# SystemML:
# Declarative Machine Learning

By Ahmed Tidjani and Lorenzo Toso

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

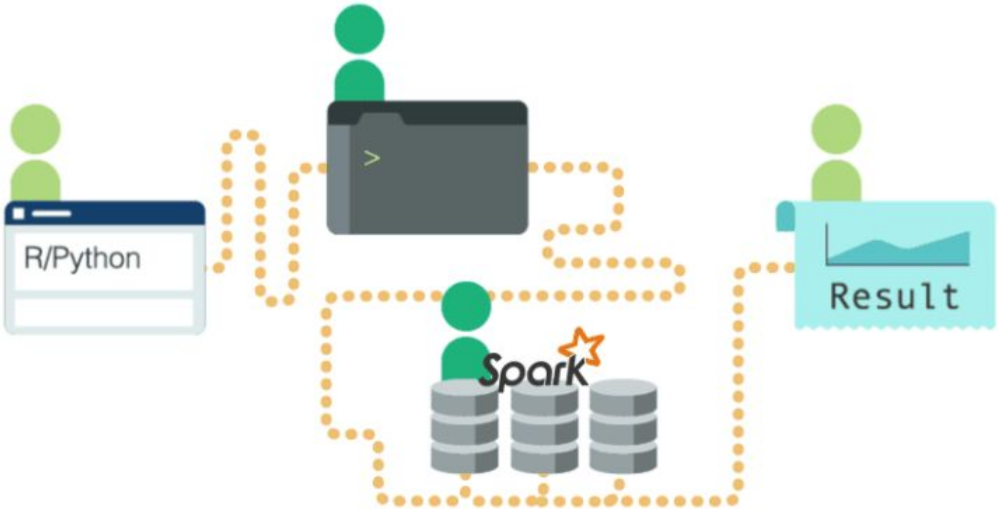
http://www.dima.tu-berlin.de/

# Agenda

- What is SystemML
- History
- Design Principles
- Technical Details
- Usage
- Comparison to similar frameworks
- Conclusion
- References

# What is SystemML

- Regular workflow:
  - Data scientist develops algorithm in R or Python
  - Tests it on small dataset
  - Systems engineer ports algorithm to Scala
  - ➔ Big Data operations are possible
  - Possibly multiple iterations required!




[1] systemml.apache.org

# What is SystemML

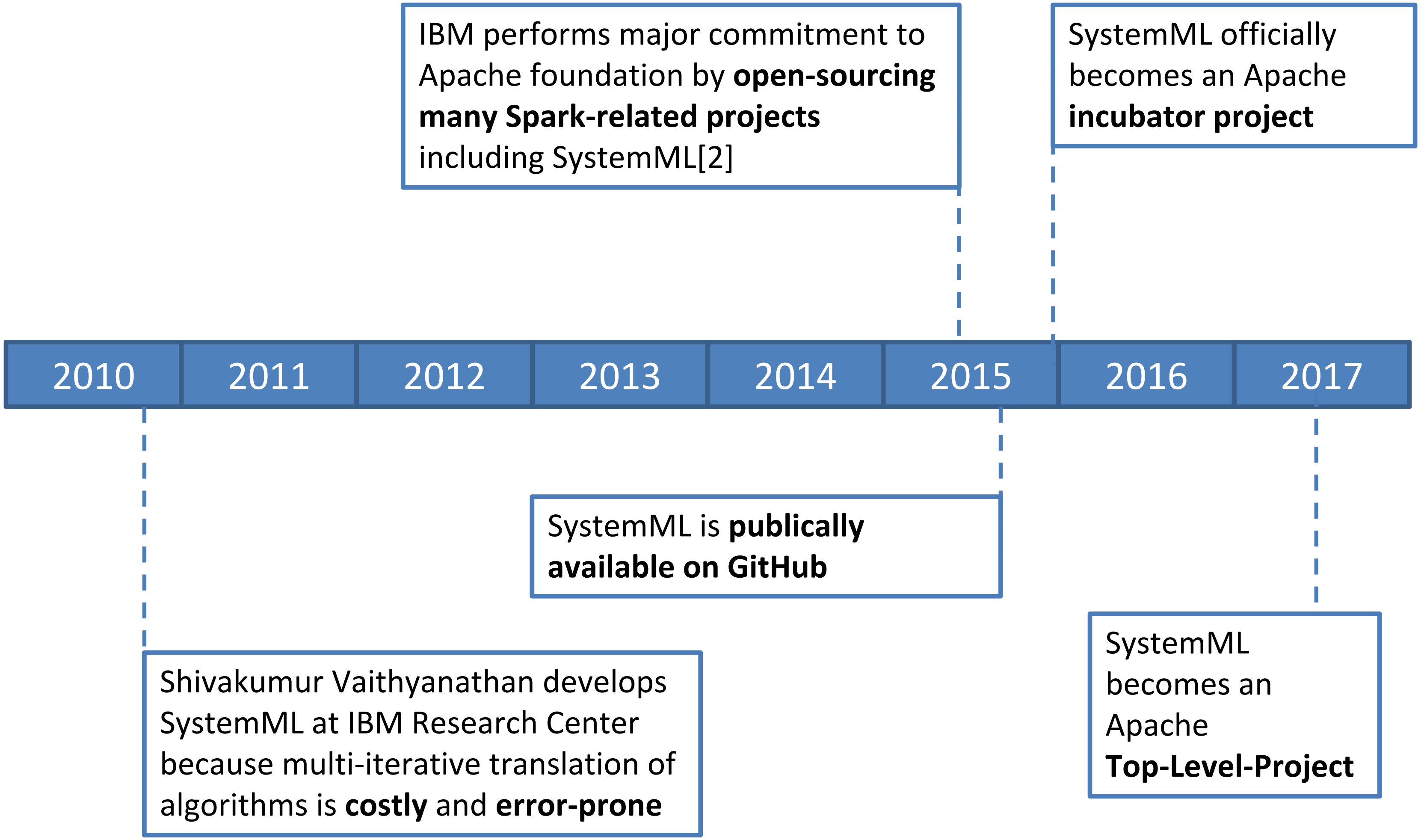
- SystemML:
  - Defines a Declarative High-Level Language for machine learning (DML)
    - Similar Syntax to Python or R
  - System running on Hadoop, Spark, Standalone or technically any other BigData-Framework
    - Flink integration is being worked on (First tests by Andreas Kunft)
  - Allows automatic scaling of machine learning algorithms


[1] systemml.apache.org

- SystemML workflow:
  - Data scientists declare algorithms like they are used to
  - Test them on a single machine with small datasets
  - Test them on a cluster with large datasets
  - Deploy them for real applications
  
  ⇒ Eliminates need error-prone translation of algorithms


[1] systemml.apache.org

# History


IBM performs major commitment to Apache foundation by **open-sourcing many Spark-related projects** including SystemML[2]
SystemML officially becomes an Apache **incubator project**
2010
2011
2012
2013
2014
2015
2016
2017
SystemML is **publically available on GitHub**
Shivakumur Vaithyanathan develops SystemML at IBM Research Center because multi-iterative translation of algorithms is **costly** and **error-prone**
SystemML becomes an Apache **Top-Level-Project**

# Design Principles: DML

- Declarative Machine Learning (DML)

- Syntax similar to R (DML) or Python (PyDML)
- Translation from Python+Numpy is straightforward
- Contains a rich set of functions:
  - matrix operations
  - statistical methods
  - machine learning algorithms

- Control sequences like „parfor"-loops for task-parallel operations

- DML code example

```
script = """
    # add constant feature to X to model intercepts
    X = cbind(X, matrix(1, rows=nrow(X), cols=1))
    max_iter = 100
    w = matrix(0, rows=ncol(X), cols=1)
    for(i in 1:max_iter){
        XtX = t(X) %*% X
        dw = XtX %*%w - t(X) %*% y
        alpha = -(t(dw) %*% dw) / (t(dw) %*% XtX %*% dw)
        w = w + dw*alpha
    }
    bias = as.scalar(w[nrow(w),1])
    w = w[1:nrow(w)-1,]
"""
```

# Design Principles: Abstraction

- SystemML aims to hide all the technical details from the user
- Transparency regarding the underlying framework
  - Spark, MapReduce, Standalone
  - Optimizations for individual frameworks
    - E.g. Spark specific optimization of caching algorithms
- Optimizes execution parameters based on data and cluster characteristics
  - E.g. decides whether to use a cluster at all or if things should be done on a single machine
  - Tries to minimize shuffling of data

# Technical Details: Binary Block Matrices

- All input data is converted into binary block matrices
  - Other systems use a similar approach [5]

- Map pair of indices to Matrix-Content
  - (long|long) -> Matrix-Content
  - Allows arbitrary distribution

- All blocks are squared and of fixed size
  - default: 1000x1000

  ⇒ 8MB fits in regular L3 Cache

- Blocks decide independently whether they are
  - Empty (Simple flag is stored)
  - Dense (All values are stored)
  - Sparse (Store either row or column index - Value pairs)
  - Ultra-Sparse (Store row-column-value pairs)

# Technical Details: Caching in Spark

- Caching is completely hidden from the user
  - → Needs to be done well automatically

- Uses storage level MEMORY_AND_DISK
  - → RDDs are stored as deserialized Objects in the JVM on Memory/Disk
  - → Except for „Ultra-Sparse-Matrices". These use MEMORY_AND_DISK_SER

- Uses Spark Cache and Checkpoints
  - → Cache: Materializes RDDs and keeps them in memory
    - Including Lineage for fault tolerance
  - → Checkpoint: Saves RDDs to HDFS
    - without lineage
    - HDFS is fault tolerant by replication

# Technical Details: Caching in Spark

- Performs fixed set of caching optimizations
  - Checkpoint injection (brings matrices into read-optimized form)
    - Before loops
    - After persistent reads
  - Deferred optimization of parfor-blocks
  - Removing of redundant checkpoints

# Technical Details: Repartitioning in Spark

- Shuffling of data is very expensive
  - → Needs to be avoided

- Use of operations that avoid shuffling is highly preferred
  - "Partition-Preserving operations" are used
    - Key must not change
    - More restrictive use of API

- Introduce explicit repartitioning to avoid unnecessary reshuffling

- Use Partition-Exploiting operations
  - E.g. Transposing on a single node

# Technical Details: Execution Type

- Basic Spark execution type
  - Simplest approach for choosing execution type
  - Worst case memory estimate for an operation is greater than the available memory on a single node

    ⇒ Run operation on spark

- Transitive Spark execution type
  - Data would fit in memory of a single machine
  - Data transfer is reduced by scheduling Spark-Job
  - Example:
    - Sum over many values
    - Every node can compute the sum of the values it already has in memory
    - Only intermediate results need to be transferred

# Technical Details: Optimizer

- Optimization of High-Level-Operators (HOPs)
  - Algebraic rewrites
  - Operator order
    - E.g. Matrix multiplication of sparse matrices
  - Compute memory estimates

- Compute Low-Level-Operators (LOPs)
  - Based on memory estimates
  - Based on cluster characteristics
  - Have corresponding runtime implementations

# Technical Details: Dynamic recompilation

- SystemML performs dynamic recompilation
  - Between major program blocks
  - When the optimizer explicitly injects recompilation points

- Allows to adapt runtime execution plan to previously unknown data characteristics

- Problem Spark Lazy Evaluation:
  - Some necessary values may not be computed yet

  ⇒ Characteristics of operations are used to determine output sizes and data types when possible

# Usage

- From a Scala-Program

```
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
val ml = new MLContext(sc)
```

- From a Spark-Shell

```
scala> import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext._

scala> import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.ScriptFactory._

scala> val ml = new MLContext(sc)

Welcome to Apache SystemML!

ml: org.apache.sysml.api.mlcontext.MLContext = org.apache.sysml.api.mlcontext.MLContext@12139db0
```

# Usage

- From Python
- Or a Jupyter-Notebook

```
import systemml as sml
import numpy as np
m1 = sml.matrix(np.ones((3,3)) + 2)
m2 = sml.matrix(np.ones((3,3)) + 3)
m2 = m1 * (m2 + m1)
m4 = 1.0 - m2
m4.sum(axis=1).toNumPy()
```

- As Standalone using DML

```
$ ./runStandaloneSystemML.sh scripts/utils/sample.dml -nvargs X=data/haberman.data sv=data/perc.csv O=data/hab
erman.part ofmt="csv"
```

# Usage: Install and instructions

- Download Spark binaries
  - https://spark.apache.org/downloads.html
- Download SystemML sources
  - https://systemml.apache.org/download
- Build sources using mvn install -DskipTests
- Set SPARK_HOME environment variable
- Set SYSTEMML_HOME environment variable

- Launch PySpark in Jupyter notebook with SystemML

- PYSPARK_DRIVER_PYTHON=jupyter
- PYSPARK_DRIVER_PYTHON_OPTS="notebook"
- $ pyspark --master local[*] --driver-memory 3G --driver-class-path SystemML.jar --jars SystemML.jar

# Usage: Linear regression with SystemML

- PySpark integration in Jupyter notebook
- Linear regression algorithm implemented in SystemML DML
- Hybrid execution: Spark + CP
- Bike-sharing data set

| | | | | | |
|---|---|---|---|---|---|
| **Data Set Characteristics:** | Univariate | **Number of Instances:** | 17389 | **Area:** | Social |
| **Attribute Characteristics:** | Integer, Real | **Number of Attributes:** | 16 | **Date Donated** | 2013-12-20 |
| **Associated Tasks:** | Regression | **Missing Values?** | N/A | **Number of Web Hits:** | 143207 |

http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

- Initializing SystemMLContext with SparkContext

```
from systemml import MLContext, dml, dmlFromResource

ml = MLContext(sc)

print ("Spark Version:" + sc.version)
print ("SystemML Version:" + ml.version())
print ("SystemML Built-Time:"+ ml.buildTime())
```

# Usage: Linear regression with SystemML

- Executing LinearRegDS.dml

```
scriptName = "LinearRegDS.dml"
dml_script = dmlFromResource(scriptName)

prog = dml_script.input(X=train_data[:,:-3], y=train_data[:,-
1].reshape(-1,1)).input('$icpt',1.0).output('beta_out')

w = ml.execute(prog).get('beta_out')
w = w.toNumPy()
bias=w[-1]
w = w[:-1,:]
test_y = test_data[:,:-3].dot(w)+bias
```

# Usage: Linear regression with SystemML

- Extract from LinerRegDS.dml

```
# BEGIN THE DIRECT SOLVE ALGORITHM (EXTERNAL CALL)

A = t(X) %*% X;
b = t(X) %*% y;
if (intercept_status == 2) {
    A = t(diag (scale_X) %*% A + shift_X %*% A [m_ext, ]);
    A =   diag (scale_X) %*% A + shift_X %*% A [m_ext, ];
    b =   diag (scale_X) %*% b + shift_X %*% b [m_ext, ];
}
A = A + diag (lambda);

print ("Calling the Direct Solver...");

beta_unscaled = solve (A, b);
```

- Output of LinearRegDS.dml script

/bin/bash

/bin/bash

```
[I 17:08:11.644 NotebookApp] Saving file at /Untitled1-Copy1.ipynb
[I 17:10:11.644 NotebookApp] Saving file at /Untitled1-Copy1.ipynb
BEGIN LINEAR REGRESSION SCRIPT
Reading X and Y...
Calling the Direct Solver...
Computing the statistics...
AVG_TOT_Y,191.1264125534282
STDEV_TOT_Y,182.0341548848353
AVG_RES_Y,3.009216507337137E-13
STDEV_RES_Y,142.3986352096892
DISPERSION,20277.371309582133
R2,0.38849414143527206
ADJUSTED_R2,0.38806415958230267
R2_NOBIAS,0.38849414143527206
ADJUSTED_R2_NOBIAS,0.38806415958230267
Writing the output matrix...
END LINEAR REGRESSION SCRIPT
```

# Usage: Linear regression with SystemML

- SystemML stats: ml = ml.setStatistics(True)

```
Writing the output matrix...
END LINEAR REGRESSION SCRIPT
SystemML Statistics:
Total elapsed time:              0.181 sec.
Total compilation time:          0.000 sec.
Total execution time:            0.181 sec.
Number of compiled Spark inst:   2.
Number of executed Spark inst:   2.
Cache hits (Mem, WB, FS, HDFS):  26/0/0/0.
Cache writes (WB, FS, HDFS):     9/0/0.
Cache times (ACQr/m, RLS, EXP):  0.000/0.002/0.001/0.000 sec.
HOP DAGs recompiled (PRED, SB):  0/0.
HOP DAGs recompile time:         0.000 sec.
Spark ctx create time (lazy):    0.000 sec.
Spark trans counts (par,bc,col):0/0/0.
Spark trans times (par,bc,col):  0.000/0.000/0.000 secs.
Total JIT compile time:          0.084 sec.
Total JVM GC count:              19.
Total JVM GC time:               0.584 sec.
Heavy hitter instructions (name, time, count):
-- 1)   tsmm    0.006 sec       3
-- 2)   ba+*    0.004 sec       2
-- 3)   append  0.003 sec       9
```

# Comparison to similar systems

- Property 1: **Independence of Data structures**
  - All data types are given as abstractions with no access to physical data
- Property 2: **Independence of Data Flow Properties**
  - User has no control over partitioning, caching, blocking etc.
- Property 3: **Analysis-Centric Operation Primitives**
  - The framework implements basic functions for statistics and ML
- Property 4: **Known Semantics of Operation Primitives**
  - Knowledge of semantics (associativity, sparse-safeness, etc)

# Comparison to similar systems

- Property 5: **Implementation-Agnostic Operations**
  - The specification of algorithms is independent of the underlying runtime
- Property 6: **Well-Defined Plan Optimizazion Objective**
  - Objective for execution plan optimization is clearly specified
- Property 7: **Implementation-Agnostic Results:**
  - Results of algorithms are equivalent independently of the used runtime
- Property 8: **Deterministic Results:**
  - Results for multiple runs are equivalent

# Comparison to similar systems

**Table 2: Classification of Existing Systems wrt Declarative ML Algorithms (Type 1).**
(P1 Indep. Data Structures, P2 Indep. Data Flow Properties, P3 Analysis-Centric Operations, P4 Known Operations, P5 Impl.-Agnostic Operations, P6 Well-Def. Optim. Objective, P7 Impl.-Agnostic Results, P8 Deterministic Results)

| Name | Dist. | Basic Properties | | | | | | | | Type | Objective |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | | |
| RIOT [43] | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 1 | min runtime |
| OptiML [35] | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 1 | min runtime |
| SystemML [7, 22] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 1 | min runtime s.t. memory constraints |
| Mahout Samsara [15] | ✓ | | | ✓ | ✓ | | ✓ | ✓ | ✓ | N/A | min runtime |
| Distributed R [39] | ✓ | | | ✓ | | | ✓ | ✓ | ✓ | N/A | min runtime |
| Cumulon [24, 25] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | N/A | min costs s.t. runtime constraints |
| DMac [40] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | N/A | min runtime s.t. memory constraints |
| TensorFlow [1] | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | N/A | min runtime s.t. resource constraints |
| SciDB [9, 34] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 1 | min runtime |
| SimSQL [11] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 1 | min runtime |
| ScalOps [8] | ✓ | | ✓ | | | | ✓ | ✓ | ✓ | N/A | min runtime |
| Tupleware [13] | ✓ | | ✓ | | | | ✓ | ✓ | ✓ | N/A | min runtime |
| Emma [4] | ✓ | | ✓ | | | | ✓ | ✓ | ✓ | N/A | min runtime |

[6] Declarative Machine Learning - A Classification of Basic Properties and Types

# Conclusion

- SystemML performs a great job at hiding technical details from the user
  - User only concentrates on developing the algorithm
  - Algorithms is automatically scaled

- The provided DML is very simple and straightforward if you know R or Python

- The framework is relatively new and currently unproven
  - Small community
  - Only academic uses known

- Can heuristics really perform sufficient parameter optimization?
  - Not well tested
  - No representative benchmarks

# References


- [1] systemml.apache.org
- [2] Ghoting, Amol, et al. "SystemML: Declarative machine learning on MapReduce." *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011.
- [3] Matthias Boehm, Michael W. Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Arvind C. Surve, and Shirish Tatikonda. 2016. SystemML: declarative machine learning on spark. *Proc. VLDB Endow.* 9, 13 (September 2016), 1425-1436. DOI: http://dx.doi.org/10.14778/3007263.3007279
- [4] Matthias Boehm, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen, Yuanyuan Tian, Douglas R. Burdick, and Shivakumar Vaithyanathan. 2014. Hybrid parallelization strategies for large-scale machine learning in SystemML. *Proc. VLDB Endow.* 7, 7 (March 2014), 553-564. DOI=http://dx.doi.org/10.14778/2732286.2732292
- [5] Bosagh Zadeh, Reza, et al. "Matrix computations and optimization in apache spark." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.
- [6] Boehm, Matthias, et al. "Declarative Machine Learning-A Classification of Basic Properties and Types." arXiv preprint arXiv:1605.05826 (2016).
- [7] https://github.com/apache/systemml