Big Data processing: a framework suitable for Economists and Statisticians

Giuseppe Bruno[1], D. Condello[1] and A. Luciani[1]

[1]Economics and statistics Directorate, Bank of Italy;

Economic Research in High Performance Computing Environments,

October 9-10 2018 @ Kansas City, MO.

The views expressed in the presentation are the authors' only and do not imply those of the Bank of Italy.

Outline

Huge growth of Computing needs

The need of processing Big Data in short amount of time requires specialized computing platforms.

- Big Data applications are flooding all branches of scientific knowledge.
- Economic and statistical research needs to apply Big Data methodologies to improve on timeliness and accuracy.
- To purse these goals it is of paramount importance the selection of a suitable computing framework.

Huge growth of Computing needs

The need of processing Big Data in short amount of time requires specialized computing platforms.

- Big Data applications are flooding all branches of scientific knowledge.
- Economic and statistical research needs to apply Big Data methodologies to improve on timeliness and accuracy.
- To purse these goals it is of paramount importance the selection of a suitable computing framework.

Huge growth of Computing needs

The need of processing Big Data in short amount of time requires specialized computing platforms.

- Big Data applications are flooding all branches of scientific knowledge.
- Economic and statistical research needs to apply Big Data methodologies to improve on timeliness and accuracy.
- To purse these goals it is of paramount importance the selection of a suitable computing framework.

Huge growth of Computing needs

The need of processing Big Data in short amount of time requires specialized computing platforms.

- Big Data applications are flooding all branches of scientific knowledge.
- Economic and statistical research needs to apply Big Data methodologies to improve on timeliness and accuracy.
- To purse these goals it is of paramount importance the selection of a suitable computing framework.

Parallel computing framework is the avenue

The most relevant parallel computing frameworks:

- openMP which aims at multicore, single image architecture,
- Message Passing Interface (MPI), suitable for loosely coupled networks,
- Apache Hadoop which provides a parallel batch processing environment employing the Map Reduce paradigm,
- Apache Spark offers a very fast and general engine for large-scale iterative data processing.

Parallel computing framework is the avenue

The most relevant parallel computing frameworks:

- openMP which aims at multicore, single image architecture,
- Message Passing Interface (MPI), suitable for loosely coupled networks,
- Apache Hadoop which provides a parallel batch processing environment employing the Map Reduce paradigm,
- Apache Spark offers a very fast and general engine for large-scale iterative data processing.

Parallel computing framework is the avenue

The most relevant parallel computing frameworks:

- openMP which aims at multicore, single image architecture,
- Message Passing Interface (MPI), suitable for loosely coupled networks,
- Apache Hadoop which provides a parallel batch processing environment employing the Map Reduce paradigm,
- Apache Spark offers a very fast and general engine for large-scale iterative data processing.

Parallel computing framework is the avenue

The most relevant parallel computing frameworks:

- openMP which aims at multicore, single image architecture,
- Message Passing Interface (MPI), suitable for loosely coupled networks,
- Apache Hadoop which provides a parallel batch processing environment employing the Map Reduce paradigm,
- Apache Spark offers a very fast and general engine for large-scale iterative data processing.

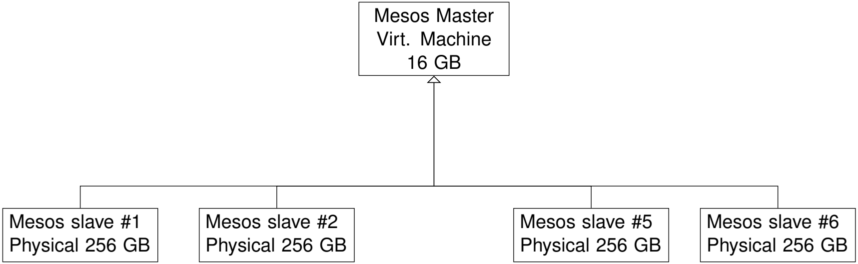
Outline

1. Motivation
2. Hardware Architecture
   - Client Server framework.
3. Software Architecture
   - Apache Spark Framework.
4. Econometric Benchmarks
   - Three Econometric Applications
   - *SparkR* vs *R*
   - *PySpark* vs *Python*
5. Concluding Remarks

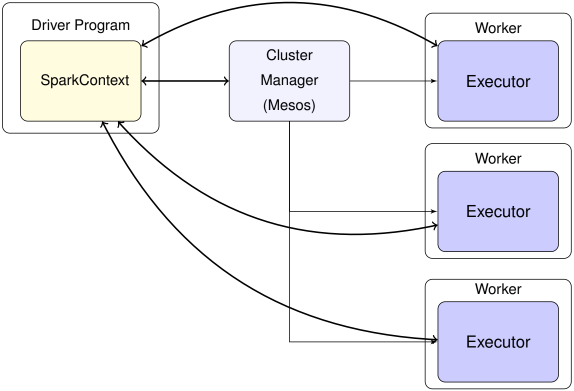
Hardware Architecture

For the worker nodes we employ a High Performance Computing Platform based on standard blade server HP-BL460c based on INTEL XEON 2630 with 40 cores in Hyperthreading.

Hardware Architecture

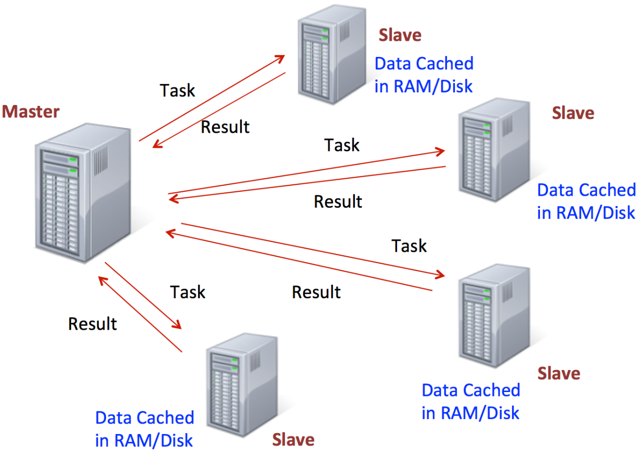


Apache Spark Architecture



Apache Spark Architecture

How does Spark execute a job



Sharing Computing resources

A software platform for efficient distribution of a set of limited resources:

- fair sharing of the resources amongst users;
- Providing resource guarantees to users (e.g. quota, priorities, etc.);
- Providing accurate resource accounting.

The platform shows the user a unified view of the state of services throughout the cluster.
Our choice is fallen on Mesos v. 1.6

Sharing Computing resources

A software platform for efficient distribution of a set of limited resources:

- fair sharing of the resources amongst users;
- Providing resource guarantees to users (e.g. quota, priorities, etc.);
- Providing accurate resource accounting.

The platform shows the user a unified view of the state of services throughout the cluster.
Our choice is fallen on Mesos v. 1.6

Sharing Computing resources

A software platform for efficient distribution of a set of limited resources:

- fair sharing of the resources amongst users;
- Providing resource guarantees to users (e.g. quota, priorities, etc.);
- Providing accurate resource accounting.

The platform shows the user a unified view of the state of services throughout the cluster.
Our choice is fallen on Mesos v. 1.6

Sharing Computing resources

A software platform for efficient distribution of a set of limited resources:

- fair sharing of the resources amongst users;
- Providing resource guarantees to users (e.g. quota, priorities, etc.);
- Providing accurate resource accounting.

The platform shows the user a unified view of the state of services throughout the cluster.
Our choice is fallen on Mesos v. 1.6

Sharing Computing resources

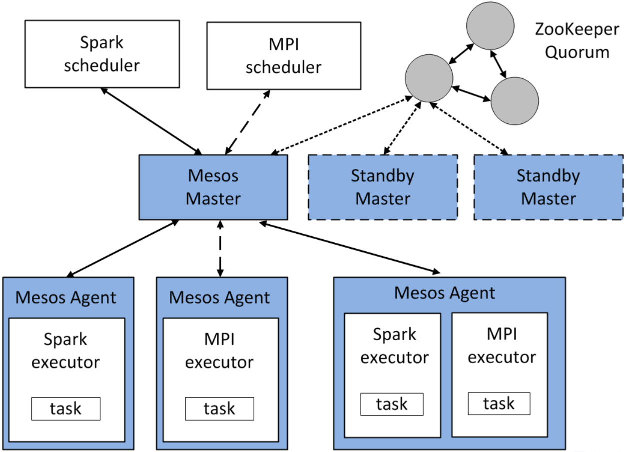
A software platform for efficient distribution of a set of limited resources:

- fair sharing of the resources amongst users;
- Providing resource guarantees to users (e.g. quota, priorities, etc.);
- Providing accurate resource accounting.

The platform shows the user a unified view of the state of services throughout the cluster.
Our choice is fallen on Mesos v. 1.6

Mesos platform

Here we see the general features of a Mesos cluster:



Outline

1. Motivation
2. Hardware Architecture
   - Client Server framework.
3. Software Architecture
   - Apache Spark Framework.
4. Econometric Benchmarks
   - Three Econometric Applications
   - *SparkR* vs *R*
   - *PySpark* vs *Python*
5. Concluding Remarks

The Spark cluster.

Apache Spark provides the whole software stack for cluster computing.

The main Spark features are:

- designed to efficiently deal with **iterative computation**,
- distributed and fault tolerant data abstraction (**Resilient Distributed Dataset**),
- **Lazy Evaluation** for reducing computation and preventing unnecessary I/O and memory usage.
- open source

The Spark cluster.

Apache Spark provides the whole software stack for cluster computing.
The main Spark features are:

- designed to efficiently deal with **iterative computation**,
- distributed and fault tolerant data abstraction (**Resilient Distributed Dataset**),
- **Lazy Evaluation** for reducing computation and preventing unnecessary I/O and memory usage.
- open source

The Spark cluster.

Apache Spark provides the whole software stack for cluster computing.
The main Spark features are:

- designed to efficiently deal with **iterative computation**,
- distributed and fault tolerant data abstraction (**Resilient Distributed Dataset**),
- **Lazy Evaluation** for reducing computation and preventing unnecessary I/O and memory usage.
- open source

The Spark cluster.

Apache Spark provides the whole software stack for cluster computing.
The main Spark features are:

- designed to efficiently deal with **iterative computation**,
- distributed and fault tolerant data abstraction (**Resilient Distributed Dataset**),
- **Lazy Evaluation** for reducing computation and preventing unnecessary I/O and memory usage.
- open source

The Spark cluster.

Apache Spark provides the whole software stack for cluster computing.
The main Spark features are:

- designed to efficiently deal with **iterative computation**,
- distributed and fault tolerant data abstraction (**Resilient Distributed Dataset**),
- **Lazy Evaluation** for reducing computation and preventing unnecessary I/O and memory usage.
- open source

The Spark Pillars.

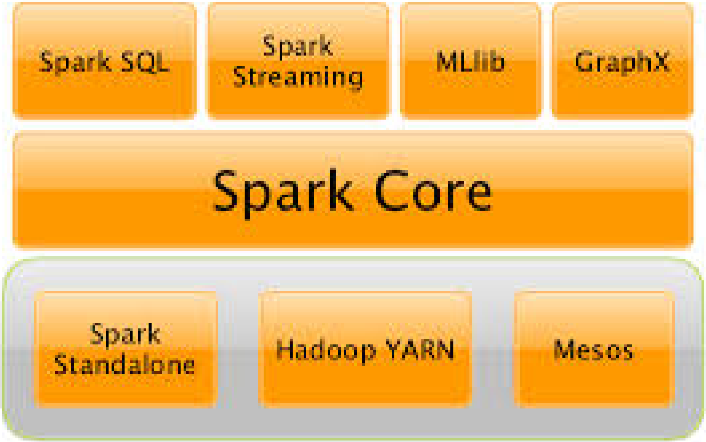
Apache Spark has a layered architecture where all the layers are loosely coupled and integrated with various libraries.
The Spark architecture relies on two main concepts:

1. Resilient Distributed Datasets (RDD);
2. Directed Acyclic Graph (DAG);

a RDD is collection of data that are split into partitions and can be stored in-memory on workers nodes of the cluster.
a DAG represents a sequence of computations performed on an RDD partition.

Apache Spark breakdown

Here are shown the Spark main software components:



Apache Spark API.

Apache Spark supplies an ample set of Application Programming Interfaces (API). Among them we have:

- Java,
- Python,
- R,
- Scala (which is the language used for Spark).

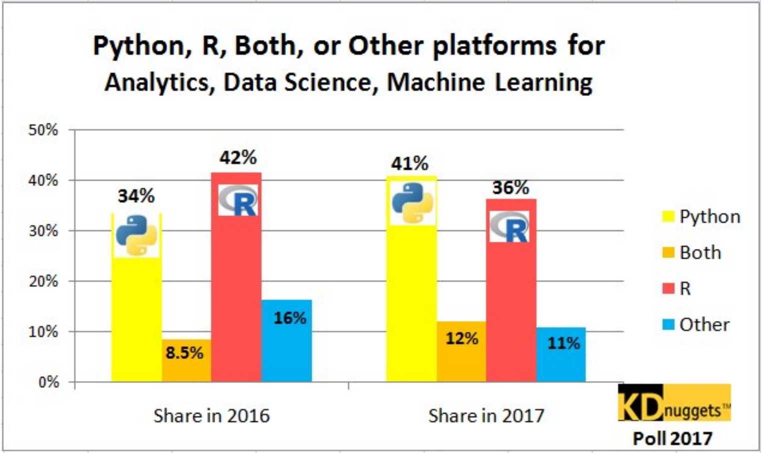
Apache Spark API.

Apache Spark supplies an ample set of Application Programming Interfaces (API). Among them we have:

- Java,
- Python,
- R,
- Scala (which is the language used for Spark).

Software for Data Science

Some of the software frameworks employed for data science:



Outline

1. Motivation
2. Hardware Architecture
   - Client Server framework.
3. Software Architecture
   - Apache Spark Framework.
4. Econometric Benchmarks
   - Three Econometric Applications
   - *SparkR* vs *R*
   - *PySpark* vs *Python*
5. Concluding Remarks

The Econometric Applications.

Our benchmarks are based on the following three examples:

1. Generalised Linear Models (GLM) with gaussian family;
2. GLM with binomial family;
3. Random Forests.

The Econometric Applications.

Our benchmarks are based on the following three examples:

1. Generalised Linear Models (GLM) with gaussian family;
2. GLM with binomial family;
3. Random Forests.

Generalised Linear Model.

The main elements of a GLM are:

1. a linear predictor:

$$y_i = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi} \tag{1}$$

2. a link function describing how the mean $E(y_i)$ depends on the linear predictor

$$\mathbb{E}(y_i) = \mu_i = g^{-1}(\mathbf{X}_i \cdot \beta) \tag{2}$$

Generalised Linear Model.

In case of a dependent variable binomially distributed we use

$$g(\mu_i) = logit(\mu_i) = \log\left(\frac{\mu_i}{1-\mu_i}\right) \tag{3}$$

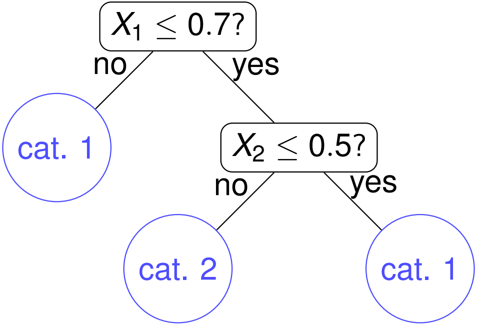
Random Forest.

A Random Forest is:

- 1) an ensemble classifier built with many decision trees;
- 2) a device suitable for classification and regression;
- 3) it generates accuracy and variable importance information.

Random Forest.

A simple decision tree is shown here:



A binary classification decision tree

Empirical Application.

The three used algorithms have been applied to a dataset with growing size.

file name	# obs.	size	seconds wc -l
data_1e+03.csv	1,000	90KB	≈ 0
data_1e+04.csv	10,000	898KB	≈ 0
data_1e+05.csv	100,000	8.8MB	≈ 0
data_1e+06.csv	10^6	88MB	≈ 0
data_1e+07.csv	10^7	877MB	.7
data_1e+08.csv	10^8	8.6GB	5
data_1e+09.csv	10^9	86GB	86
data_1e+10.csv	10^{10}	860GB	929

you can't use interactive editor with the largest files. ☹

Empirical Application.

The three used algorithms have been applied to a dataset with growing size.

file name	# obs.	size	seconds wc -l
data_1e+03.csv	1,000	90KB	≈ 0
data_1e+04.csv	10,000	898KB	≈ 0
data_1e+05.csv	100,000	8.8MB	≈ 0
data_1e+06.csv	10^6	88MB	≈ 0
data_1e+07.csv	10^7	877MB	.7
data_1e+08.csv	10^8	8.6GB	5
data_1e+09.csv	10^9	86GB	86
data_1e+10.csv	10^{10}	860GB	929

you can't use interactive editor with the largest files. ☹

Empirical Application.

The three used algorithms have been applied to a dataset with growing size.

file name	# obs.	size	seconds wc -l
data_1e+03.csv	1,000	90KB	≈ 0
data_1e+04.csv	10,000	898KB	≈ 0
data_1e+05.csv	100,000	8.8MB	≈ 0
data_1e+06.csv	10^6	88MB	≈ 0
data_1e+07.csv	10^7	877MB	.7
data_1e+08.csv	10^8	8.6GB	5
data_1e+09.csv	10^9	86GB	86
data_1e+10.csv	10^{10}	860GB	929

you can't use interactive editor with the largest files. ☹

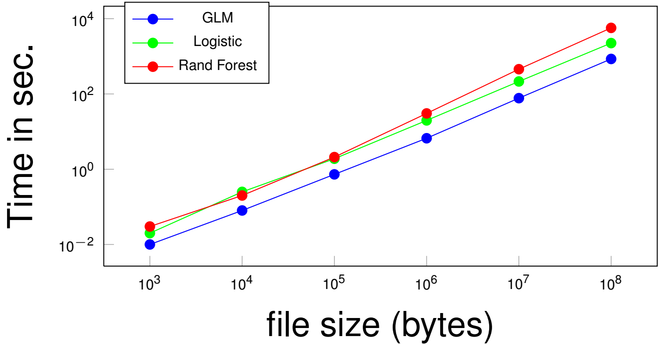
SparkR code

```
sparkR.session(
  master =                        "mesos://osi2-virt-516.utenze.bankit.it:5050",
  appName =                       "test_GLM_RF_SparkR",
  sparkConfig =                    list(
    spark.local.dir =                   "/tmp/work/wisi089",
    spark.serializer =                  "org.apache.spark.serializer.KryoSerializer",
    spark.local.dir =                   "/tmp/work/wisi089",
    spark.eventLog.enabled =            "true",
    spark.eventLog.dir =                "/tmp/work/wisi089",
    spark.executor.heartbeatInterval =  "20s",
    spark.driver.memory =               "10g",
    spark.executor.memory =             "220g",
    spark.driver.extraJavaOptions =     "-Djava.io.tmpdir=/tmp/work/wisi089",
    spark.executor.extraJavaOptions =   "-Djava.io.tmpdir=/tmp/work/wisi089",
    spark.cores.max =                   as.character(spark_cores),
    spark.executor.cores =              as.character(spark_executor_cores)  ))
    modelSparkLinearGLM <-spark.glm(main_data_spark, y ~ x1 + x2 + x3 + x4 + x5,
                                     family = "gaussian");
    fitted_modelSparkLinearGLM <- predict(modelSparkLinearGLM, main_data_spark);
```

Outline

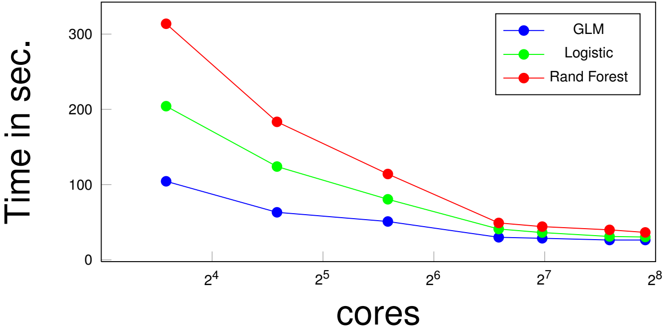
1. Motivation
2. Hardware Architecture
   - Client Server framework.
3. Software Architecture
   - Apache Spark Framework.
4. Econometric Benchmarks
   - Three Econometric Applications
   - *SparkR* vs *R*
   - *PySpark* vs *Python*
5. Concluding Remarks

Scalar Python results



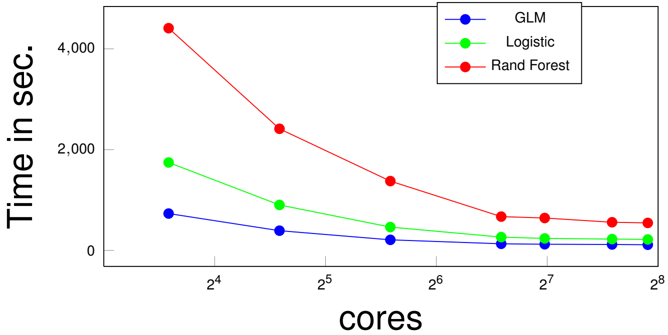
Scalar R example

SparkR 10 GB file.



SparkR with dataset size 10^8

SparkR 100 GB file.



SparkR with dataset size 10^9

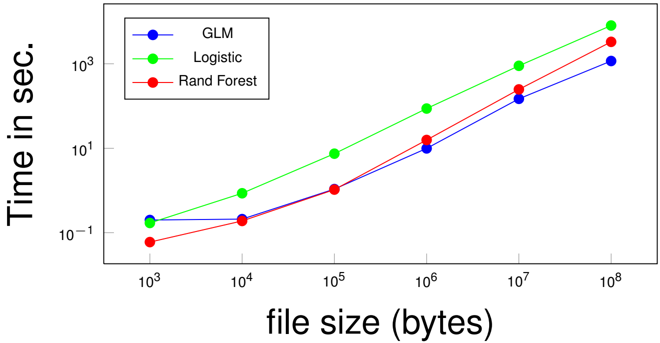
Pyspark code

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.regression import GeneralizedLinearRegression
from pyspark.ml.classification import RandomForestClassifier as RF
from pyspark.ml.feature import StringIndexer, VectorIndexer, VectorAssembler, SQLTransformer
    data = spark.read.csv(inputfile, schema=schema, header=True)
    data.rdd.getNumPartitions()
    cols_now = ['x1','x2','x3','x4','x5']
    assembler_features = VectorAssembler(inputCols=cols_now, outputCol='features')
    labelIndexer = StringIndexer(inputCol='y', outputCol="label")
    tmp = [assembler_features, labelIndexer]
    pipeline = Pipeline(stages=tmp)
    allData = pipeline.fit(data).transform(data)['label','features']
    allData.cache()
    glm = GeneralizedLinearRegression(family="gaussian", maxIter=1000)
    model = glm.fit(allData)
```

Outline

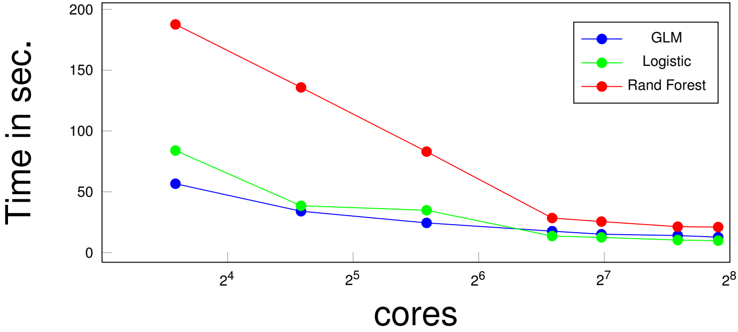
1. Motivation
2. Hardware Architecture
   - Client Server framework.
3. Software Architecture
   - Apache Spark Framework.
4. Econometric Benchmarks
   - Three Econometric Applications
   - *SparkR* vs *R*
   - *PySpark* vs *Python*
5. Concluding Remarks

Scalar Python results



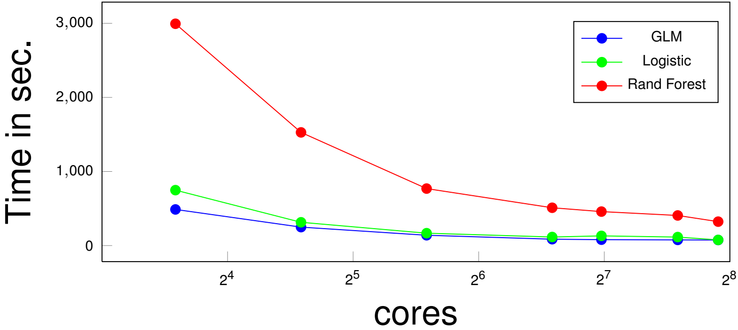
Scalar python example

PySpark 10 GByte file



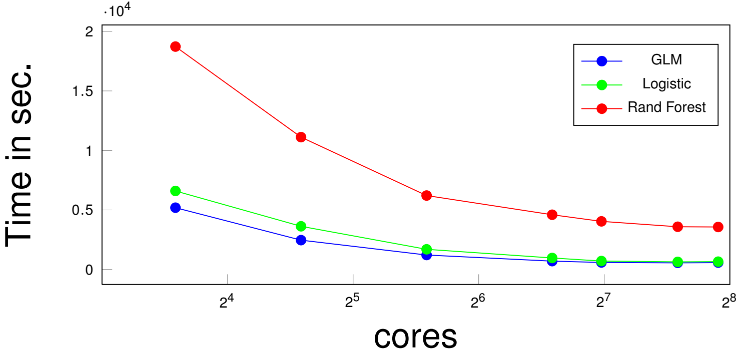
PySpark with dataset size 10^8

PySpark 100 GByte file



PySpark with dataset size 10^9

PySpark 1 TByte file



PySpark with dataset size 10^{10}

Concluding Remarks

- We have presented an easy to deploy computational platform for Big Data applications;
- we have shown the extensibility towards cluster programming for two popular software framework such as *R* and *Python*;
- we have pinned down a threshold above which it is convenient to shift towards cluster computing;
- In some instances *R* failed to solve the problem with dataset size around one billion.

For Further Reading

T. Drabas and D. Lee.
Learning PySpark.
Packt publishing, 2017.

S. Venkataram et al.
SparkR: Scaling R Programs with Spark.
International Conference on Management of Data, 2016.

M. Zaharia et al.
Spark: Cluster Computing with Working Sets.
Technical report, University of California Berkley, 2009.

Thank you for your attention.

Any questions?