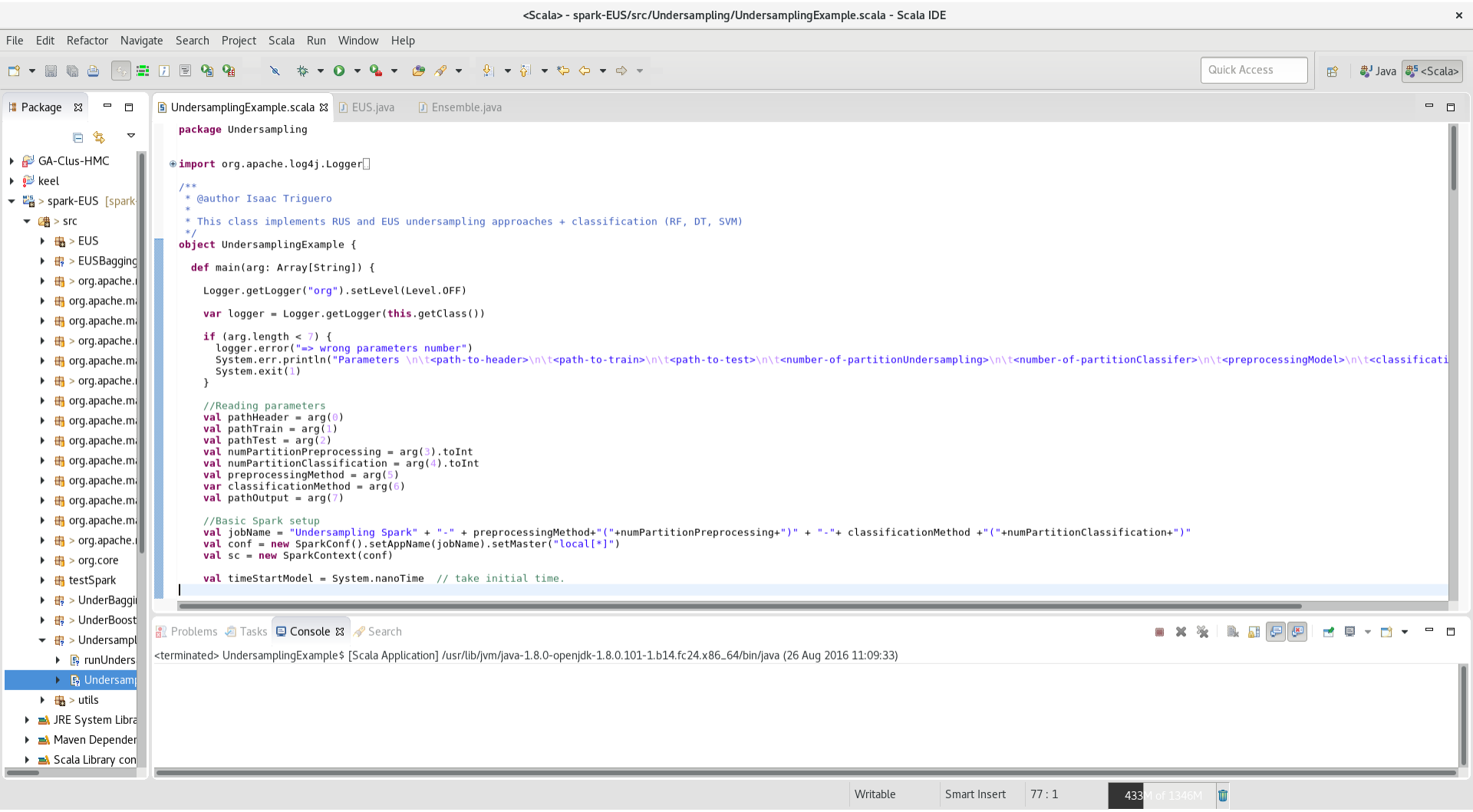


FIGURE : An example of an imbalanced data-set

I. Triguero et al, **Evolutionary Undersampling for Extremely Imbalanced Big Data Classification under Apache Spark**. IEEE Congress on Evolutionary Computation (CEC 2016), Vancouver (Canada), 640-647, July 24-29.

Imbalanced Big Data Classification with Spark



Imbalanced Big Data Classification with Spark

```
// Obtain the header.
var HeaderString = ""
 for(x <- sc.textFile(pathHeader).collect())  HeaderString = HeaderString+"\n"+ x

// Read the training dataset with numPartitionPreprocessing.
val trainRaw = sc.textFile(pathTrain: String, numPartitionPreprocessing).persist() // we use persist to re-use this variable.

print("trainRaw.count(): " + trainRaw.count()+"\n")

// Here some code to read the header file which will be used to transform the data into LabeledPoints (Spark API).
val typeConversion = KeelParser.parseHeaderFile(sc, pathHeader)
val bcTypeConv = sc.broadcast(typeConversion)


// Decision Tree and RandomForest parameters:
val algo: Algo = Classification

val strategy = new Strategy(algo = algo,
      impurity = impurity.Entropy,
      maxDepth = 5,  // Maximum allowed is 30
      maxBins = 1000,
      numClasses = 2,
      minInstancesPerNode = 2,
      minInfoGain = 0
     )

// initialising partial runtimes.
var PreprocessingTime = 0.0
var LearningTime = 0.0
var ClassificationTime = 0.0

// Preprocessing Stage:
var timeStartPreprocessing =System.nanoTime

val data = if (preprocessingMethod.equalsIgnoreCase("RUS")) {
    trainRaw.mapPartitions(dataset => RandomUndersamplingMixedDataset(dataset)).persist()
}else if (preprocessingMethod.equalsIgnoreCase("EUS")) {
    var param = "seed = 48127491\n" + "ISmethod = QstatEUB_M_GM\n" + "No. of window minority = 1\n" + "No. of window majority = -1";
    trainRaw.mapPartitions(dataset => preprocessingEUS(dataset, param, HeaderString)).persist()
}else{
    trainRaw // if you don't want to apply any preprocessing.
}

logger.info("=> Size of the preprocessed dataset: \"" + data.count() + "\"")
```

Imbalanced Big Data Classification with Spark

```
// coalescing if necessary.
val dataRepartitioned = if(numPartitionClassification!= numPartitionPreprocessing){
   data.coalesce(numPartitionClassification).persist()
}else{
   data
}

val endPreprocessing = System.nanoTime

PreprocessingTime = (endPreprocessing - timeStartPreprocessing)/ 1e9

// Learning phase:

var modelDT:DecisionTreeModel = null
var modelRF:RandomForestModel = null
var modelSVM:SVMModel = null

var timeStartLearning = System.nanoTime

// converting the resulting dataset to RDD[LabeledPoint]
val train = dataRepartitioned.map(line => KeelParser.parseLabeledPoint(bcTypeConv.value, line)).cache

print("train.count(): " + train.count()+"\n")

dataRepartitioned.unpersist()  // we can delete the

if (classificationMethod.equalsIgnoreCase("DecisionTree")) {
    modelDT = DecisionTree.train(train,strategy)
}else if (classificationMethod.equalsIgnoreCase("RandomForest")) {
    modelRF = RandomForest.trainClassifier(train,strategy,100,"sqrt",12345678)
}else if (classificationMethod.equalsIgnoreCase("SVM")) {
    modelSVM = SVMWithSGD.train(train,100)
}

val endLearning = System.nanoTime
LearningTime = (endLearning - timeStartLearning)/ 1e9

data.unpersist()
```

Imbalanced Big Data Classification with Spark

```
//  Classification of test set:

val timeStartClassification = System.nanoTime

// read test data
val testRaw = sc.textFile(pathTest: String, numPartitionClassification)
val testData = testRaw.map(line => KeelParser.parseLabeledPoint(bcTypeConv.value, line)).cache

val labelsTestData = testData.map { point =>
   point.label.toInt
  }.collect()                                          // take the label for evaluation

val sizeTest= labelsTestData.length

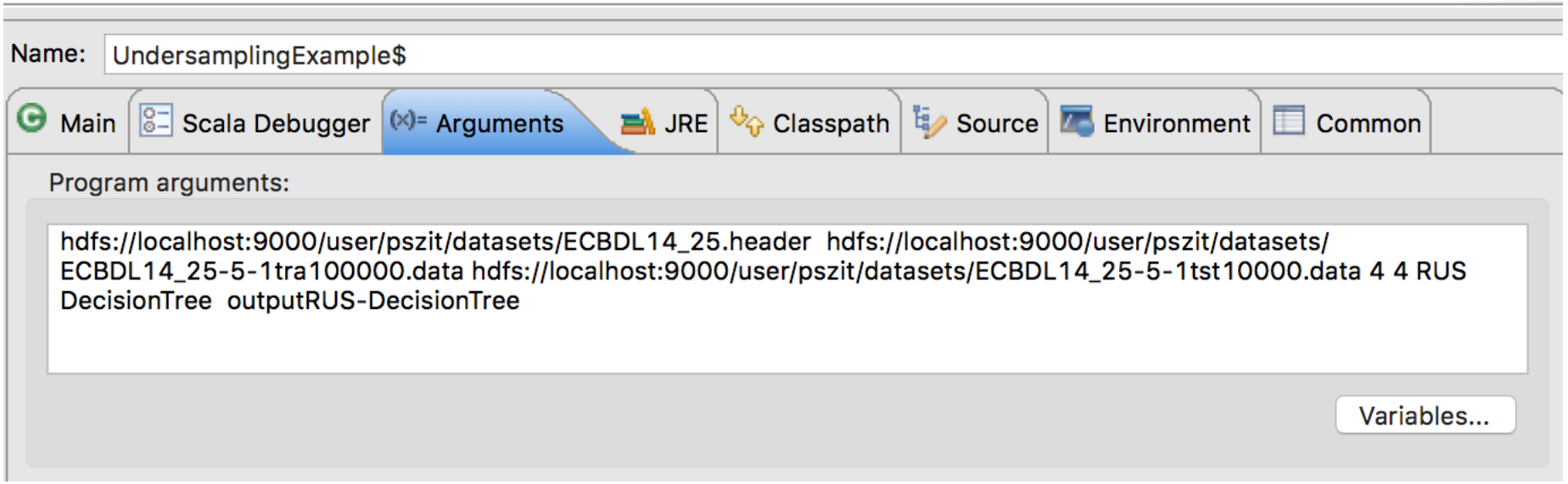
print("sizeTest: "+ sizeTest + "\n")

val Preds =  if (classificationMethod.equalsIgnoreCase("DecisionTree")) {
      testData.map { point =>
      val prediction = modelDT.predict(point.features)
      (prediction.toInt)
      }.collect()
}else if (classificationMethod.equalsIgnoreCase("RandomForest")) {
      testData.map { point =>
      val prediction = modelRF.predict(point.features)
      (prediction.toInt)
      }.collect()
}else  {
      testData.map { point =>
      val prediction = modelSVM.predict(point.features)
      (prediction.toInt)
       }.collect()
 }

val timeEndClassification = System.nanoTime

ClassificationTime = ClassificationTime + (timeEndClassification - timeStartClassification)/ 1e9
```

Run examples from ScalaIDE




```
16/09/09 13:38:29 INFO UndersamplingExample$: ######################
Confusion Matrix
7459                            2306
95                              140
############################################################
AUC:            0.6797975836410977
GM:             0.6745812510415957
############################################################
Preprocessing Time:             1.53729896 seconds
Learning Time:          9.558088184 seconds
Classification Time:            0.811491757 seconds
```

Run examples from terminal

```
$  mvn package -Dmaven.test.skip=true

$ /opt/spark/bin/spark-submit --master local[*] --class
Undersampling.UndersamplingExample target/EUS-0.0.1-
BETA.jar hdfs://localhost:9000/user/pszit/datasets/
ECBDL14_25.header  hdfs://localhost:9000/user/pszit/
datasets/ECBDL14_25-5-1tra100000.data hdfs://localhost:
9000/user/pszit/datasets/ECBDL14_25-5-1tst10000.data 4 4
RUS DecisionTree  /Users/pszit/outputRUS-DecisionTree
```

Outline

- ❑ What is Big data?
- ❑ How to deal with Data Intensive applications?
- ❑ Big Data Analytics
- ❑ A demo with MLlib
- ❑ Conclusions

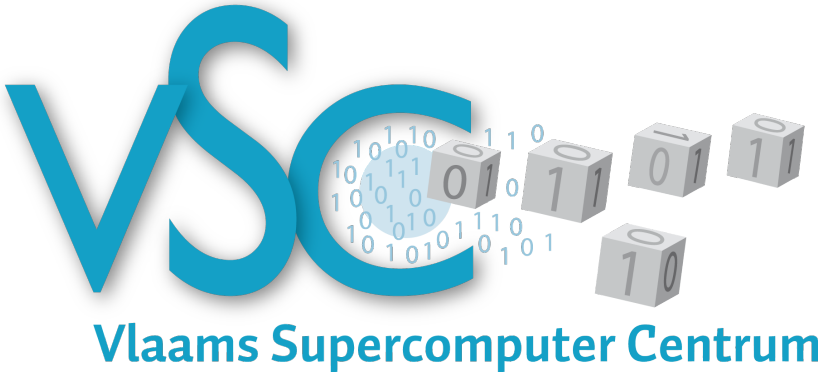
Conclusions

- We need new strategies to perform ML in big datasets
  - Choosing the right technology is like choosing the right data structure in a program.
- The world of big data is rapidly changing. Being up-to-date is difficult but necessary.
- Interactive notebooks are very useful for a quick start and standard experiments.

Acknowledgments







Thank you

BENELEARN 2016

BELGIAN-DUTCH CONFERENCE ON MACHINE LEARNING

Big Data Learning in Practice

Isaac Triguero

School of Computer Science

University of Nottingham

United Kingdom

Isaac.Triguero@nottingham.ac.uk

http://www.cs.nott.ac.uk/~pszit/





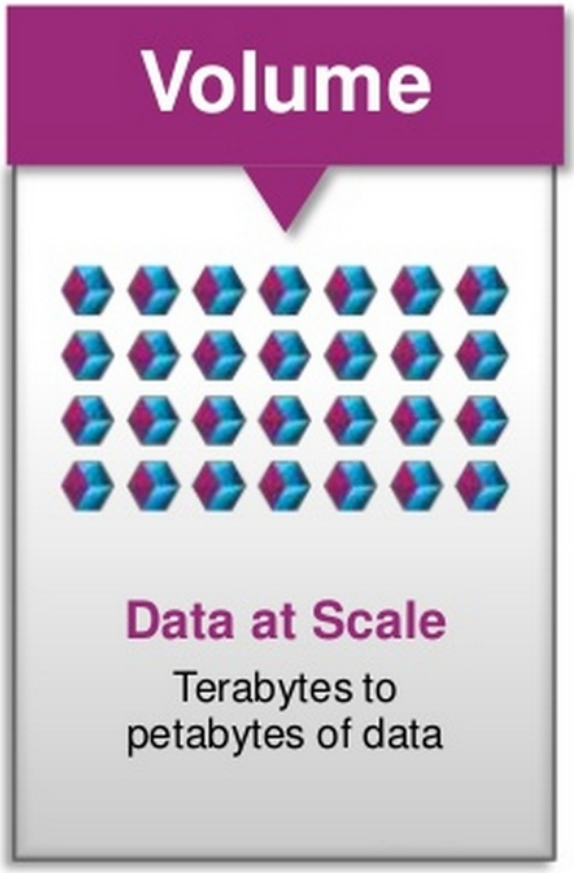
12th September 2016

Extra slides

What is Big Data? The 5V's definition

Volume: data at rest

- Vast amounts of data generated every second
- Data sets are becoming too large to store using traditional database technology
- Big data technology stores these data sets using distributed systems



What is Big Data? The 5V's definition

Velocity: data in motion

- Speed at which:
  - Data is generated
  - Data needs to be analyzed.
- Continuous data streams are being captured (e.g. from sensors or mobile devices) and produced
- Late decisions imply missed opportunities



What is Big Data? The 5V's definition

Variety: data in many forms

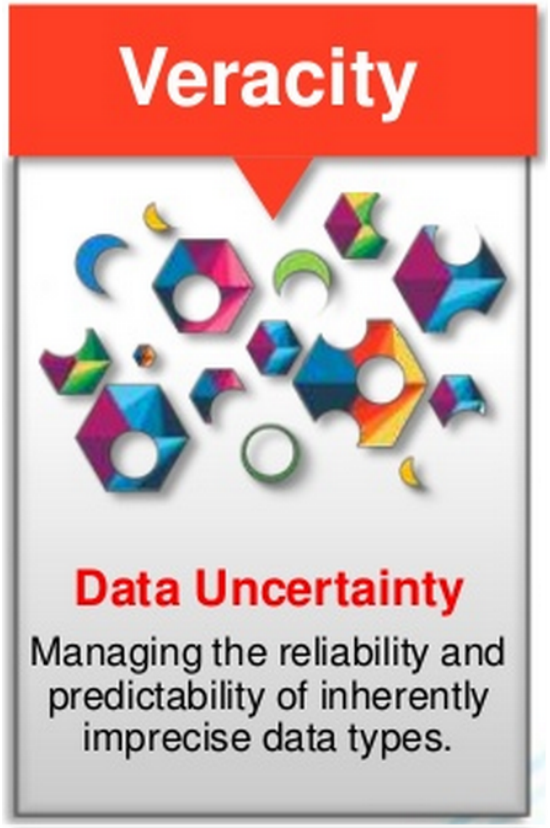
- One application may generate many different kind of data
- Several formats and structures:
  - Structured data:
    - Tables, relation databases
  - Unstructured data:
    - Text, images, audio, video.



What is Big Data? The 5V's definition

Veracity: data in doubt

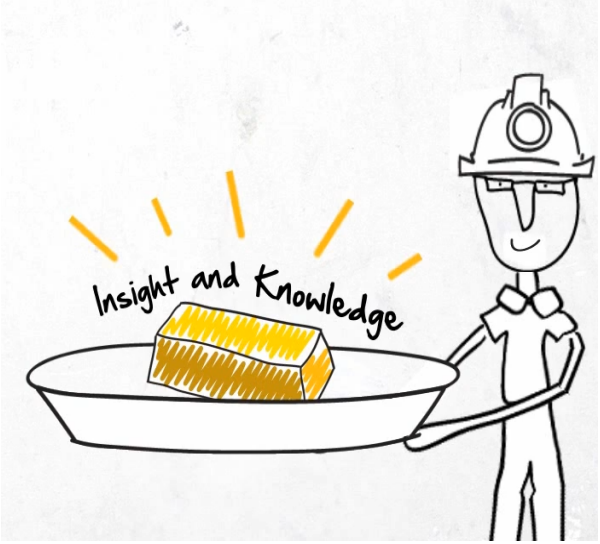
- Uncertainty about the quality of the data.
  - E.g. natural language processing on social media: typos, abbreviations, colloquial speech.
- Data may be missing, ambiguous, or even completely wrong.



What is Big Data? The 5V's definition

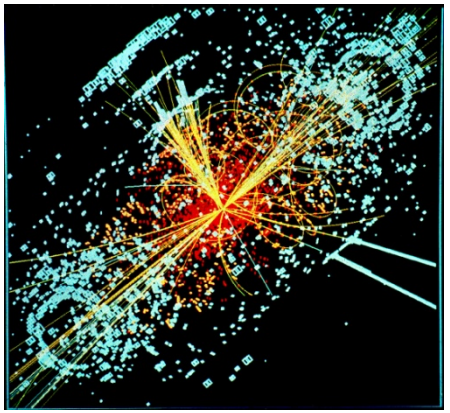
Value: data in use

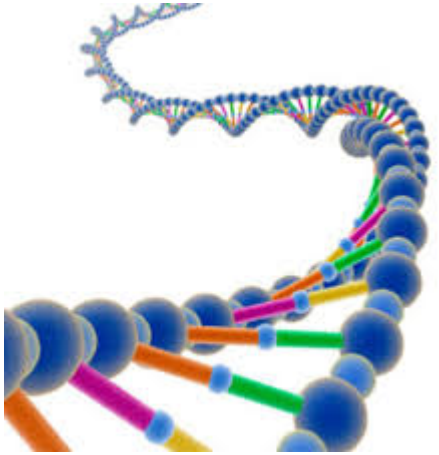
- Most important motivation for big data
- Big data may result in:
  - Better statistics/models
  - Novel insights
  - New opportunities for research and industry



Big Data: applications

- Science and research:
  - E.g. Physics, Bioinformatics, astronomy.
- Healthcare and public health:
  - Better personalized medicine
- Business and e-commerce
  - Personalized advertisement.
- Financial services
  - Insurance, banks.





MapReduce



- Based on **functional programming** (e.g. Lisp)
  - Operates on **<key, value>** pairs
    - Web-based example: key = URL ; value = webpage
    - Graph-based example: key = nodes ; value = adjacency list
  - Users specifies two functions:

$$\textbf{map:}\ (k_1, v_1) \rightarrow \text{list}[k_2, v_2]$$

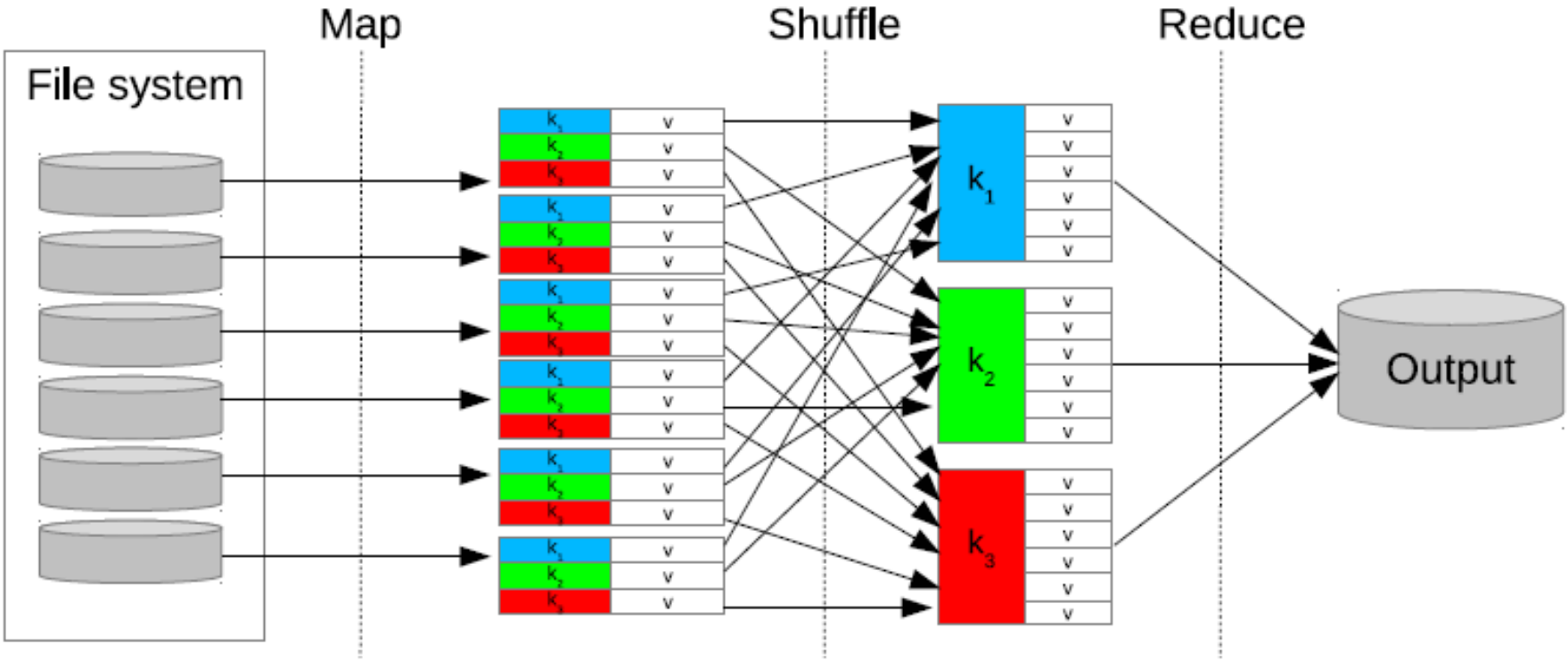
$$\textbf{reduce:}\ (k_2, \text{list}[v_2]) \rightarrow \text{list}[k_3, v_3]$$

  - Sorting of intermediate keys between map and reduce phase

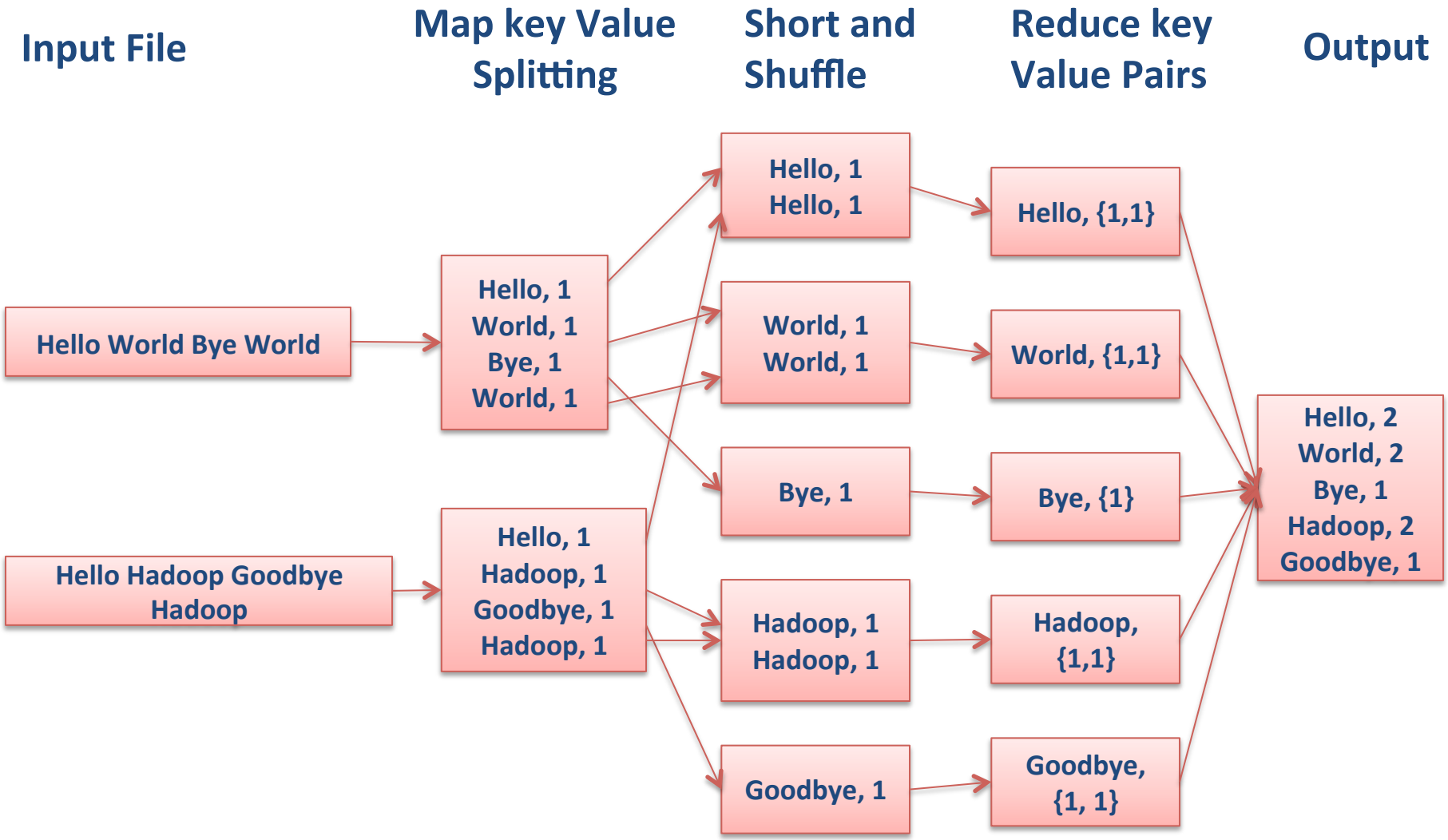
MapReduce



The dataflow in MapReduce is transparent to the programmers



WordCount using MapReduce



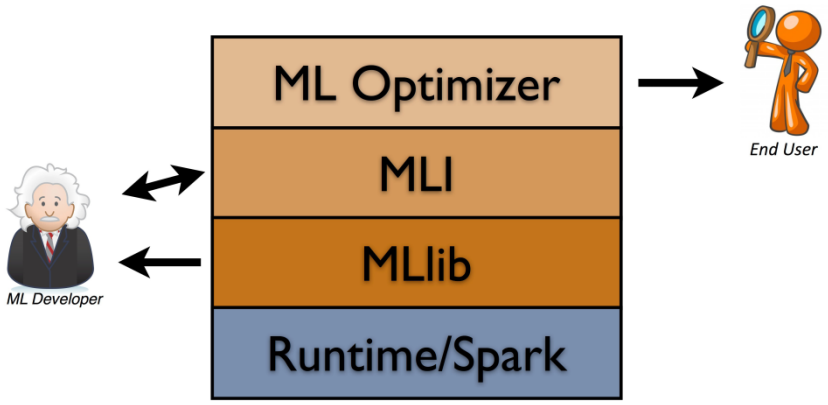
Machine learning for Big Data

- Data mining techniques have demonstrated to be very useful tools to extract new valuable knowledge from data.
- The knowledge extraction process from big data has become a very difficult task for most of the **classical** and **advanced** data mining tools.
- The **main challenges** are to deal with:
  - The increasing scale of data
    - at the level of **instances**
    - at the level of **features**
  - The **complexity** of the problem.
  - And many other points

Mllib: Spark Machine learning library

- **MLlib (2010):** is a Spark implementation of some common machine learning functionality, as well associated tests and data generators.

- Includes:
  - Binary classification (SVMs and
  - Logistic Regression)
  - Random Forest
  - Regression (Lasso, Ridge, etc.)
  - Clustering (K-Means)
  - Collaborative Filtering
  - Gradient Descent Optimization
  - Primitive



https://spark.apache.org/docs/latest/mllib-guide.html