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 Dmitry Lyubimov and Andrew Palumbo - Feb 2016
- v 0.10.0 - Apr 2015 - Mahout-Samsara vector-math DSL
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:

\[
G = BB^T - C - C^T + \xi^T \xi s_q^T s_q
\]

```scala
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
Data Types

• Scalar real values
  $\text{val } x = 2.367$

• In-memory vectors
  – dense
    $\text{val } v = \text{dvec}(1, 0, 5)$
  – 2 types of sparse
    $\text{val } w = \text{svec}((0 \rightarrow 1)::(2 \rightarrow 5)::\text{Nil})$

• In-memory matrices
  – sparse and dense
    – a number of specialized matrices
    $\text{val } A = \text{dense}((1, 0, 5), (2, 1, 4), (4, 3, 1))$
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
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

\[
\begin{align*}
val \ (drmQ, \ inMemR) &= \text{thinQR}(drmA) \\
val \ (drmU, \ drmV, \ s) &= \text{dssvd}(drmA, \ k = 50, \ q = 1)
\end{align*}
\]

• Caching of DRMs

\[
\begin{align*}
val \ drmA\_cached &= drmA\_.checkpoint() \\
drmA\_cached\_.uncache()
\end{align*}
\]
Unary Operators
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 => {1:0.6931471805599453,2:1.0986122888681098}
  1 => {0:1.0986122888681098,1:1.3862943611198906,2:1.6094379124341003}
}

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
// 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
• Execution is deferred, user composes logical operators

    \[ \text{val drmC = drmA.t } \%\% \text{ drmA} \]

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

    \[ \text{drmI.dfsWrite(path)} \]
    \[ \text{val inMemV = (drmU } \%\% \text{ drmM).collect} \]

• 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.)

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

• Computation of $A^T A$ in example

  ```scala
  val C = A.t %*% A
  ```
Optimization Example

• Computation of $A^T A$ in example

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

• Naïve execution

1\textsuperscript{st} pass: transpose $A$
(requires repartitioning of $A$)
Optimization Example

• Computation of $A^T A$ in example
  
  ```scala
  val C = A.t %*% A
  ```

• Naïve execution

  1\textsuperscript{st} pass: transpose $A$ (requires repartitioning of $A$)

  2\textsuperscript{nd} pass: multiply result with $A$ (expensive, potentially requires repartitioning again)
Optimization Example

Computation of $A^T A$ in example

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

Naïve execution:
1\textsuperscript{st} pass: transpose $A$ (requires repartitioning of $A$)
2\textsuperscript{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 $A^T A$ via **row-outer-product** formulation
  — executes in a single pass over row-partitioned $A$

$$A^T A = \sum_{i=0}^{m} a_i . a_i .^T$$
• Samsara computes $A^T A$ via **row-outer-product** formulation
  -- executes in a single pass over row-partitioned $A$

\[
A^T A = \sum_{i=0}^{m} a_i a_i^T
\]
Samsara computes $A^T A$ via the **row-outer-product** formulation.

It executes in a single pass over row-partitioned $A$.

$$A^T A = \sum_{i=0}^{m} a_i a_i^T$$
**Transpose-Times-Self**

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

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

![Diagram](https://example.com/diagram.png)
Transpse-Times-Self

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

$$A^T A = \sum_{i=0}^{m} a_i. a_i^T$$
Transpouse-Times-Self

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

$$A^T A = \sum_{i=0}^{m} a_i a_i^T$$
Samsara computes $A^T A$ via **row-outer-product** formulation

- executes in a single pass over row-partitioned $A$

\[
A^T A = \sum_{i=0}^{m} a_i a_i^T
\]
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 $r = h \text{AtA}$ to include arbitrary Cross-Occurrences with an LLR correlation test

$$r = h_a \text{AtA} + h_b \text{AtB} + h_c \text{AtC} \ldots$$

- $A =$ conversion history for all users, $B, C, \ldots =$ 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
// Imports and creating the distributed context, similar but not exactly the same

import org.apache.spark.api.java.*
import org.apache.spark.api.java.sc._
import org.apache.spark.api.java曦context._
import org.apache.spark.api.java曦partitions._
import org.apache.spark.api.java曦rdd._
import scala._
import org.apache.spark.api.java曦SparkContext.

val sc = new SparkContext("local", "Sample Application")
val sparkConf = new SparkConf().setAppName("Sample Application")
val spark = SparkSession.builder().appName("Sample Application").getOrCreate()


(data.map(_.length)).collect()
Distributed SSVD
Stochastic SVD (SSVD)

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

- $U = \text{Left Singular Vectors}$
- $V = \text{Right Singular Vectors}$
- $E = \text{Diagonal Matrix with decaying singular values}$

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](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 decomposition 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 reqmt that k + p <= rank(A)

    Upper bound of rank(A) = min(drmA.nrows, drmA.ncols)
EigenFaces

• 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

Images aligned by Funnelings, credit Learning to Align 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

Download Faces Data

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

Download Faces Data

```
%sh
mkdir -p zeppelin-0.7.0-SNAPSHOT/webapps/webapp/eigenfaces/input
tar -xzf lfw-deepfunneled.tgz
```

Put Faces Data in HDFS

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

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

Download Faces Data

```
%sh
```

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

Put Faces Data in HDFS

```
%sh
```

Tack 2 min 17 sec. Last updated by anonymous at November 10 2016, 10:15:25 AM. (outdated)
Add Image Processing Dependencies

```scala
%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_2.18-0.12.3-SNAPSHOT.jar")
```

DepInterpreter(%depl) deprecated. Load dependency through GUI interpreter menu instead.
DepInterpreter(%depl) deprecated. Load dependency through GUI interpreter menu instead.
DepInterpreter(%depl) deprecated. Load dependency through GUI interpreter menu instead.
DepInterpreter(%depl) 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

```scala
%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._

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

import org.apache.mahout.math._
```
Create DRM of Vectorized Images

```scala
%sparkMahout.spark

import com.sksamuel.scrmage._
import com.sksamuel.scrmage.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.scrmage._
import com.sksamuel.scrmage.filter.GrayscaleFilter
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

```scala
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) // calculates 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.SubtractMean()

---

Took 42 min 22 sec. Last updated by anonymous at November 13 2016, 10:41:57 PM.
Mahout Distributed SSVD to get Eigenfaces

```scala
%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(1

Took 1 hrs 1 min 48 sec. Last updated by anonymous at November 13 2016, 11:35:42 PM.)```
%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.scimage.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
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: 1.8284546624238976E-5}
  1 => {0: -0.5849668540131455, 1: 0.008078753046618811, 2: 0.13397645857511886}
  2 => {0: -0.5322766038520063, 1: 0.2591761531093102, 2: 0.13348013659591368}
  3 => {0: 1.4728661457904864, 1: -2.85898158079829, 2: 0.02628048664696E-4}
  4 => {0: -1.0669267542210157, 1: -0.9975250697214353, 2: 0.054959594073650874}
  5 => {0: 2.190375803711219, 1: 0.1076973899036782, 2: 0.01443187324573094}
  6 => {0: -2.322801014520712, 1: 3.2169688654260145, 2: 0.008687349000867E-5}
  7 => {0: 0.9564289803558209, 1: 0.020272550397296755, 2: 0.10089785428577039}
  8 => {0: -3.154759694595444, 1: 2.8778564830191375, 2: 1.7426370513352694E-5}
  9 => {0: -1.8361745541885914, 1: 1.9873637471279224, 2: 0.004089036909448981}
...

drm5000Sampled: org.apache.mahout.math.Matrix =
{
  0 => {0: -1.55955426611664, 1: -2.877697279171333, 2: 7.458223045528731E-4}
  1 => {0: -2.877697279171333, 2: -7.458223045528731E-4}
...}

Took 8 seconds. Last updated by anonymous at time Jun 1, 2016 4:23:29 PM. (outdated)
z.put("mahout1000Table", matrix2table(drm10000Sampled))
z.put("mahout5000Table", matrix2table(drm50000Sampled))
z.put("mahout10000Table", matrix2table(drm100000Sampled))

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

```r
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"
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
```r
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
```
Tablify Matrix Using Zeplin + Angular

```javascript
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)
```

Took 1 seconds. Last updated by anonymous at time Jun 1, 2010 4:09:07 PM. (outdated)
Solve on CPU, GPU or JVM
With GPU Integration, the Mahout syntax will not change at all.
Initial benchmarking on latest release

- Sparse MMul at geometry of $1000 \times 1000$ $\% \times \%$ $1000 \times 1000$ 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 $1000 \times 1000$ $\% \times \%$ $1000 \times 1000$ 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 $1000 \times 1000$ $\% \times \%$ $1000 \times 1000$ density = .002, with 5 runs
  - Mahout JVM Sparse multiplication time: 1 ms
  - Mahout jCUDA Sparse multiplication time: 1 ms
Credits
Pointers

• Apache Mahout has extensive documentation on Samsara

• Mahout Committer, Dmitriy Lyubimov’s Blog -

• 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?