SCALING FACTORIZATION MACHINES ON APACHE SPARK WITH PARAMETER SERVERS

Nick Pentreath

Principal Engineer, IBM

SPARK SUMMIT
EUROPE 2016

About

- About me
  - *@MLnick*
  - Principal Engineer at IBM working on machine learning & Spark
  - Apache Spark PMC
  - Author of *Machine Learning with Spark*

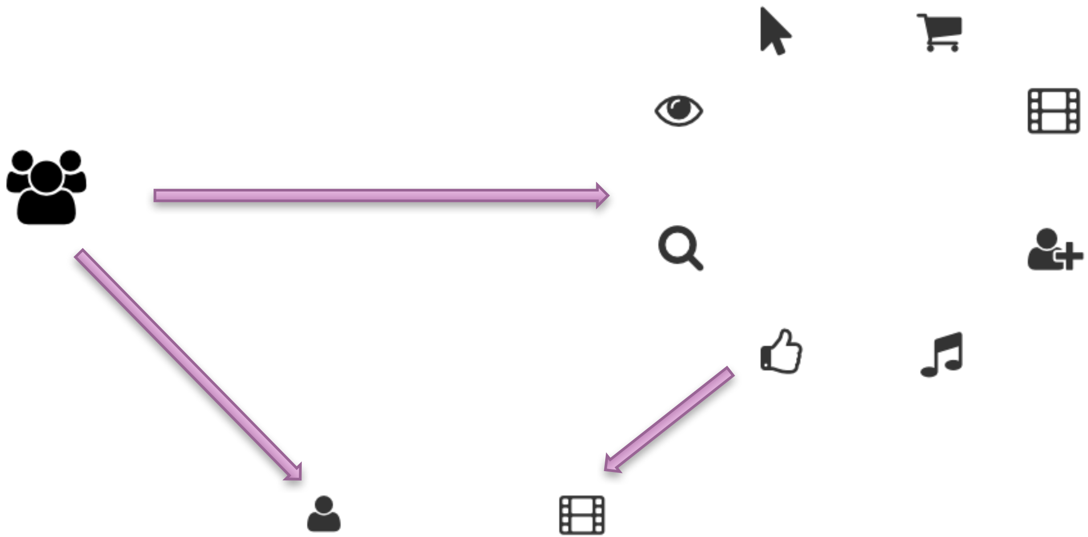
Agenda

- Brief Intro to Factorization Machines
- Distributed FMs with Spark and Glint
- Results
- Challenges
- Future Work

FACTORIZATION MACHINES

Factorization Machines

Feature interactions



[1, 0, 0, 1, 0, 0]

Factorization Machines

Linear Models

$$w_0 + \sum_{i=1}^{n} w_i x_i \longrightarrow w_0 + \underbrace{[w_u, 0, 0, w_i}_{\text{Bias terms}}, \ldots]$$

Not expressive enough!

Factorization Machines

Polynomial Regression

$$w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j \Longrightarrow w_0 + [\underbrace{w_u, 0, 0, w_i,}_{\text{Bias terms}} \ldots, \underbrace{w_{ui},}_{\text{Interaction term}} \ldots]$$

$O(d^2)$

$n << d$

Factorization Machines

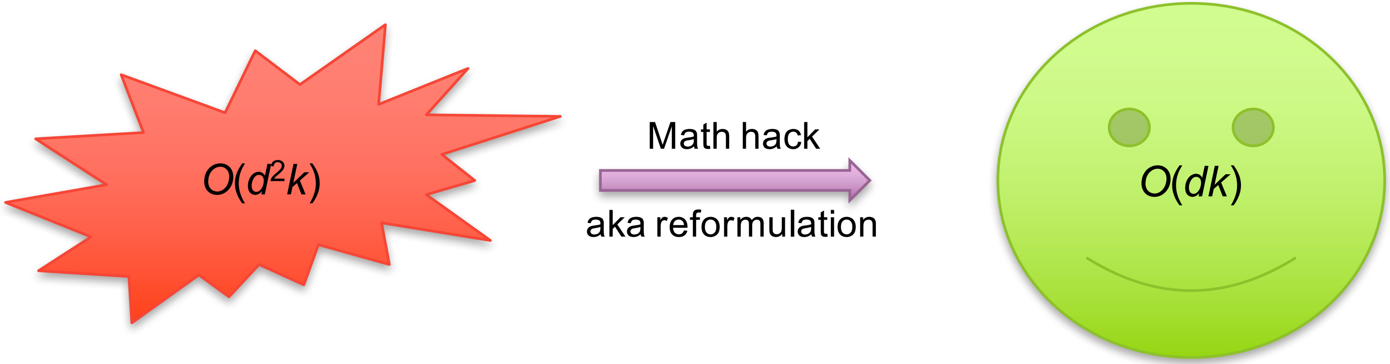
Factorization Machine

$$w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \vec{v_i}\, \vec{v_j} \rangle x_i x_j \Rightarrow w_0 + [\underbrace{w_u, 0, 0, w_i}_{\text{Bias terms}}, \ldots, \underbrace{\langle \mathbf{v}_u \mathbf{v}_i \rangle}_{\text{Factorized interaction term}}, \ldots]$$

$O(d^2k)$

Factorization Machines

Factorization Machine



Not convex, but efficient to train using SGD, coordinate descent, MCMC

Factorization Machines

Factorization Machine

[1,....,1,....,1,....,1,....,1,....,3.14,....1,....]

Standard matrix factorization (with biases)

Metadata features

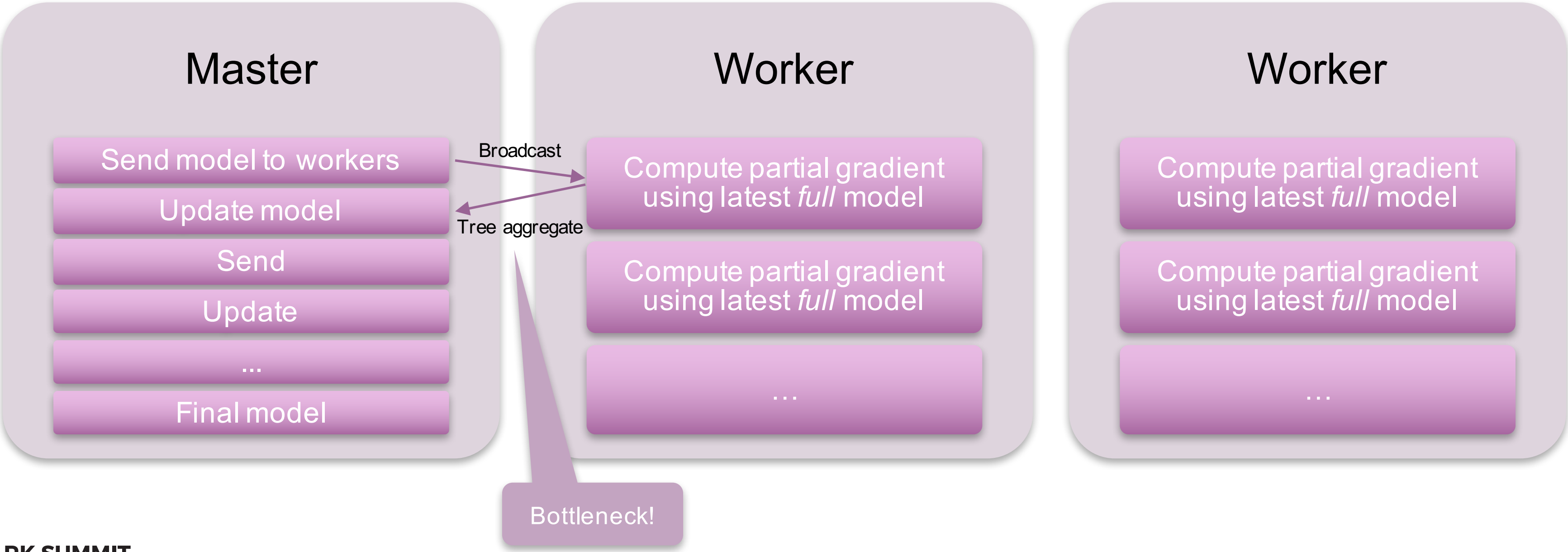
Contextual features

Model size can still be very large! e.g. video sharing, online ads, social networks

DISTRIBUTED FM MODELS ON SPARK

Linear Models on Spark

- Data parallel



Parameter Servers

- Model & data parallel

Master

Schedule work

…

Worker

Compute partial gradient using latest *partial* model

Compute partial gradient using latest *partial* model

…

Pull partial model

Push partial gradient

Parameter Server

Update model

Update model

…

Only push/pull required features

Distributed FMs

- spark-libFM
  - Uses old MLlib `GradientDescent` and `LBFGS` interfaces
- DiFacto
  - Async SGD implementation using parameter server (ps-lite)
  - Adagrad, L1 regularization, frequency-adaptive model-size
- Key is that most real-world datasets are *highly sparse* (especially high-cardinality categorical data), e.g. online ads, social network, recommender systems
- Workers only need access to a small piece of the model

Glint is a parameter server built using Akka
For more info, see Rolf's talk at 17:15!

GlintFM

- Procedure:

  1. Construct Glint Client
  2. Create distributed parameters
  3. Pre-compute required feature indices (per partition)
  4. Iterate:
     - Pull partial model (blocking)
     - Compute partial gradient & update
     - Push partial update to parameter servers (can be async)
  5. Done!

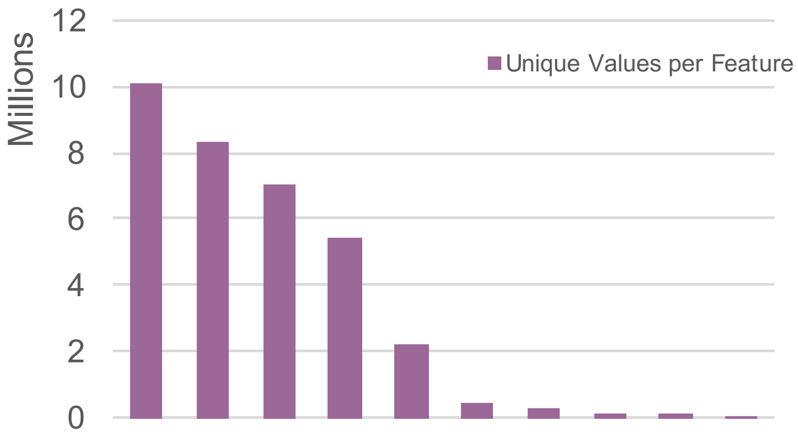
```scala
val client = Client(config)
val w = client.vector[Double](d)
val V = client.matrix[Double](d, k)
// training
train.foreachPartition { iter =>
  // compute partition statistics
  val localKeys = { ... }
  // iterate
  for (i <- 1 to numIterations) {
    // pull latest model for l local keys
    val localW = w.pull(localKeys) // 1 x l vector
    val localV = V.pull(localKeys) // l x k matrix
    // compute gradient
    partitionData.foldLeft(new FMAggregator(...)) { case (agg, features, label) =>
      agg.add(features, label, localW, localV)
    }
    // compute and push update
    val updates = ...
    w.push(...)
    V.push(...)
  }
}
```

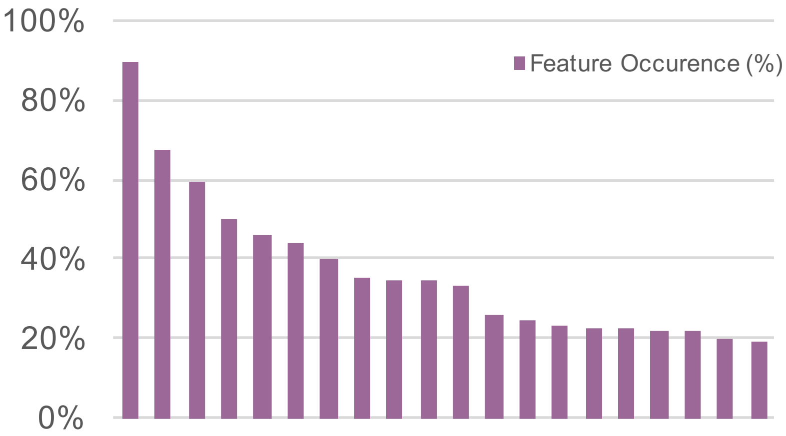
RESULTS

Data

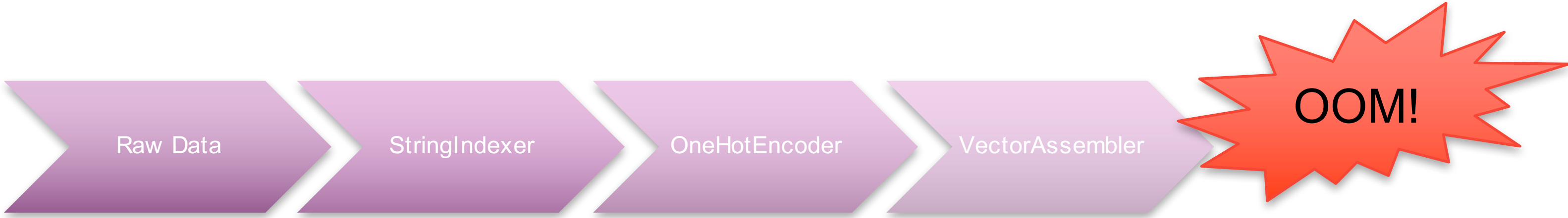
Criteo Display Advertising Challenge Dataset

- 45m examples, 34m unique features, 48 nnz /example





Feature Extraction



label	i1	i2	i3	i4	i5	i6	i7	i8	i9
0	1	1	5	0	1382	4	15	2	181
0	2	0	44	1	102	8	2	2	4
0	2	0	1	14	767	89	4	2	245
0	NULL	893	NULL	NULL	4392	NULL	0	0	0
0	3	-1	NULL	0	2	0	3	0	0

label	i1_idx	i1_ohe	features
0	2.0	(152,[2],[1.0])	(273492,[2,153,28...
0	3.0	(152,[3],[1.0])	(273492,[3,152,28...
0	3.0	(152,[3],[1.0])	(273492,[3,152,28...
0	0.0	(152,[0],[1.0])	(273492,[0,923,28...
0	4.0	(152,[4],[1.0])	(273492,[4,154,28...

Feature Extraction

Solution? "Stringify" + `CountVectorizer`

```
from pyspark.sql import Row
cols = df.columns
def convert_row(row):
    l = row.label
    i = 1
    v = []
    for c in cols[1:]:
        if row[i] is not None:
            v.append("%s=%s" % (c, row[i]))
        i += 1
    return Row(label=l, raw=v)

df_stringified = spark.createDataFrame(df.rdd.map(lambda row: convert_row(row)))
```

```
Row(i1=u'1', i2=u'1', i3=u'5', i4=u'0', i5=u'1382',... )
```

Convert set of `String` features into `Seq[String]`

```
Row(raw=[u'i1=1', u'i2=1', u'i3=5', u'i4=0', u'i5=1382', ...)
```

Feature Extraction

label	i1	i2	i3	i4	i5	i6	i7	i8	i9
0	1	1	5	0	1382	4	15	2	181
0	2	0	44	1	102	8	2	2	4
0	2	0	1	14	767	89	4	2	245
0	NULL	893	NULL	NULL	4392	NULL	0	0	0
0	3	-1	NULL	0	2	0	3	0	0

label	raw	features
0	[i1=1, i2=1, i3=5...	(273531,[0,1,2,3,...
0	[i1=2, i2=0, i3=4...	(273531,[0,1,2,3,...
0	[i1=2, i2=0, i3=1...	(273531,[0,3,4,6,...
0	[i1=NULL, i2=893,...	(273531,[0,1,2,3,...
0	[i1=3, i2=-1, i3=...	(273531,[0,1,2,3,...

Feature Extraction

Raw Data

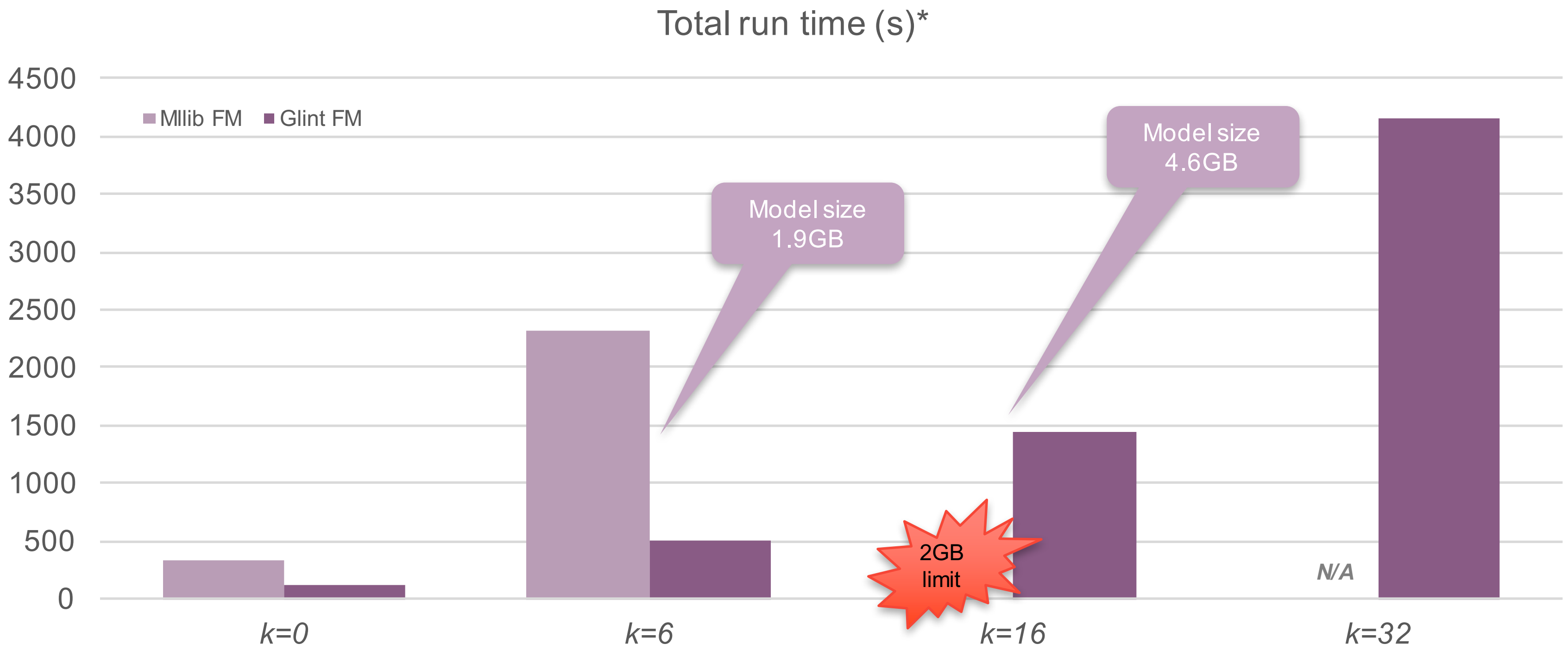
Stringify

HashingTF

label	i1	i2	i3	i4	i5	i6	i7	i8	i9
0	1	1	5	0	1382	4	15	2	181
0	2	0	44	1	102	8	2	2	4
0	2	0	1	14	767	89	4	2	245
0	NULL	893	NULL	NULL	4392	NULL	0	0	0
0	3	-1	NULL	0	2	0	3	0	0

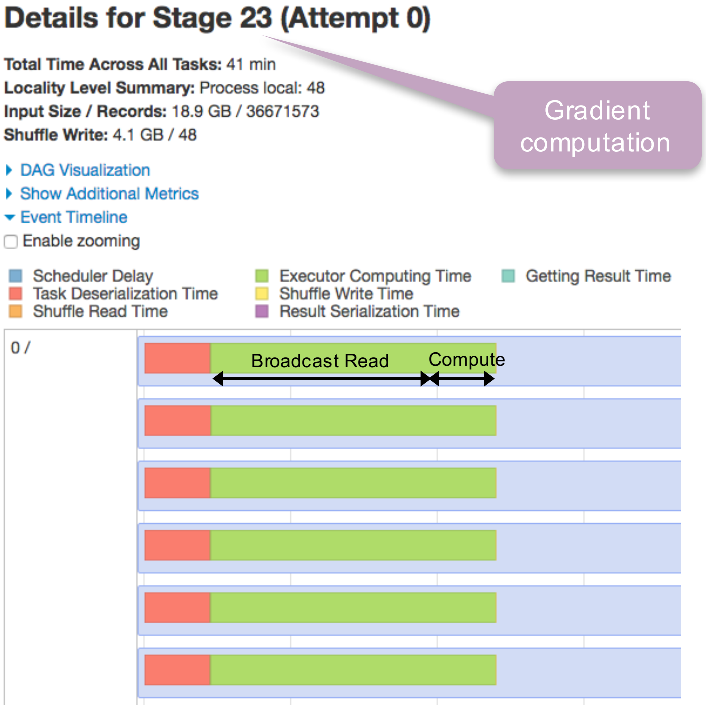
label	raw	features
0	[i1=1, i2=1, i3=5...	(262144,[2411,726...
0	[i1=2, i2=0, i3=4...	(262144,[5352,934...
0	[i1=2, i2=0, i3=1...	(262144,[14069,15...
0	[i1=NULL, i2=893,...	(262144,[4201,693...
0	[i1=3, i2=-1, i3=...	(262144,[6935,140...

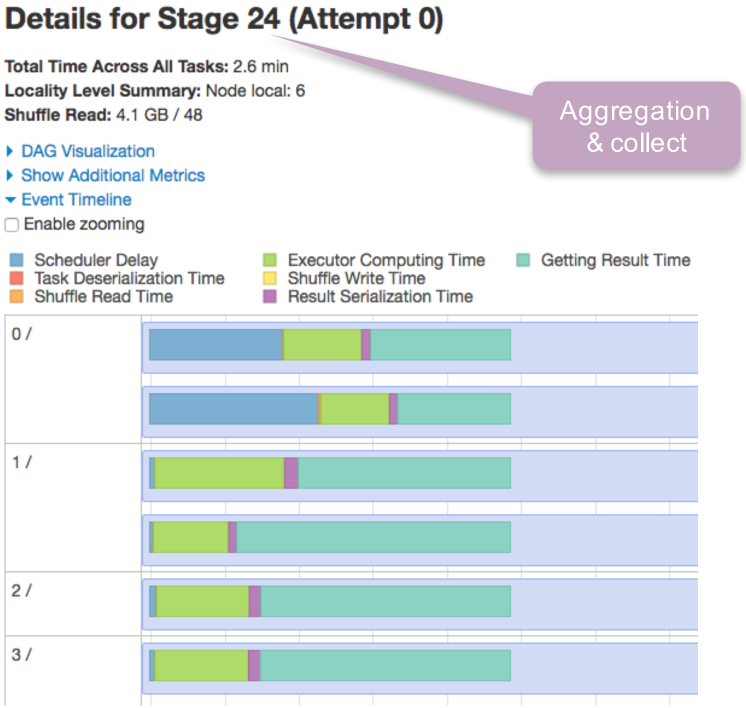
Performance



*10 iterations, 48 partitions, fit intercept

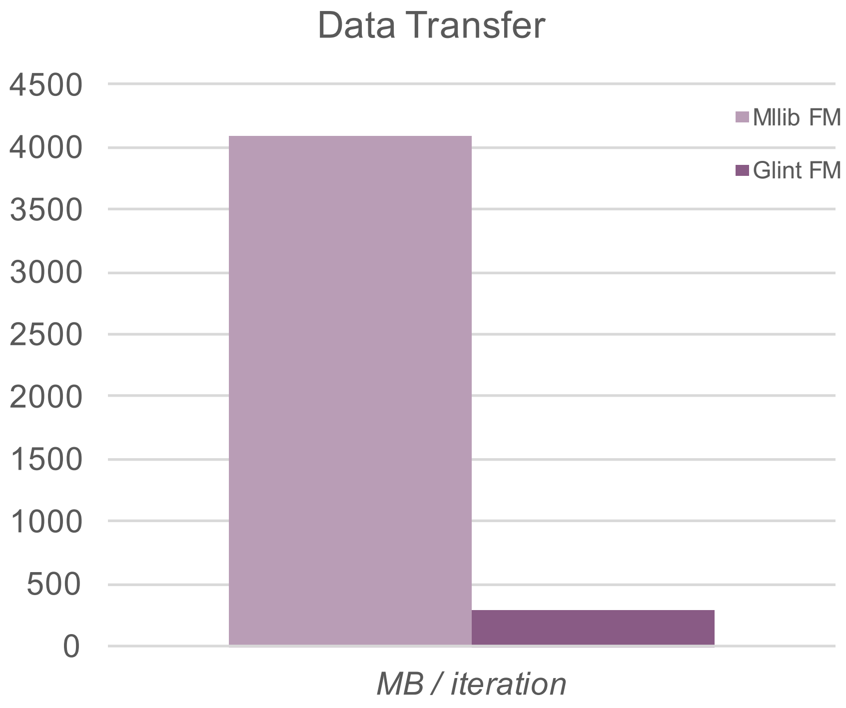
Performance

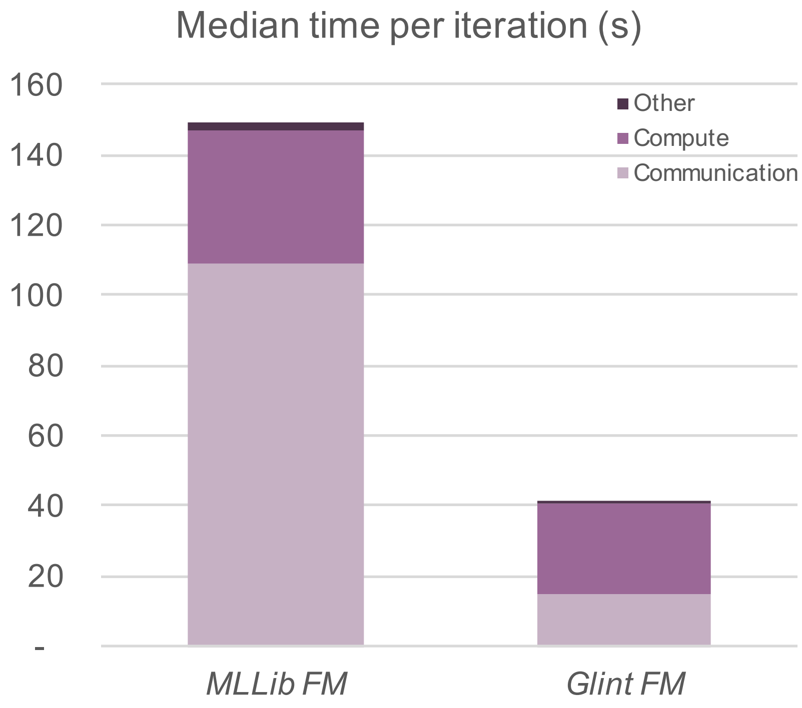




$k = 6$, 10 iterations, 48 partitions, fit intercept

Performance





$k = 6$, 10 iterations, 48 partitions, fit intercept

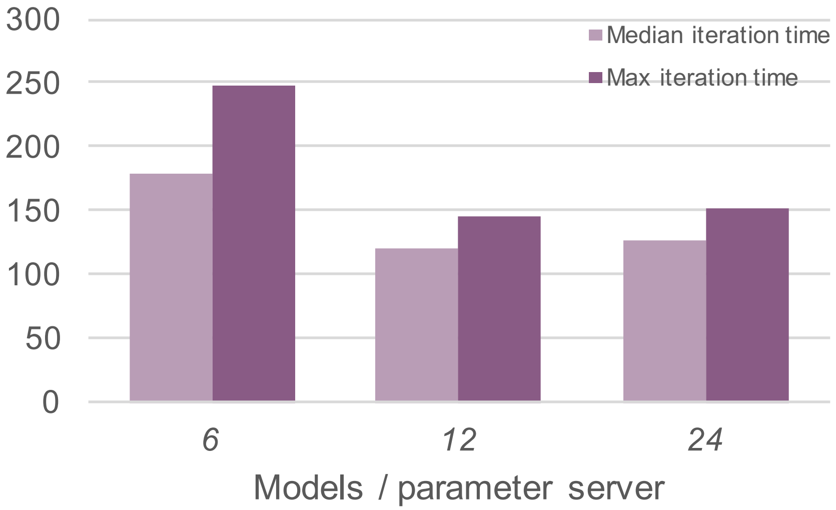
CHALLENGES & FUTURE WORK

Challenges

- Tuning configuration
  - Glint - models/server, message size, Akka frame size
  - Spark - data partitioning (can be seen as “mini-batch” size)
- Lack of server-side processing in Glint
  - For L1 regularization, adaptive sparsity, Adagrad
  - These result in better performance, faster execution
- Backpressure / concurrency control

Challenges

- Tuning models / server



***k = 16**, *10 iterations, 48 partitions, fit intercept*

Challenges

- Index partitioning for "hot features"
  - `CountVectorizer` for features leads to hot spots & straggler tasks due to sorting by occurrence
  - `OneHotEncoder` OOMed... but can also face this problem
  - Spreading out features is critical (used feature hashing)

***k = 16**, *10 iterations, 48 partitions, fit intercept*

Future Work

- Glint enhancements
  - Add features from DiFacto, i.e. L1 regularization, Adagrad & memory-adaptive *k*
  - Requires support for UDFs on the server
  - Built-in backpressure (Akka Artery / Streams?)
  - Key caching – 2x decrease in message size
- Mini-batch SGD within partitions
- Distributed solvers for ALS, MCMC, CD
- Relational data / block structure formulation
  - www.vldb.org/pvldb/vol6/p337-rendle.pdf

References

- Factorization Machines
  - http://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf
  - https://github.com/ibayer/fastFM
  - www.libfm.org
  - https://github.com/zhengruifeng/spark-libFM
  - https://github.com/scikit-learn-contrib/polylearn
- DiFacto
  - https://github.com/dmlc/difacto
  - www.cs.cmu.edu/~yuxiangw/docs/fm.pdf
- Glint / parameter servers
  - https://github.com/rjagerman/glint
  - https://github.com/dmlc/ps-lite

THANK YOU.

@MLnick

github.com/MLnick/glint-fm
spark.tc

SPARK SUMMIT
EUROPE 2016