Distributed and Native Optimizations for Machine Learning Workloads

Suneel Marthi
June 12, 2017
Berlin Buzzwords, Berlin, Germany

$WhoAmI

Suneel Marthi

Senior Principal Engineer, Office of CTO, Red Hat Inc.
Member of Apache Software Foundation
PMC member on Apache Mahout, Apache OpenNLP, Apache Streams

@suneelmarthi

Agenda

- What is Apache Mahout?
- Mahout Samsara: Declarative, R-like DSL for Matrix Math
- Distributed SSVD
- EigenFaces
- Integration with Apache Zeppelin
- Solve on CPU, GPU or JVM
- What’s Coming Next?

Intro to Apache Mahout

Apache Mahout is an environment for creating scalable, performant, machine-learning applications

Apache Mahout provides:

- Mathematically Expressive Scala DSL (Samsara)
- A collection of pre-canned Math and Statistics algorithms
- Interchangeable Distributed Engines (Spark, Flink or use your own)
- Interchangeable "Native Solvers" (JVM, CPU, GPU, CUDA, or write your own!)

Recent work on the Project

- v 0.13.1 - In the Works - CUDA Solvers, Scala 2.11 support
- v 0.13.0 - Apr 2017 - GPU/CPU Solvers, algo framework
- v 0.12.2 - Nov 2016 - Apache Zeppelin integration for visualization
- v 0.12.0 - Apr 2016 - Apache Flink Backend support
- Feb 2016- New Mahout Book - ‘Apache Mahout: Beyond MapReduce’ by Dmitriy Lyubimov and Andrew Palumbo - Feb 2016
- v 0.10.0 - Apr 2015 - Mahout-Samsara vector-math DSL

Mahout Samsara

Mahout Samsara

Mahout-Samsara is an easy-to-use domain-specific language (DSL) for large-scale machine learning on distributed systems like Apache Spark/Flink

- Uses **Scala** as programming/scripting environment
- **System-agnostic, R-like DSL**:

$$G = BB^T - C - C^T + \xi^T \xi s_q^T s_q$$

```
val G = B %*% B.t - C - C.t + (ksi dot ksi) * (s_q cross s_q)
```

- **algebraic expression optimizer** for distributed linear algebra
  - provides a translation layer to distributed engines
  - Support for Spark RDDs and Flink DataSets

Samsara Basics

Data Types

- Scalar real values
- In-memory vectors
  - dense
  - 2 types of sparse
- In-memory matrices
  - sparse and dense
  - a number of specialized matrices

```
val x = 2.367

val v = dvec(1, 0, 5)

val w =
  svec((0 -> 1)::(2 -> 5)::Nil)

val A = dense((1, 0, 5),
              (2, 1, 4),
              (4, 3, 1))
```

Data Types (contd)

- Distributed Row Matrices (DRM)
  - huge matrix, partitioned by rows
  - lives in the main memory of the cluster
  - provides small set of parallelized operations
  - lazily evaluated operation execution

```
val drmA = drmDfsRead(...)
```

Features (1)

- Matrix, vector, scalar operators: in-memory, distributed

```
drmA %*% drmB
A %*% x
A.t %*% drmB
A * B
```

- Slicing operators

```
A(5 until 20, 3 until 40)
A(5, ::); A(5, 5)
x(a to b)
```

- Assignments (in-memory only)

```
A(5, ::) := x
A *= B
A -=: B; 1 /:= x
```

- Vector-specific

```
x dot y; x cross y
```

mahout

Features (2)

- Summaries

```
A.nrow; x.length; A.colSums; B.rowMeans;
A.norm
```

- Solving linear systems

```
val x = solve(A, b)
```

- In-memory decompositions

```
val (inMemQ, inMemR) = qr(inMemM)
val ch = chol(inMemM)
val (inMemV, d) = eigen(inMemM)
val (inMemU, inMemV, s) = svd(inMemM)
```

Features (3)

- Distributed decompositions

```
val (drmQ, inMemR) = thinQR(drmA)
val (drmU, drmV, s) =
  dssvd(drmA, k = 50, q = 1)
```

- Caching of DRMs

```
val drmA_cached = drmA.checkpoint()
drmA_cached.uncache()
```

Unary Operators

In-Core

```
mahout> val mxA = dense((1,2,3),(3,4,5))
mxA: org.apache.mahout.math.DenseMatrix =
{
 0 =>	{0:1.0,1:2.0,2:3.0}
 1 =>	{0:3.0,1:4.0,2:5.0}
}

mahout> mlog(mxA)
res2: org.apache.mahout.math.Matrix =
{
 0 =>	{1:0.6931471805599453,2:1.0986122886681098}
 1 =>	{0:1.0986122886681098,1:1.3862943611198906,2:1.6094379124341003}
}

mahout> msignum(mxA)
res3: org.apache.mahout.math.Matrix =
{
 0 =>	{0:1.0,1:1.0,2:1.0}
 1 =>	{0:1.0,1:1.0,2:1.0}}
```

In-Core (Contd)

```
// add some negative numbers in
mahout> val mxB = dense((-1,2,-3),(-3,4,-5))
mxB: org.apache.mahout.math.DenseMatrix =
{
 0 =>    {0:-1.0,1:2.0,2:-3.0}
 1 =>    {0:-3.0,1:4.0,2:-5.0}
}

mahout> msignum(mxB)
res7: org.apache.mahout.math.Matrix =
{
 0 =>    {0:-1.0,1:1.0,2:-1.0}
 1 =>    {0:-1.0,1:1.0,2:-1.0}
}
```

Distributed Row Matrix (DRM)

```
mahout> val drmA = drmParallelize(mxA)

mahout> dlog(drmA).collect

res10: org.apache.mahout.math.Matrix =
{
 0 =>    {1:0.6931471805599453,2:1.0986122886681098}
 1 =>    {0:1.0986122886681098,1:1.3862943611198906,2:1.6094379124341003}
}
```

Example Algebraic Optimization

Runtime & Optimization

- Execution is deferred, user composes logical operators

```
val drmC = drmA.t %*%
drmA
```

- Computational actions implicitly trigger optimization (= selection of physical plan) and execution

```
drmI.dfsWrite(path)
val inMemV =(drmU %*% drmM).collect
```

- Optimization factors: size of operands, orientation of operands, partitioning, sharing of computational paths

- e. g.: matrix multiplication:
  - 5 physical operators for drmA %*% drmB
  - 2 operators for drmA %*% inMemA
  - 1 operator for drm A %*% x
  - 1 operator for x %*% drmA

Runtime & Optimization (contd.)

- Common computational paths ((A + B)' %*% (A + B) -> self-square(A + B)
- Tracking identically partitioned sets (“zip” vs. “join” judgements)
- Tracking data deficiencies (missing or duplicate rows)
  - automatic fixes
- Algebraic cost reducing rewrites (Expr t) t -> Expr
- Unary operator fusion dlog(X * X) -> elementwise-apply [x => log(x * x)]
- Elements of cost based optimizations (“slim” vs. “wide”)
- Product parallelism decisions
- Explicit and implicit optimization barriers
  - control the scope of optimization

Optimization Example

- Computation of A^TA in example

```
val C = A.t %*% A
```

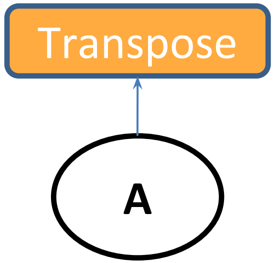
Optimization Example

- Computation of $A^T A$ in example

```
val C = A.t %*% A
```

- **Naïve execution**

1st pass: transpose A
(requires repartitioning of A)



Optimization Example

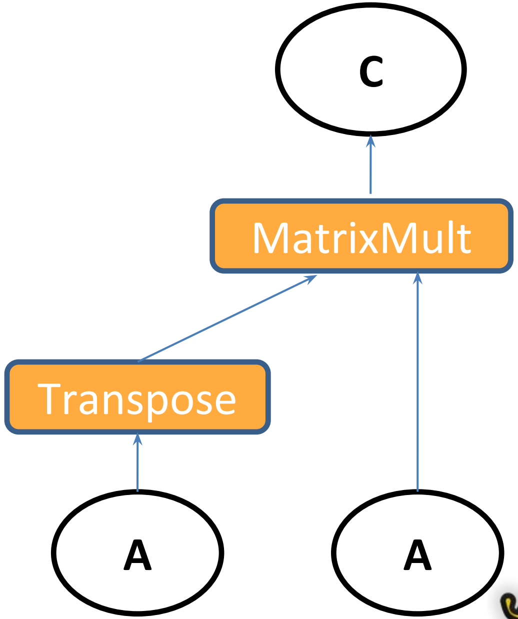
- Computation of A^TA in example

```
val C = A.t %*% A
```

- Naïve execution

1st pass: transpose A
(requires repartitioning of A)

2nd pass: multiply result with A
(expensive, potentially requires repartitioning again)



Optimization Example

Computation of A^TA in example

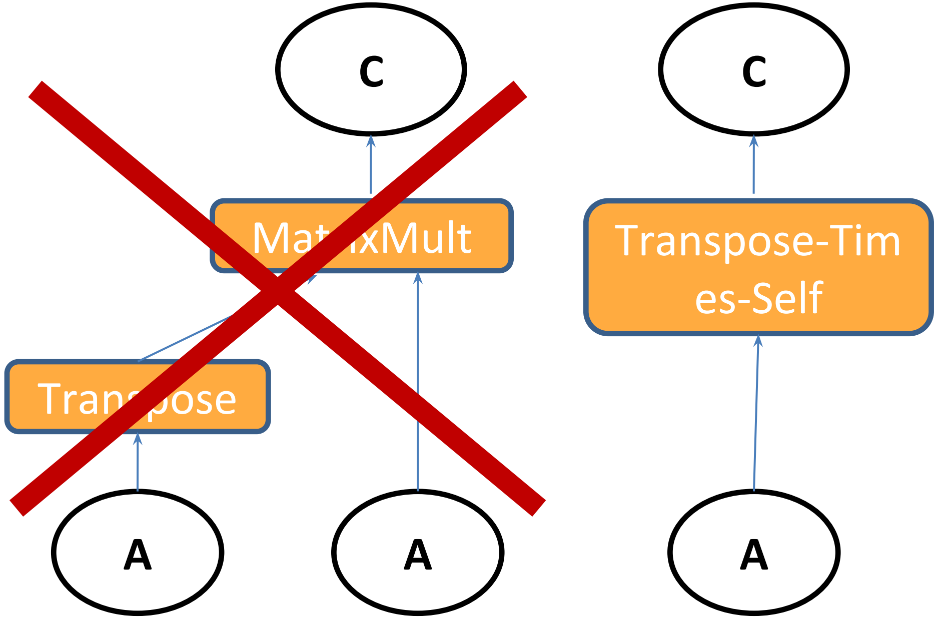
```
val C = A.t %*% A
```

~~Naïve execution~~
~~1^{st} pass: transpose A~~
~~(requires repartitioning of A)~~

~~2^{nd} pass: multiply result with A~~
(expensive, potentially requires repartitioning again)

Logical optimization

Optimizer rewrites plan to use specialized logical operator for *Transpose-Times-Self* matrix multiplication



Transpose-Times-Self

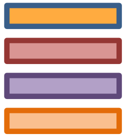
- Mahout Samsara computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{T}$$

Transpose-Times-Self

- Samsara computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^T$$

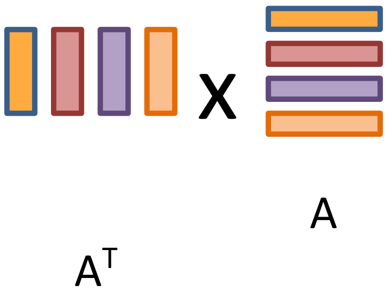


A

Transpose-Times-Self

- Samsara computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

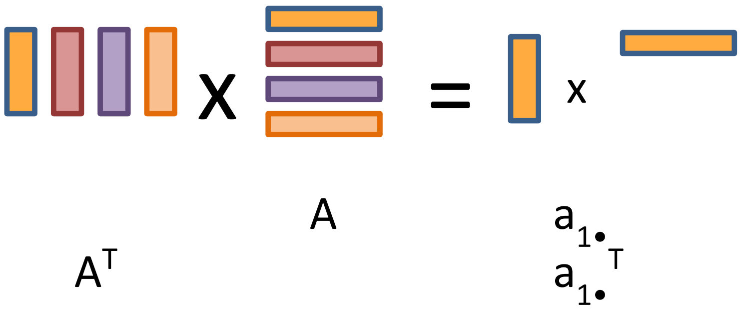
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^T$$



Transpose-Times-Self

- Samsara computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

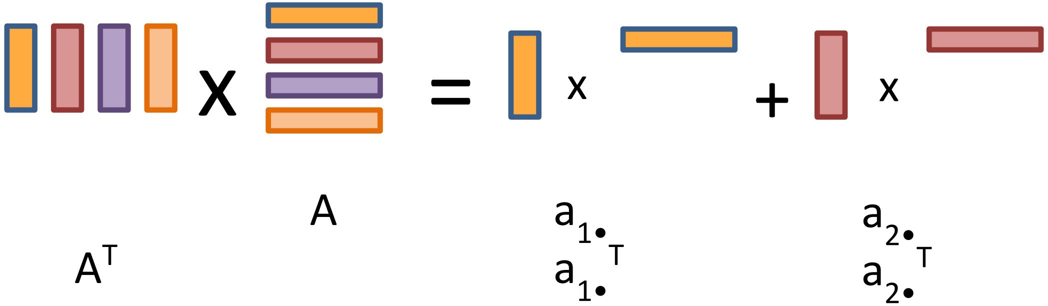
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{T}$$



Transpose-Times-Self

- Samsara computes $A^T A$ via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

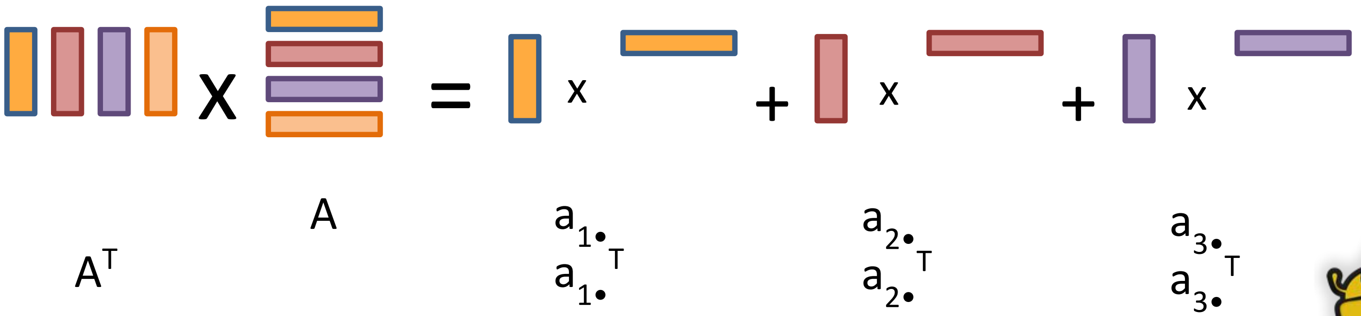
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^T$$



Transpose-Times-Self

- Mahout computes $A^T A$ via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

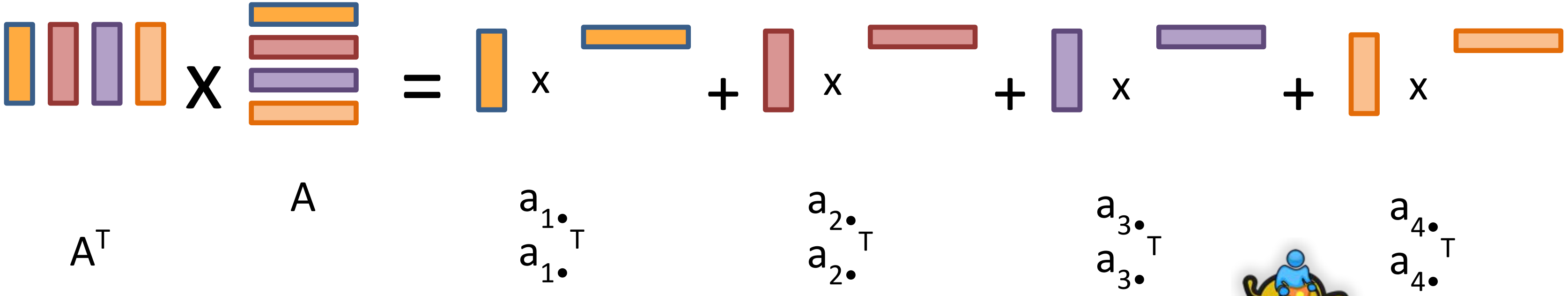
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^T$$



Transpose-Times-Self

- Samsara computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

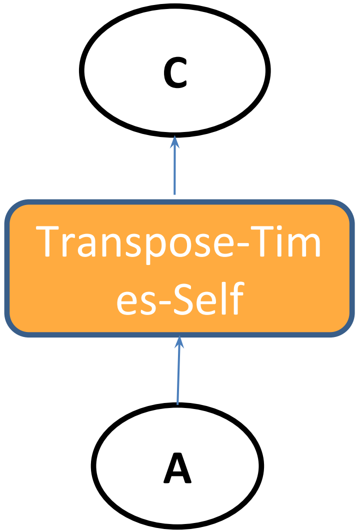
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^T$$



Physical operators for the distributed computation of $A^T A$

Physical operators for Transpose-Times-Self

- Two physical operators (concrete implementations) available for *Transpose-Times-Self* operation
  - standard operator ***AtA***
  - operator ***AtA_slim**,* specialized implementation for tall & skinny matrices



- Optimizer must choose
  - currently: depends on user-defined threshold for number of columns
  - ideally: cost based decision, dependent on estimates of intermediate result sizes

Algorithm for AtA, AtB, etc
Correlated-Cross-Occurrence

- Major extension of Cooccurrence Recommender r = hAtA to include arbitrary Cross-Occurrences with an LLR correlation test

$$r = h_a AtA + h_b AtB + h_b AtC \ldots.$$

- A = conversion history for all users, B, C, … = interaction history for all users
- h_a = a single user's history of conversion as column vector, h_b = a single user's history of another interaction...
- r = recommended items from A, **even if there is no h_a** and this is new!
- Every cross-occurrence is found with AtA operators and tested for correlation with LLR.

Backend Agnostic Programming

```
%flinkMahout

// Imports and creating the distributed context, similar but not exactly the same ////////////////////////////////////
///////
import org.apache.flink.api.scala._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.flinkbindings._
import org.apache.mahout.math._
import scalabindings._
import RLikeOps._


implicit val ctx = new FlinkDistributedContext(benv)

// CODE IS EXACTLY THE SAME FROM HERE ON - R-Like DSL ///////////////////////////////////////////////////////////////
////////////////

val drmData = drmParallelize(dense(
  (2, 2, 10.5, 10, 29.509541),  // Apple Cinnamon Cheerios
  (1, 2, 12,   12, 18.042851),  // Cap'n'Crunch
  (1, 1, 12,   13, 22.736446),  // Cocoa Puffs
  (2, 1, 11,   13, 32.207582),  // Froot Loops
  (1, 2, 12,   11, 21.871292),  // Honey Graham Ohs
  (2, 1, 16,   8,  36.187559),  // Wheaties Honey Gold
  (6, 2, 17,   1,  50.764999),  // Cheerios
  (3, 2, 13,   7,  40.400208),  // Clusters
  (3, 3, 13,   4,  45.811716)), numPartitions = 2)

drmData.collect(::, 0 until 4)

val drmX = drmData(::, 0 until 4)
val y = drmData.collect(::, 4)
val drmXtX = drmX.t %*% drmX
val drmXty = drmX.t %*% y


val XtX = drmXtX.collect
val Xty = drmXty.collect(::, 0)
val beta = solve(XtX, Xty)
```

```
%sparkMahout

// Imports and creating the distributed context, similar but not exactly the same ////////////////////////////////////
///////

import org.apache.mahout.math._
import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.sparkbindings._

implicit val sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = sc2sdc(sc)


// CODE IS EXACTLY THE SAME FROM HERE ON - R-Like DSL ///////////////////////////////////////////////////////////////
////////////////

val drmData = drmParallelize(dense(
  (2, 2, 10.5, 10, 29.509541),  // Apple Cinnamon Cheerios
  (1, 2, 12,   12, 18.042851),  // Cap'n'Crunch
  (1, 1, 12,   13, 22.736446),  // Cocoa Puffs
  (2, 1, 11,   13, 32.207582),  // Froot Loops
  (1, 2, 12,   11, 21.871292),  // Honey Graham Ohs
  (2, 1, 16,   8,  36.187559),  // Wheaties Honey Gold
  (6, 2, 17,   1,  50.764999),  // Cheerios
  (3, 2, 13,   7,  40.400208),  // Clusters
  (3, 3, 13,   4,  45.811716)), numPartitions = 2)

drmData.collect(::, 0 until 4)

val drmX = drmData(::, 0 until 4)
val y = drmData.collect(::, 4)
val drmXtX = drmX.t %*% drmX
val drmXty = drmX.t %*% y


val XtX = drmXtX.collect
val Xty = drmXty.collect(::, 0)
val beta = solve(XtX, Xty)
```

Distributed SSVD

Stochastic SVD (SSVD)

Given a large matrix A, compute reduced k-rank SVD such that A = UEV

U = Left Singular Vectors
V = Right Singular Vectors
E = Diagonal Matrix with decaying singular values

Singular Vectors sorted in decreasing order of the corresponding singular values

See Nathan Halko's Dissertation - https://amath.colorado.edu/faculty/martinss/Pubs/2012_halko_dissertation.pdf

Distributed SSVD (DSSVD) inputs

```
mahout> val (drmU, drmV, s) = dssvd(drmA, k = 90, p = 15, q = 0)

drmA = Input DRM

k = requested decompostion rank

p = oversampling parameter (default = 15)

q = number of power iterations to run (q >= 0)

    Typical q values are 0 or 1.


Note: k, p must satisfy the rqmt that k + p <= rank(A)
      Upper bound of rank(A) = min(drmA.nrows, drmA.ncols)
```

EigenFaces

- Set of Eigenvectors used for Human Face Recognition (https://en.wikipedia.org/wiki/Eigenface)
- Smaller set of images to represent original training images by dimensionality reduction
- Small set of images data to represent many different images
- Trained images are represented as collection of weights
- Classify new images by Nearest-neighbor computation

Faces in the Wild Dataset

FINISHED

Images alligned by Funnelings, credit Learning to Allign Faces from Scratch Gary B. Huang and Marwan Mattar and Honglak Lee and Erik Learned-Miller

Webpage of dataset

Dataset statistics:
- 13,233 Images
- Each image centered on face, 250x250 pixels
- side on disk: 162M decompressed

Took 2 sec. Last updated by anonymous at November 14 2016, 9:15:19 AM.

Download Faces Data

FINISHED

It is worth taking a moment to set `shell.command.timeout.millisecs` in the `sh` interpreter to `600000`

Took 0 sec. Last updated by anonymous at November 13 2016, 9:34:13 PM. (outdated)

Download Faces Data

FINISHED

```
%sh

mkdir -p zeppelin-0.7.0-SNAPSHOT/webapps/webapp/eigenfaces/input

wget http://vis-www.cs.umass.edu/lfw/lfw-deepfunneled.tgz
tar -xzf lfw-deepfunneled.tgz
```

Put Faces Data in HDFS

FINISHED

```
%sh
hdfs dfs -put /home/guest/lfw-deepfunneled /tmp/lfw-deepfunneled
```

Took 2 min 17 sec. Last updated by anonymous at November 10 2016, 10:15:25 AM. (outdated)

Faces in the Wild Dataset Sample

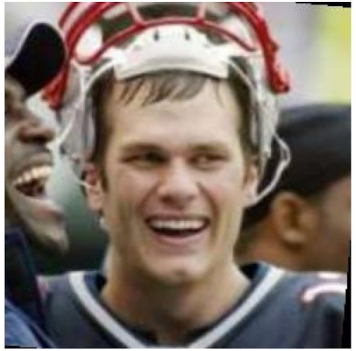
FINISHED

eigenfaces/Joel_Gallen_0001.jpg

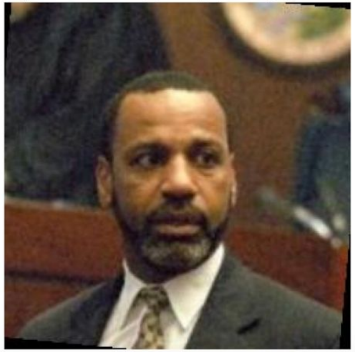
eigenfaces/Vagit_Alekperov_0001.jpg

eigenfaces/John_Darby_0001.jpg

eigenfaces/Kevin_Marshall_0001.jpg



eigenfaces/Tom_Brady_0001.jpg



Add Image Processing Dependencies

FINISHED

```
%sparkMahout.dep

z.load("com.sksamuel.scrimage:scrimage-core_2.10:2.1.0")
z.load("com.sksamuel.scrimage:scrimage-io-extra_2.10:2.1.0")
z.load("com.sksamuel.scrimage:scrimage-filters_2.10:2.1.0")

// add EXPERIMENTAL mahout algos
// https://github.com/rawkintrevo/mahout/tree/mahout-1856/algos
z.load("/home/guest/mahout-algos_2.10-0.12.3-SNAPSHOT.jar")
```

```
DepInterpreter(%dep) deprecated. Load dependency through GUI interpreter menu instead.
DepInterpreter(%dep) deprecated. Load dependency through GUI interpreter menu instead.
DepInterpreter(%dep) deprecated. Load dependency through GUI interpreter menu instead.
DepInterpreter(%dep) deprecated. Load dependency through GUI interpreter menu instead.
res3: org.apache.zeppelin.dep.Dependency = org.apache.zeppelin.dep.Dependency@469e2d6c
```

Took 19 sec. Last updated by anonymous at November 13 2016, 9:55:35 PM.

Setup Mahout Context

FINISHED

```
%sparkMahout.spark

import org.apache.mahout.math._
import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.sparkbindings._

@transient implicit val sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = sc2sdc(sc)
```

```
import org.apache.mahout.math._
```

Create DRM of Vectorized Images

```
%sparkMahout.spark

import com.sksamuel.scrimage._
import com.sksamuel.scrimage.filter.GrayscaleFilter

val imagesRDD:DrmRdd[Int] = sc.binaryFiles("/tmp/lfw-deepfunneled/*/*")
                 .map(o => new DenseVector( Image.apply(o._2.toArray)
                                  .filter(GrayscaleFilter)
                                  .pixels
                                  .map(p => p.toInt.toDouble / 10000000)
                             ) )
                  .zipWithIndex
                  .map(o => (o._2.toInt, o._1))


val imagesDRM = drmWrap(rdd= imagesRDD).par(min = 500).checkpoint()

println(s"Dataset: ${imagesDRM.nrow} images, ${imagesDRM.ncol} pixels per image")
```

```
import com.sksamuel.scrimage._
import com.sksamuel.scrimage.filter.GrayscaleFilter
imagesRDD: org.apache.mahout.sparkbindings.DrmRdd[Int] = MapPartitionsRDD[3] at map at <console>:64
imagesDRM: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.sparkbindings.drm.CheckpointedDrmS
Dataset: 13233 images, 62500 pixels per image
```

Took 35 min 10 sec. Last updated by anonymous at November 13 2016, 10:33:53 PM.

Subtract Means Column-wise

```
%sparkMahout.spark

import org.apache.mahout.algos.transformer.SubtractMean

// Subtract Mean transforms each row by subtracting the column mean
val smTransformer = new SubtractMean()

smTransformer.fit(imagesDRM)  // calculuates the column mean
val smImages = smTransformer.transform(imagesDRM) // return new DRM of subtracted means

smImages.checkpoint()
```

```
import org.apache.mahout.algos.transformer.SubtractMean
smTransformer: org.apache.mahout.algos.transformer.SubtractMean = org.apache.mahout.algos.transformer.S
smImages: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.sparkbindings.drm.Chec
res2: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.sparkbindings.drm.Checkpointe
```

Took 42 min 22 sec. Last updated by anonymous at November 13 2016, 10:41:57 PM.

Mahout Distributed SSVD to get Eigenfaces

```
%sparkMahout.spark

import org.apache.mahout.math._
import decompositions._
import drm._


val(drmU, drmV, s) = dssvd(smImages, k= 20, p= 15, q = 0)
```

```
import org.apache.mahout.math._
import decompositions._
import drm._
drmU: org.apache.mahout.math.drm.DrmLike[Int] =
OpMapBlock(OpTimesRightMatrix(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpa
 0 =>   {0:0.4235322348619768,1:0.15420057798333225,2:-0.0375073015538210 5,3:0.1415
2886395335518,9:-0.027526698307401825,10:0.25020315771392226,11:-0.062243817521616 8
26167920899813,17:0.201120285694 6398,18:0.11507386925972764,19:-0.1924203457607603 4
 1 =>   {0:0.204576635605 4534,1:-0.01756047401031135 2,2:0.15352410435078737,3:-0.117
```

Took 1 hrs 1 min 48 sec. Last updated by anonymous at November 13 2016, 11:35:42 PM.

Write Eigenfaces to Disk

```
%sparkMahout.spark

import java.io.File
import javax.imageio.ImageIO

val sampleImagePath = "/home/guest/lfw-deepfunneled/Aaron_Eckhart/Aaron_Eckhart_0001.jpg"
val sampleImage = ImageIO.read(new File(sampleImagePath))
val w = sampleImage.getWidth
val h = sampleImage.getHeight

val eigenFaces = drmV.t.collect(::,::)
val colMeans = smImages.colMeans

for (i <- 0 until 20){
    val v = (eigenFaces(i, ::) + colMeans) * 10000000
    val output = new Array[com.sksamuel.scrimage.Pixel](v.size)
    for (i <- 0 until v.size) {
        output(i) = Pixel(v.get(i).toInt)
    }
    val image = Image(w, h, output)
    image.output(new File(s"/home/guest/zeppelin-0.7.0-SNAPSHOT/webapps/webapp/eigenfaces/${i}.png"))
}
```

A little python to create an HTML table of our Eigenfaces

FINISHED

Plotting in Mahout - Apache Zeppelin

```
val maxSample = 1000 // Note there is a setting for this in Zepplein, that is by default 1000 (max.results).
val drm1000Sampled = drmSampleKRows(drmPoints, maxSample, replacement = false)
val drm5000Sampled = drmSampleKRows(drmPoints, 5 * maxSample, replacement = false)
val drm10000Sampled = drmSampleKRows(drmPoints, 10 * maxSample, replacement = false)
```

```
maxSample: Int = 1000
drm1000Sampled: org.apache.mahout.math.Matrix =
{
 0 =>   {0:2.5711533907282864,1:3.3985775949011963,2:1.828454662423897 6E-5}
 1 =>   {0:-0.5849668540131455,1:0.008078750346618811,2:0.13397645857511886}
 2 =>   {0:-0.532276603852006 3,1:0.2591761531093102,2:0.1334801365959136 8}
 3 =>   {0:1.4728661457904864,1:-2.858981586079829,2:9.026288048664696E-4}
 4 =>   {0:-1.0669267542210157,1:-0.9975250697214353,2:0.05459559407365087 4}
 5 =>   {0:2.190375803711219,1:0.10769738990367872,2:0.01443187324573094}
 6 =>   {0:-2.32280101452071 2,1:3.216968865426014 5,2:6.064867349000867E-5}
 7 =>   {0:0.9564289803558209,1:0.020272550397296755,2:0.10089785428577039}
 8 =>   {0:-3.154759699459544 4,1:2.877856483019137 5,2:1.742637051335269 4E-5}
 9 =>   {0:-1.8361745541885914,1:1.9873637471279224,2:0.004089036909448981}
... }
drm5000Sampled: org.apache.mahout.math.Matrix =
{
 0 =>   {0:-1.55955426611664,1:-2.87769727917133 3,2:7.45822304552873 1E-4}
 1 =>   {0:-2.5076350828808582,1:1.3857096994895128,2:0.0026186879929642574}
```

Took 8 seconds. Last updated by anonymous at time Jun 1, 2016 4:23:29 PM. (outdated)

```
z.put("mahout1000Table", matrix2table(drm1000Sampled))
z.put("mahout5000Table", matrix2table(drm5000Sampled))
z.put("mahout10000Table", matrix2table(drm10000Sampled))
```

Took 22 seconds. Last updated by anonymous at time Jun 1, 2016 3:51:02 PM.

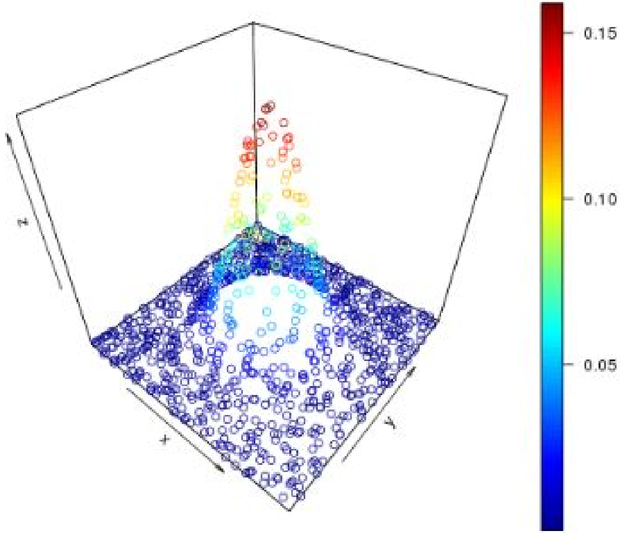
```
%r
# Sometimes this works, sometimes not.  If not open up R and install this package the old fasioned way...
install.packages("plot3D", repos='http://cran.us.r-project.org')
```

```
The downloaded source packages are in
    '/tmp/RtmpKRPnfz/downloaded_packages'
```

Took 7 seconds. Last updated by anonymous at time Jun 1, 2016 3:52:16 PM. (outdated)

```
%r {"imageWidth": "400px"}
library("plot3D")
dfStr = z.get("mahout1000Table")
data <- read.table(text= dfStr, sep="\t", header=TRUE)
colnames(data)
points3D(data$col1, data$col2, data$col3)
```

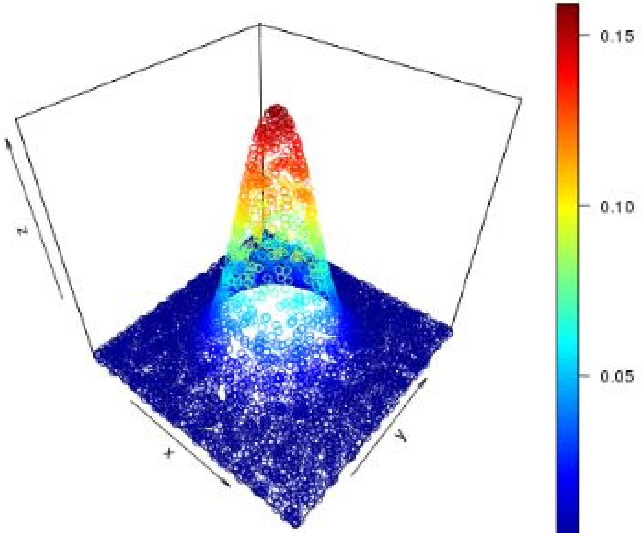
"col1 "col2 "col3



Took 1 seconds. Last updated by anonymous at time Jun 1, 2016 3:52:52 PM. (outdated)

```
%r {"imageWidth": "400px"}
library("plot3D")
dfStr = z.get("mahout5000Table")
data <- read.table(text= dfStr, sep="\t", header=TRUE)
colnames(data)
points3D(data$col1, data$col2, data$col3)
```

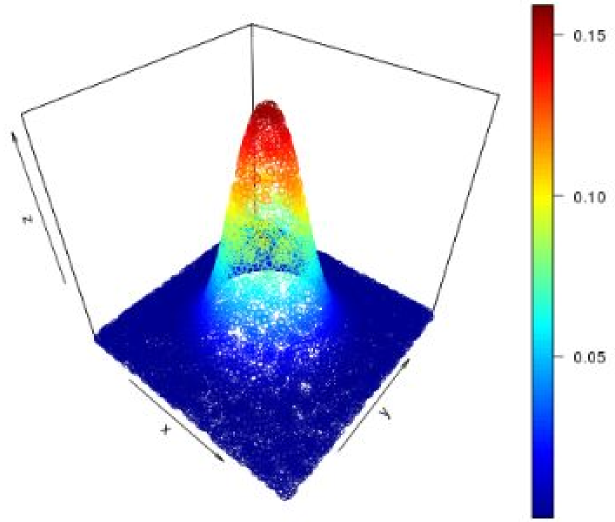
“col1 “col2 “col3



Took 1 seconds. Last updated by anonymous at time Jun 1, 2016 3:53:14 PM. (outdated)

```
%r {"imageWidth": "400px"}
library("plot3D")
dfStr = z.get("mahout10000Table")
data <- read.table(text= dfStr, sep="\t", header=TRUE)
colnames(data)
points3D(data$col1, data$col2, data$col3)
```

"col1 "col2 "col3



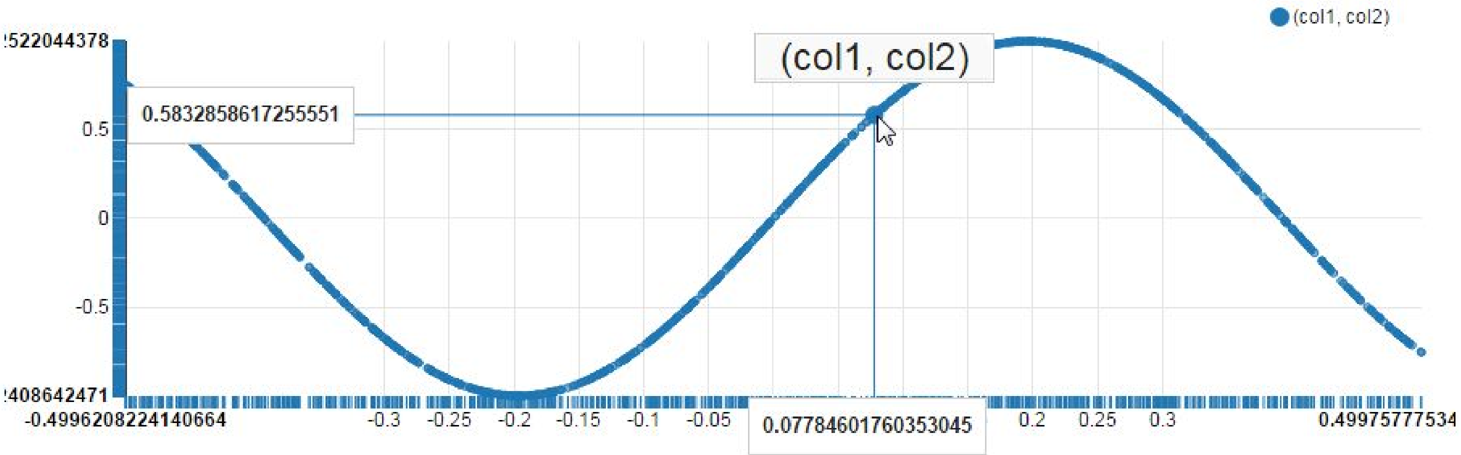
Took 1 seconds. Last updated by anonymous at time Jun 1, 2016 3:53:27 PM. (outdated)

Tablify Matrix Using Zepplin + Angular

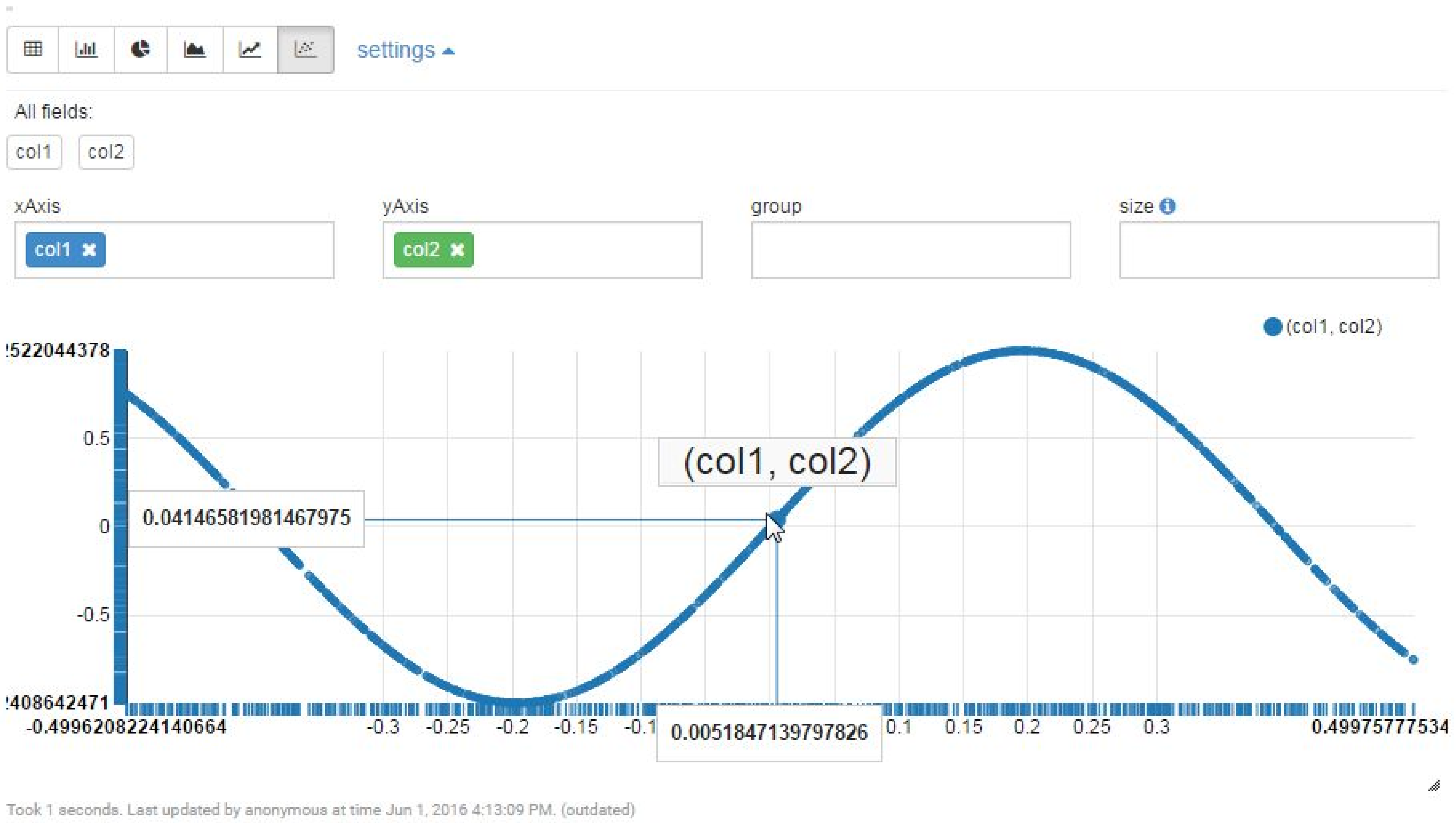
FINISHED

```
%spark

var str = ""
//println("matrix collected")
for (i <- 0 until mPlotMatrix.numRows()) {
    //println("i: " + i.toString)
    for (j <- 0 until mPlotMatrix.numCols()) {
        str += mPlotMatrix(i, j)
        if (j <= mPlotMatrix.numCols() - 2) { str += "\t" }
    }
    str += "\n"
  }

val tableStr = "col1\tcol2\n"+ str
println("%table\n"+tableStr)
```

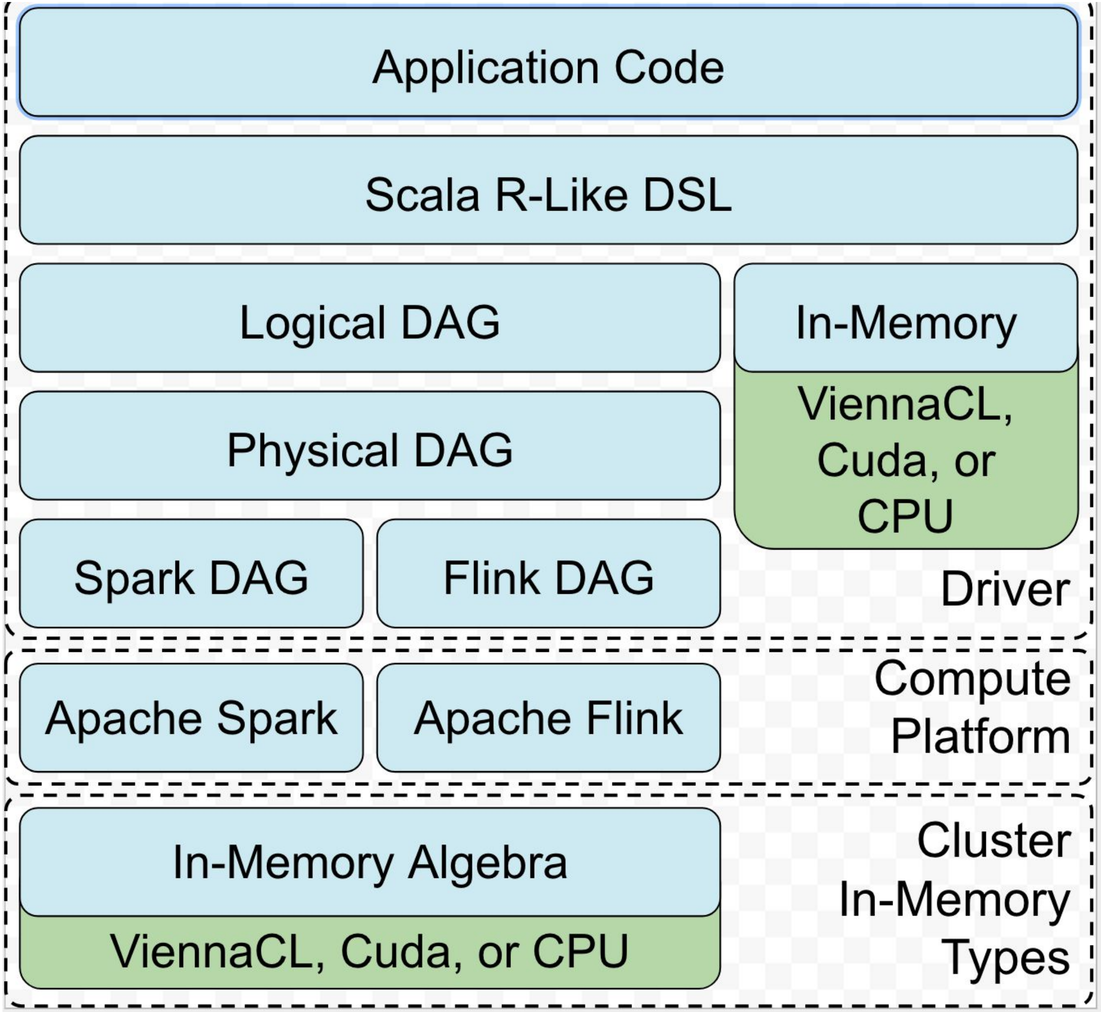
settings



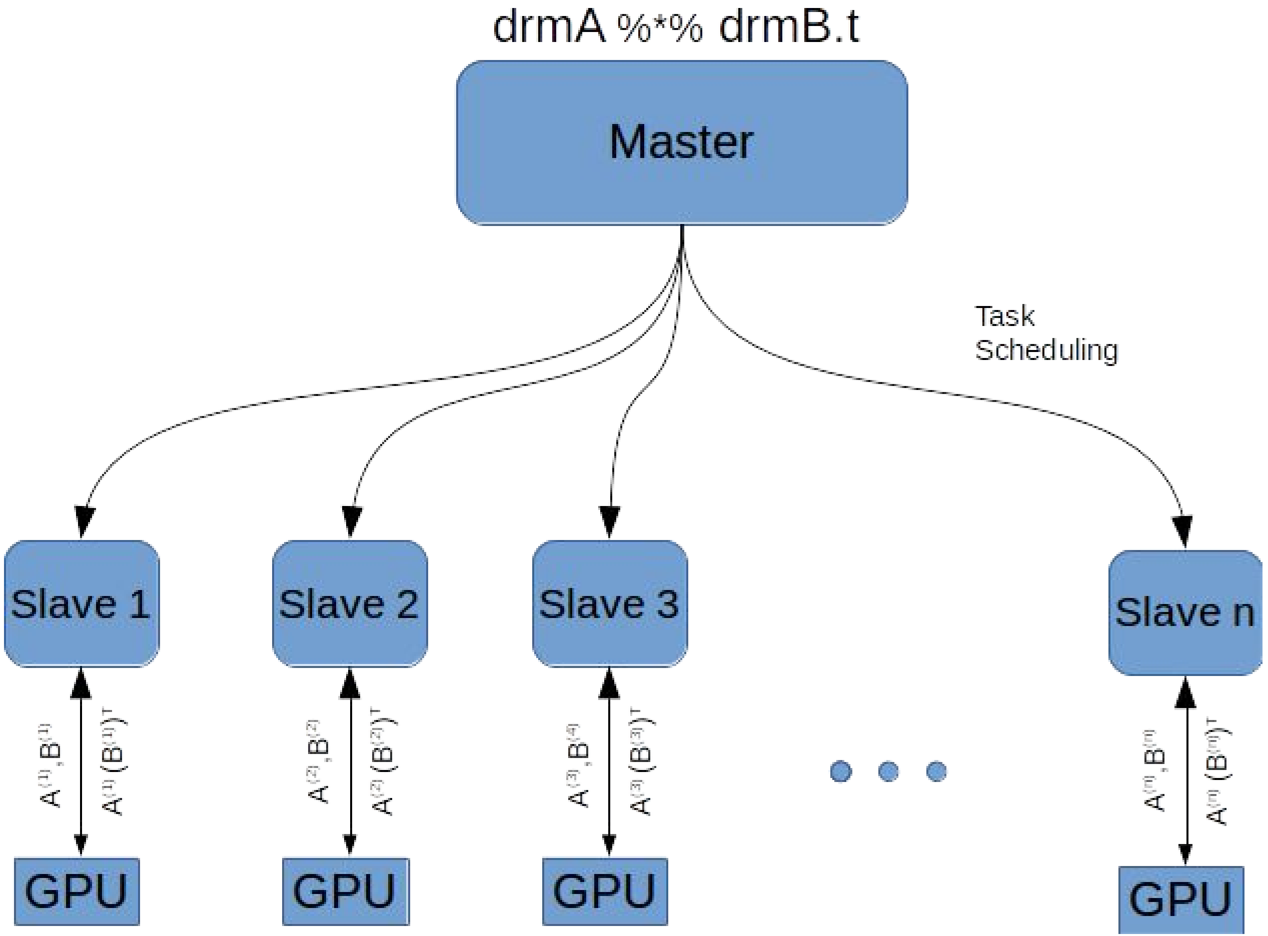
Took 1 seconds. Last updated by anonymous at time Jun 1, 2016 4:09:07 PM. (outdated)


settings
All fields:
col1
col2
xAxis
yAxis
group
size
col1
col2
(col1, col2)
522044378
0.5
0
-0.5
408642471
0.04146581981467975
(col1, col2)
-0.4996208224140664
-0.3
-0.25
-0.2
-0.15
-0.1
0.0051847139797826
0.1
0.15
0.2
0.25
0.3
0.49975777534
Took 1 seconds. Last updated by anonymous at time Jun 1, 2016 4:13:09 PM. (outdated)

Solve on CPU, GPU or JVM



Application Code
Scala R-Like DSL
Logical DAG
In-Memory
ViennaCL, Cuda, or CPU
Physical DAG
Spark DAG
Flink DAG
Driver
Apache Spark
Apache Flink
Compute Platform
In-Memory Algebra
ViennaCL, Cuda, or CPU
Cluster In-Memory Types



With GPU Integration, the Mahout syntax will not change at all.

Initial benchmarking on latest release

- Sparse MMul at geometry of 1000 x 1000 %*% 1000 x 1000 density = 0.2, with 5 runs
  Mahout JVM Sparse multiplication time: 1501 ms
  Mahout jCUDA Sparse multiplication time: 49 ms

  30x speedup

- Sparse MMul at geometry of 1000 x 1000 %*% 1000 x 1000 density = .02, with 5 runs
  Mahout JVM Sparse multiplication time: 34 ms
  Mahout jCUDA Sparse multiplication time: 4 ms

  8.5x speedup

- Sparse MMul at geometry of 1000 x 1000 %*% 1000 x 1000 density = .002, with 5 runs
  Mahout JVM Sparse multiplication time: 1 ms
  Mahout jCUDA Sparse multiplication time: 1 ms

Credits

- Anand Avati
- Andrew Musselman
- Andrew Palumbo
- Dmitriy Lyubimov
- Nathan Halko
- Pat Ferrel
- Sebastian Schelter
- Trevor D. Grant
- Suneel Marthi
- Alexey Grigorev
- Lucas Schelter
- Ted Dunning
- Zeno Gantner
- Isabel Drost-Fromm
- Drew Farris
- Grant Ingersoll
- Benson Margulies
- Frank Scholten
- Shannon Quinn
- Stevo Slavic
- Gokhan Capan
- Dan Filimon
- Ellen Friedman
- Tom Pierce
- Robin Anil
- Jim Benson
- Paritosh Ranjan

Pointers

- Apache Mahout has extensive documentation on Samsara
  - http://mahout.apache.org/users/environment/in-core-reference.html
  - https://mahout.apache.org/users/environment/out-of-core-reference.html

- Mahout Committer, Dmitriy Lyubimov's Blog - http://www.weatheringthroughtechdays.com/2015/04/mahout-010x-first-mahout-release-as.html

- Trevor Grant's Blog - https://rawkintrevo.org/2016/05/19/visualizing-apache-mahout-in-r-via-apache-zeppelin-incubating/

Contact Us

Mailing Lists

- dev@mahout.apache.org
- user@mahout.apache.org

Twitter: @ApacheMahout

Thank you. Questions?