Data Management in Machine Learning: Challenges, Techniques, and Systems

Arun Kumar
UC San Diego
La Jolla, CA, USA

Matthias Boehm
IBM Research – Almaden
San Jose, CA, USA

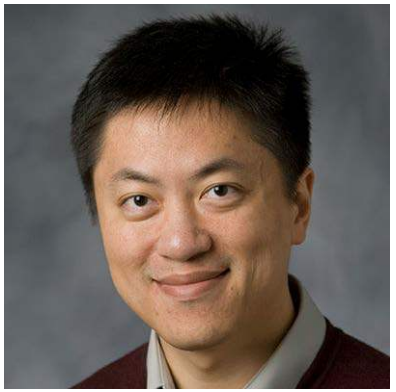
Jun Yang
Duke University
Durham, NC, USA

SIGMOD 2017

Who We Are







Arun Kumar

UC San Diego
La Jolla, CA, USA

Matthias Boehm

IBM Research – Almaden
San Jose, CA, USA

Jun Yang

Duke University
Durham, NC, USA



Bismarck



Columbus



Orion



Hamlet

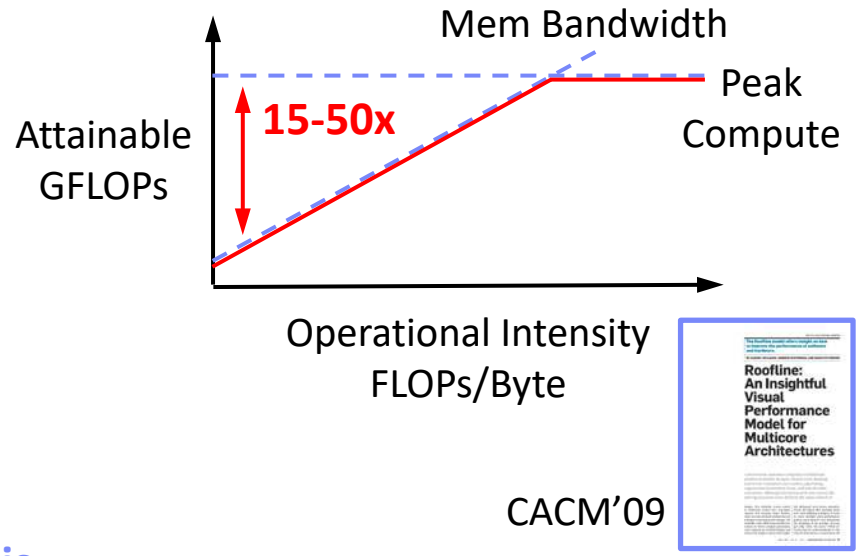
Apache SystemML™



Motivation: A Data-Centric View of ML

- **Application Perspective**
  - Machine learning / advanced analytics / deep analytics
  - ➔ **Modern data-driven applications** (e.g., BI, e-commerce, healthcare)

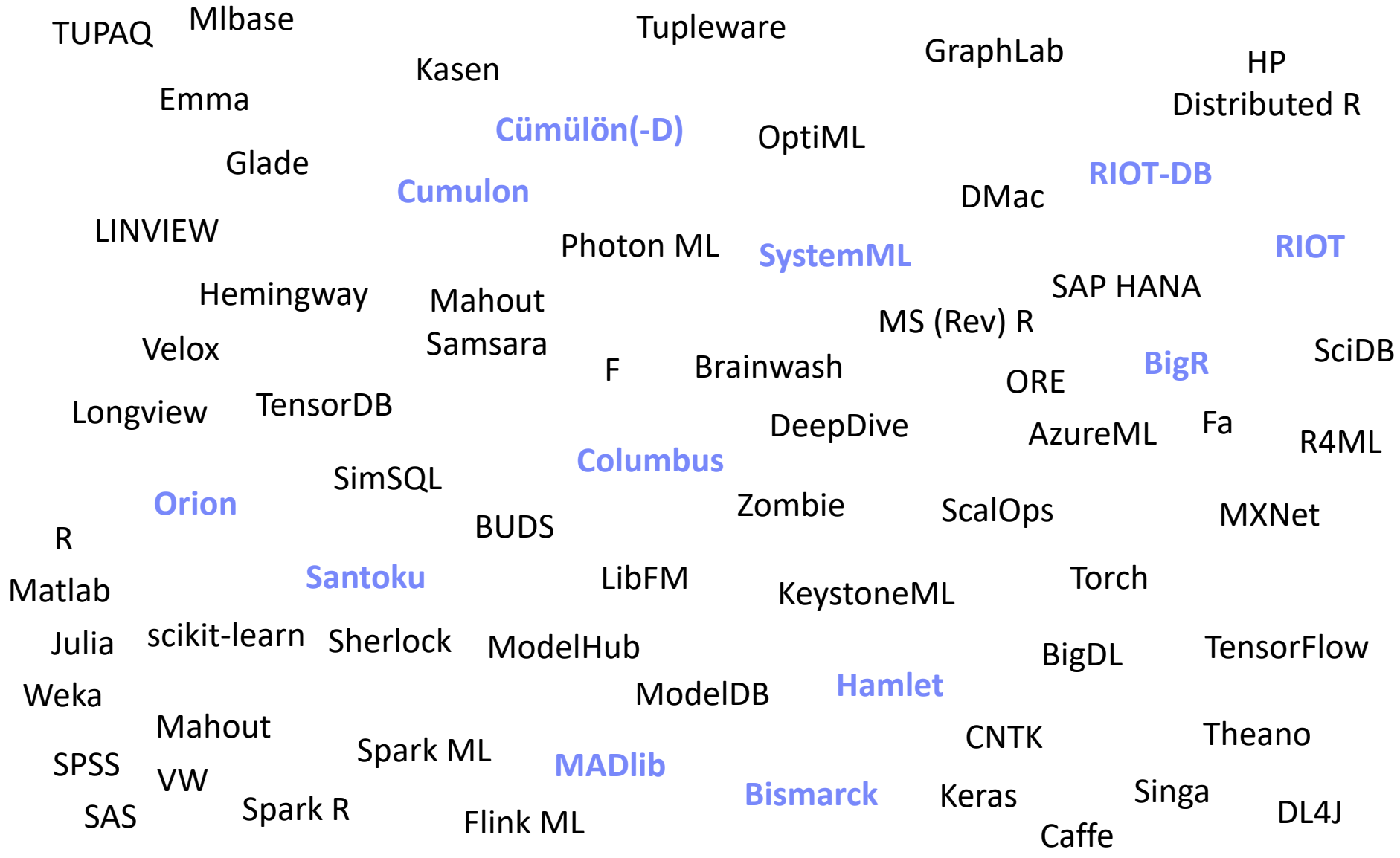
- **Workload Perspective**
  - Repetitive ML workflows
  - Often **iterative ML algorithms**
  - Often **I/O-bound operations** (e.g., matrix-vector multiplications)



- **Systems Perspective**
  - **ML in data systems**
  - **DB-inspired ML systems**
  - **ML Lifecycle Systems**

This Tutorial

Motivation: Systems Landscape



Motivation: Tutorial Goals

- **Overall Goal:** Comprehensive review of systems and techniques that tackle data management challenges in the context of ML workloads

- **#1 Categorize Existing Systems**
  - ML in data systems, DB-inspired ML systems, ML lifecycle systems
- **#2 Survey State-of-the-Art Techniques**
  - Query gen, UDFs, factorized learning, deep DBMS integration
  - Optimization and runtime techniques, incl. resource elasticity
  - Model selection and model management

➔ **Intended Takeaways**
  - Awareness of existing systems and techniques
  - Survey of effective optimization and runtime techniques
  - Overview of open research problems

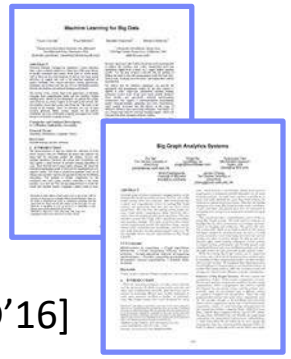
What this Tutorial is NOT

- **Introduction to Machine Learning**

- **Tutorial on General-Purpose Systems**
  - Dataflow systems
  - Graph-focused systems

[SIGMOD'13]

[SIGMOD'16]



- **Tutorial on Deep Learning**
  - Deep learning algorithms
  - Deep learning systems (e.g., Torch, Theano, BigDL, TensorFlow, MXNet, CNTK, Singa, Keras, Caffe, DL4J)

[SIGMOD Record'16]



- **Tutorial on ML for RDBMS Internals**
  - Cost models
  - Workload prediction (e.g., in Peloton)

[CIDR'17]



Tutorial Outline

ML in Data Systems

- **2 Query Generators and UDFs** 14min JY
- **3 Factorized Learning and Deep RDBMS Integration** 8min AK

DB-Inspired ML Systems

- **4 Rewrites, Operator Selection, and Fusion** 14min MB
- **5 Compression, Scan Sharing, and Index Structures** 10min MB
- **6 Cloud Resource Elasticity** 10min JY

ML Lifecycle Systems

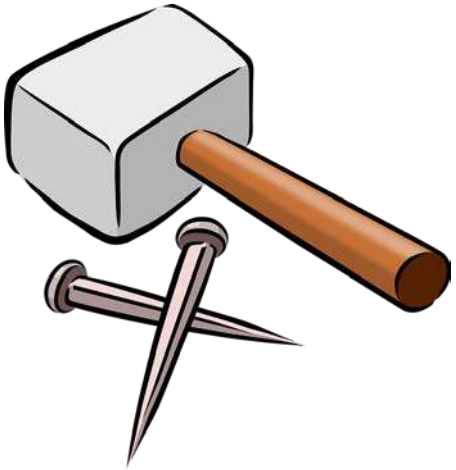
- **7 Feature Engineering, Model Selection/Management** 16min AK

Open Problems and Q&A

Part 2: ML with SQL & UDF

"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail."

Abraham Maslow, 1966



Jun Yang

Duke University
Durham, NC, USA

SIGMOD 2017

https://pixabay.com/en/nails-hammer-woodwork-tool-37063/

ML in Database – Why?

- **Convenience**
  - "Elephants" (octopi?) have shown remarkable flexibility
  - A single platform for not only data management, transformation, and querying, but also ML and application of insights
- **Efficiency**
  - Move the analysis, not data
  - Can co-optimize various steps involved in the "big data pipeline"
- **Declarativeness**
  - Simplifies development
  - Enables effective automatic optimization, which helps scalability/efficiency
  - One area where the DB community has plenty to offer

Roadmap

- **First, examples of what SQL can do for ML, at various levels of abstraction:**
  - Matrix multiply
  - Ordinary least squares
  - Gradient descent

  (See backup slides for
  - k-means
  - Markov-chain Monte-Carlo)

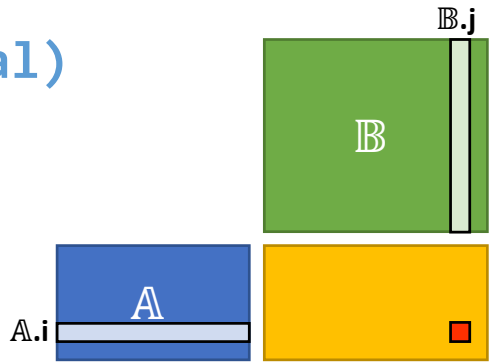
- **Then, a brief discussion of approaches to using SQL for ML**

Matrix Multiply: Take 1

- **Data:** `A(i,j,val)`, `B(i,j,val)`
  - Basically a sparse representation

MAD Skills [VLDB'09]

-
```
SELECT A.i, B.j, SUM(A.val*B.val)
FROM A, B
WHERE A.j = B.i
GROUP BY A.i, B.j;
```



- **Works pretty well for sparse matrices**
- **Not so good for dense matrices, but still beats "small-data" platforms when data doesn't fit in memory**

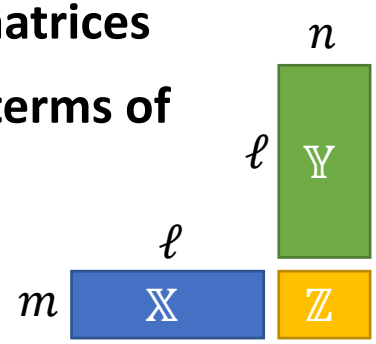
Matrix Multiply: Take 2

MAD Skills [VLDB'09]

- **Data:** `A(i,row), B(j,col)`
  - `row` and `col` are `ARRAY` types or user-defined vector types
  - Basically a row-/column-major representation
- **UDF (user-defined function):** `dotproduct(`v_1`,`v_2`)` **computes the dot product of two vectors**

```
SELECT A.i, B.j, dotproduct(A.row, B.col)
FROM A, B;
```

- **Works fine for dense matrices**
- **But still suboptimal in terms of compute-to-I/O ratio**





Computation: $O(\ell mn)$, or volume I/O: $O(m\ell + \ell n + nm)$, or surface

☞ Want instead "blocky" units to maximize compute-to-I/O ratio

- **Also note the change in representation (from input to output)**

Matrix Multiply: Take 3

- **Data:** `A(i,j,V)`, `B(i,j,V)`

  RIOT-DB [CIDR'09] *SimSQL* [ICDE'17]

  - `V` represents a submatrix; assume the dimensions are compatible
  - Basically a blocked representation
- **UDFs**
  - `matmult(`V_1`,`V_2`)` computes the product of two matrices
  - `matsum(`V`)` is a UDA (user-defined aggregate) that sums up input matrices

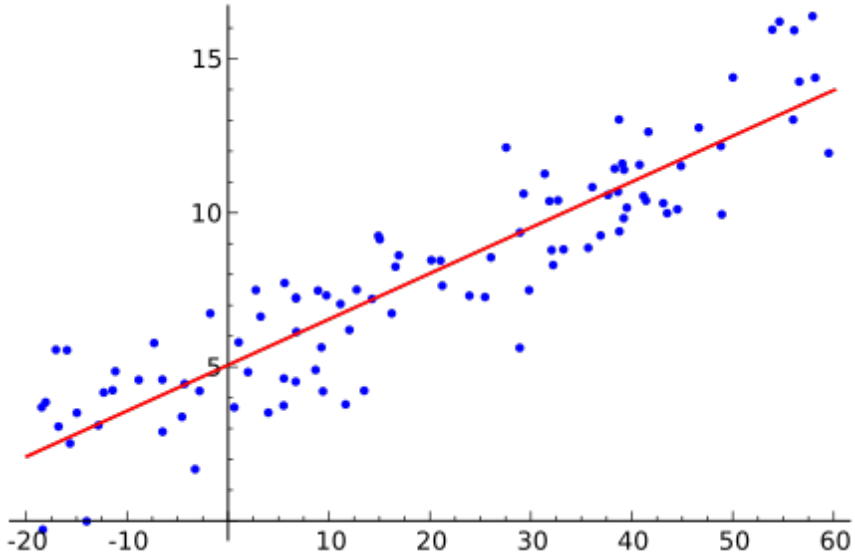
```
SELECT A.i, B.j, matsum(matmult(A.V, B,V))
FROM A, B
WHERE A.j = B.i
GROUP BY A.i, B.j;
```

- **Choose a "big enough" `V` with good aspect ratio**
  - E.g., square `V`'s beat skinny `V`'s
- **UDFs can use optimized libraries like BLAS**

Ordinary Least Squares

- **To fit data (X, y) to a linear model $y = X\beta + \epsilon$:**

$$\beta^{\star} = (X^T X)^{-1} X^T y$$



- **Computation involves basic matrix operators expressible in SQL with help of UDFs**
  - Inverse is tougher, but assuming the input matrix is small:
    - Code it as a UDF with memory-resident input
    - Processing won't benefit from DBMS though

***MAD* [VLDB'09, '12]**
***SimSQL* [ICDE'17]**

https://en.wikipedia.org/wiki/File:Linear_regression.svg

Observation

- **How far can UDF and UDA go? Surprisingly very!**
- **UDF (oftentimes coded in other languages, e.g., Python and R)**
  - Either on the tuple-level (invoked by SQL queries),
  - Or like an application program (invoking SQL queries)
- **UDA**
  - `Init(state)` initializes the state
  - `Accumulate(state, data)` computes updated state with new data
  - [optional] `Merge(state, state)` merges intermediate results computed over disjoint input subsets
  - `Finalize(state)` computes the final result from the state

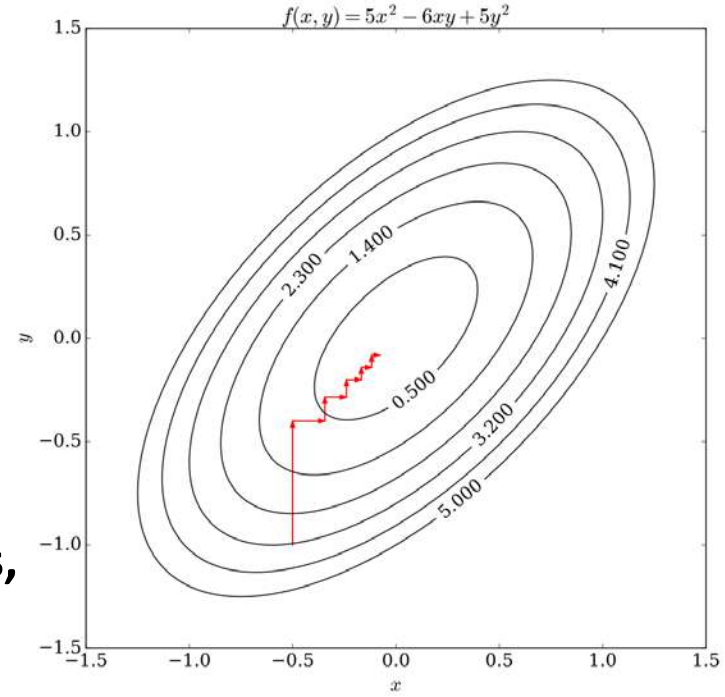
  ☞This pattern covers lots of iterative computation in ML, e.g.
    - k-means (backup slides) ***GLADE*** **[LADIS'11,SIGMOD'12],** ***MADlib*** **[VLDB'12]**
    - Gradient descent (next)

Gradient Descent (GD)

- **Given a model with parameters w, we want to learn from data D, i.e., minimize a loss function $F(w; D)$**
  - E.g., sum of loss over all training data + a regularization term

- **Start with some guess w_0**
- **In each step $t + 1$, update w in the direction of the gradient of the loss function at w_t, i.e., $F'(w_t)$**
- **Rinse and repeat**

- **Under certain (commonly held) conditions, GD converges to a local minimum**
  - If F is convex, that's its global minimum



https://zh.wikipedia.org/wiki/File:Coordinate_descent.svg

Stochastic GD (SGD)

- **Suppose $F(w; D)$ is linearly separable over D**
  - I.e., $F(w; D) = \sum_i f_i(w; d_i)$,
    where i iterates over the data points $D = \{d_i\}_i$
- **Instead of updating w using the "full gradient" computed over D in each GD step, just choose a single point in D**
  - I.e., use $f_i'(w)$ to approximate $F'(w)$

- **Remarkably, for convex $F(w)$, SGD also converges to the global minimum, even if we pick points from D in a fixed, arbitrary order**
  - Albeit at a slower rate

GD/SGD in SQL

- **GD (full gradient)**
  - Computation of full gradient over D can be done by a query using UDA
  - Several options for driving outer loop
    - ***MADlib* [VLDB'12]** uses Python UDF
    - ***ScalOps* [DeBull'12]** uses Datalog
      - Underlying implementation is MapReduce instead of SQL

- **SGD** ***Bismarck* [SIGMOD'12]**
  - The entire procedure can be written as a query over D using UDA—each `Accumulate()` corresponds to one step

MCMC in SQL

- **MCMC (Markov-Chain Monte-Carlo) is a key method in Bayesian ML**
- **Bayesian ML comes down to analyzing the "posterior" distribution**

  P(parameters, hidden variables | observations)
- **Direct analysis is often hard, so we use Monte-Carlo simulation**
  - Repeatedly sample from the posterior, and analyze the samples
- **But sampling directly from the posterior is often hard, so we use MCMC**
  - A sampler generates a Markov chain of samples, whose stationary distribution is the target posterior

☞**You can do Gibbs sampling (a form of MCMC) in *SimSQL* [SIGMOD'13]**
  - With user-define "value-generating" functions that draw samples
  - See backup slides for details

Approaches to SQL+ML

Backend choices

- **"On top of" (e.g., *RIOT-DB* [CIDR'09], *MAD* [VLDB'09,VLDB'12]) vs. "inside" DBMS (e.g., *SimSQL* [ICDE'17])**
- **Not DBMS, but still inspired by or rooted in DBMS**
  - General-purpose "big-data" platform (e.g., ***SystemML* [ICDE'11,VLDB'16]**, ***Cumulon* [SIGMOD'13]**)
  - Specialized system from ground up (e.g., ***RIOT* [ICDE'10]**, ***SciDB* [CSE'13]**)

Interface choices

- **SQL + libraries or extensions (e.g., *MAD* [VLDB'09,VLDB'12], *SimSQL* [ICDE'17], *Oracle Data Mining*, …)**
- **ML-oriented languages on top of SQL (e.g., *RIOT-DB* [CIDR'09], *BUDS/SimSQL* [SIGMOD'17], *Oracle R Enterprise*, …)**

Interface: SQL + Libraries/Extensions

- **Especially nice with integrated model management, e.g., *Oracle Data Mining***
  - Can create, store, update, and apply models in SQL

```
-- Create model settings:
CREATE TABLE svm_settings(
  setting_name VARCHAR2(30), setting_value VARCHAR2(30));
INSERT INTO svm_settings VALUES(
  dbms_data_mining.algo_name,
  dbms_data_mining.algo_support_vector_machines);
-- …
-- Build model:
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'svm_model',
  mining_function => dbms_data_mining.classification,
  data_table_name => 'mining_data_build_v',
  case_id_column_name => 'cust_id',
  target_column_name => 'affinity_card',
  settings_table_name => 'svm_settings');
-- Apply model:
DBMS_DATA_MINING.APPLY(
  model_name => 'svm_model',
  data_table_name => 'mining_data_apply_v',
  case_id_column_name => 'cust_id',
  result_table_name => 'svm_apply_result');
```

Interface: no SQL

- **Let user write wl … R, Python, etc.)**
  - Provide a libra … ns implemented by the underlying
  - SQL undernea … *QL* [SIGMOD'17], *Oracle R Ente*
  - Other "big-da … CDE'11,VLDB'16], *Spark R*, *Mah*

```
invGamma = externalFunction(...)...
invGaussian = externalFunction(...)...
multiNormal = externalFunction(...)...

X = read("test_data/xb.bin", format="binary")
y = read("test_data/yb.bin", format="binary")
y_avg = avg(y)
y = y - y_avg

# compute the matrix X'X, and X'Y
XX = t(X) %*% X
XY = t(X) %*% y

# number of data points and number of features
n = nrow(X)
m = ncol(X)

shape_prior = 1.0
scale_prior = 1.0
mean_prior = matrix(1.0, rows=1, cols=m)
sigma2= invGamma(shape_prior, scale_prior)
tau= invGaussian(mean_prior, shape_prior)

niter = 5

for (i in 1:niter) {

  A = XX + diag(t(tau))
  A_inv = inv(A)
  mu = A_inv %*% XY
  covariance = A_inv * sigma2
  beta = multiNormal(t(mu), covariance)
  remain_sum1 = (t(y) - beta %*% t(X))
    (y - X %*% t(beta)) / 2.0
  remain_sum2 = (beta * beta) %*% t(tau) / 2.0
  scale_m = 1.0 + remain_sum1 + remain_sum2
  scale = as.scalar(scale_m[1,1])
  shape = 1.0 + (n-1.0)/2.0 + m/2.0
  sigma2 = invGamma(shape, scale)
  tau_mu = sqrt(sigma2 / (beta * beta))
  tau = invGaussian(tau_mu, 1.0)
}
```

Bayesian LASSO in *B* … in *SystemML*

(Examples from *BUDS/SimSQL* [SIGMOD'17])

Summary



- **You can get a lot of mileage for machine learning with SQL+UDF (octopus + hammer)**
- **Deep roots in**
  - DBMS extensibility research
  - Array DBMS, e.g., ***SciDB*** **[CSE'13]**; see **Rusu & Cheng [arXiv 2013]** for survey
- **Next: more opportunities for deeper ML+DB integration**

http://www.bongpages.com/index.php?route=product/product&product_id=115

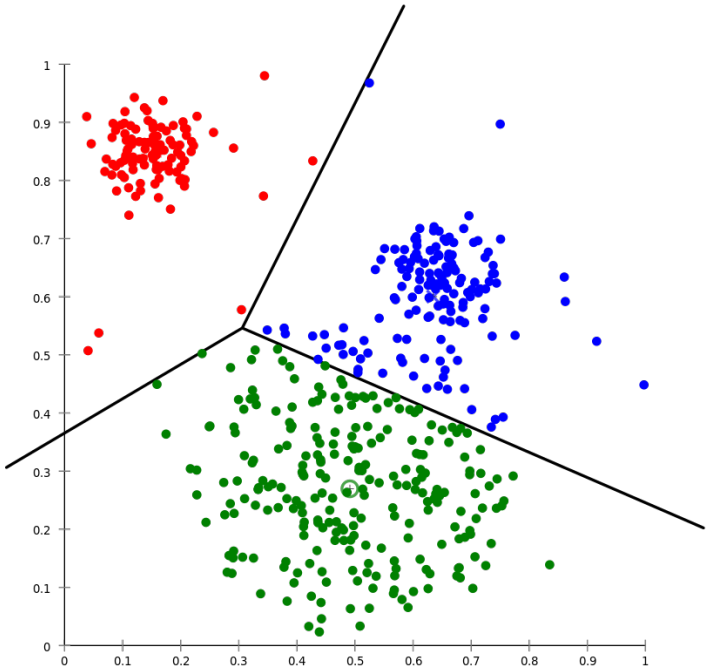
References for Part 2: ML with SQL & UDF

- ***Bismarck*** **[SIGMOD'12]** Feng et al. "Towards a Unified Architecture for in-RDBMS Analytics." SIGMOD 2012
- ***BUDS/SimSQL*** **[SIGMOD'17]** Gao et al. "The BUDS Language for Distributed Bayesian Machine Learning." SIGMOD 2017
- ***Cumulon*** **[SIGMOD'13]** Huang et al. "Cumulon: optimizing statistical data analysis in the cloud." SIGMOD 2013
- ***GLADE*** **[LADIS'11]** Rusu & Dobra. "GLADE: A Scalable Framework for Efficient Analytics." LADIS 2011
- ***GLADE*** **[SIGMOD'12]** Cheng et al. "GLADE: Big Data Analytics Made Easy." SIGMOD 2012
- ***MAD Skills*** **[VLDB'09]** Cohen et al. "MAD skills: new analysis practices for big data." PVLDB 2(2), 2009
- ***MADlib*** **[VLDB'12]** Hellerstein et al. "The MADlib Analytics Library or MAD Skills, the SQL." PVLDB 5(12), 2012
- ***RIOT-DB*** **[CIDR'09]** Zhang et al. "RIOT: I/O-efficient numerical computing without SQL." CIDR 2009
- ***RIOT*** **[ICDE'10]** Zhang et al. "I/O-efficient statistical computing with RIOT." ICDE 2010
- **Rusu & Cheng [arXiv 2013]** Rusu & Cheng. "A Survey on Array Storage, Query Languages, and Systems." https://arxiv.org/abs/1302.0103
- ***ScalOps*** **[DeBull'12]** Borkar et al. "Declarative systems for large-scale machine learning." IEEE Data Eng. Bulletin, 35(2), 2012
- ***SciDB*** **[CSE'13]** "SciDB: A Database Management System for Applications with Complex Analytics." Comp. Sci. Eng. 15(3), 2013
- ***SimSQL*** **[SIGMOD'13]** Cai et al. "Simulation of Database-Valued Markov Chains Using SimSQL." SIGMOD 2013
- ***SimSQL*** **[ICDE'17]** Luo et al. "Scalable Linear Algebra on a Relational Database System." ICDE 2017
- ***SystemML*** **[ICDE'11]** Ghoting et al. "SystemML: Declarative machine learning on MapReduce." ICDE 2011
- ***SystemML*** **[VLDB'16]** Boehm et al. "SystemML: Declarative machine learning on Spark." PVLDB 9(13), 2016

Part 2 Backup/Extra Slides

k-Means Clustering

- **Given n points, find k centroids to minimize sum of squared distances between each point and its closest centroid**



- **EM-style iterative algorithm:**
  1. Pick initial k candidate centroid locations
  2. Assign each point to the closest candidate
  3. Reposition each candidate as the centroid of its assigned points
  4. Repeat 2-3 above until assignment changes no more (or very little)

https://commons.wikimedia.org/wiki/File:KMeans-Gaussian-data.svg

k-Means as UDA

- **State: k candidates with locations + cluster info $\{\langle \text{loc}_i, \text{sum}_i, \text{cnt}_i \rangle\}_{1 \le i \le k}$**
- **`Init`: given centroid locations, with sum and count of 0**
- **`Accumulate`: given a data point p, find the candidate i closest to p; increment sum_i by p's coordinates and cnt_i by one**
- **`Merge`: merge $\langle \text{loc}, \text{sum}, \text{cnt} \rangle$ records by loc; add sum and cnt**
- **`Finalize`: for each i, compute new loc_i as $\text{sum}_i / \text{cnt}_i$**
- **One SQL query with this UDA gives one iteration of the EM algorithm**
  - For the next iteration, the UDA will be initialized with the k locations computed from the previous
  - Can use a UDF to drive overall iterations
  - Termination condition can be evaluated in SQL too (see *MADlib*)

***GLADE* [LADIS'11,SIGMOD'12]**
***MADlib* [VLDB'12]**

Markov-Chain Monte-Carlo (MCMC)

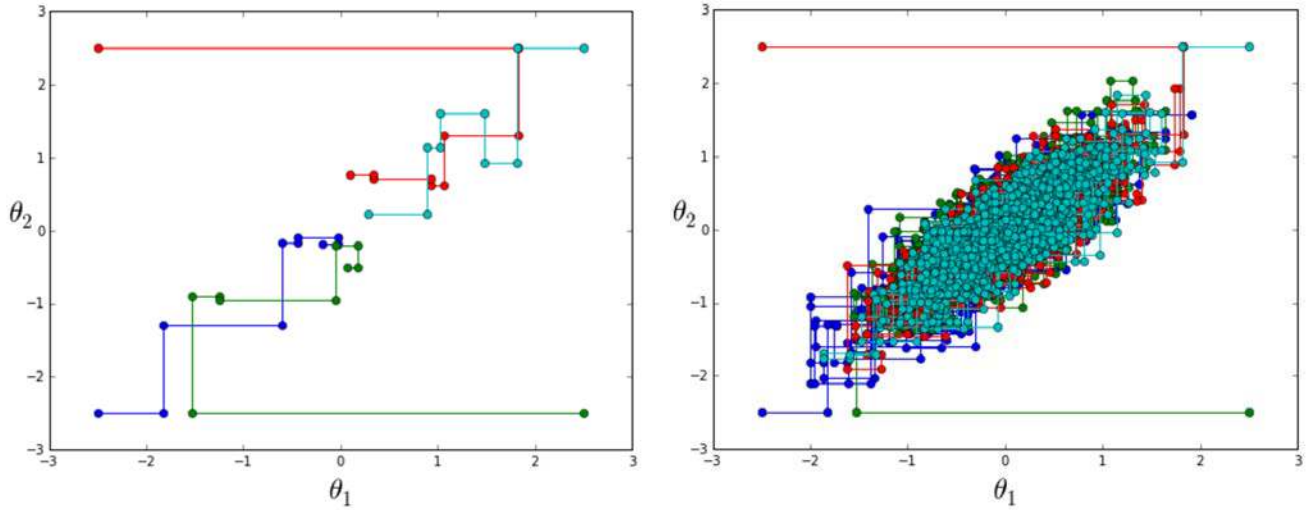
- **Bayesian ML comes down to analyzing the "posterior" distribution**

  P(parameters, hidden variables | observations)

- **Direct analysis is often hard, so we use Monte-Carlo simulation**
  - Repeatedly sample from the posterior, and analyze the samples
- **But sampling directly from the posterior is often hard, so we use MCMC**
  - A sampler generates a Markov chain of samples, whose stationary distribution is the target posterior

Example: Gibbs Sampling

- **Suppose we have an n-variate distribution, but the conditional distributions are easier to sample from**
- **Begin with some initial sample $\mathbb{z}^{(0)}$**
- **For the $(t+1)$-th sample $\mathbb{z}^{(t+1)}$, sample each component $z_i^{(t+1)}$ conditioned on all other components sampled most recently, i.e., $p\left(z_i^{(t+1)} \middle| z_1^{(t+1)}, \ldots, z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, z_n^{(t)}\right)$**
- **Rinse and repeat**



https://ga7g08.github.io/2015/02/09/Gibbs-sampler-with-a-bivariate-normal-distribution/

MCMC in SimSQL

SimSQL [SIGMOD'13]

- **Think of each sample as a table (tables)**
- **Write UDF to define "VG" (value-generating) functions that draw samples**
- **Write SQL with VG functions to define how to generate T[t] (instance of table T in the t-th sample) from T[$t-1$]**
- **Write SQL to simulate multiple MCMC chains, and to compute compute distributional properties for variables of interest from T[t]'s across T's, t's, and chains**

☞**An example of staying true to the declarative roots of databases**

- But also need new techniques not in traditional DBMS, e.g.:
  - Plans are huge—cut them into "frames"; observe execution stats of last frame and to optimize the next
  - Use "tuple bundles" to instantiate/process multiple possible worlds simultaneously

Part 3: Learning Over Joins, SRL, and Deep RDBMS Integration

Arun Kumar

UC San Diego
La Jolla, CA, USA

SIGMOD 2017

Overview: Learning Over Joins

Problem: Many datasets are multi-table ⟷ ML toolkits assume single-table inputs → ML after joining tables



Overheads:
Extra storage
Computational redundancy
Join time
Maintenance headaches

Learning Over Joins: "Push Down" ML through joins

1) Over standard data systems: Orion, Santoku, Morpheus
2) Over a "factorized database" system: FDB-F
3) Special-purpose tools: libFM, TensorDB, Compressed ML

Related but orthogonal: Statistical relational learning (DeepDive, etc.)

Learning Over Joins

Over standard data systems: Orion, Morpheus, Santoku

Example: GLMs with gradient descent (GD)

$$L(w) = \sum_{i=1}^{n} f(w'x_i, y_i) \qquad \nabla L(w) = \sum_{i=1}^{n} g(w'x_i, y_i)x_i$$

$$w'x = w'_S x_S + w'_R x_R \qquad x = [x_S \; x_R]$$

$$T = S \bowtie R$$

Orion [SIGMOD'15]:
Introduced the scalable "factorized learning" idea
Easy UDA implementation on existing data systems (RDBMS/Hive/Spark)

Morpheus [VLDB'17]:
Generalizes factorized learning to any ML algorithm in *linear algebra*
"Push down" rewrites for matrix-vector mult., gramian, ginv, etc.

Santoku [VLDB'15]: Discrete features (Naive Bayes, trees, etc.)

Learning Over Joins

Over a “factorized database” system: **FDB-F [SIGMOD’16]**

Generalized semiring-based aggregates over “factorized joins”

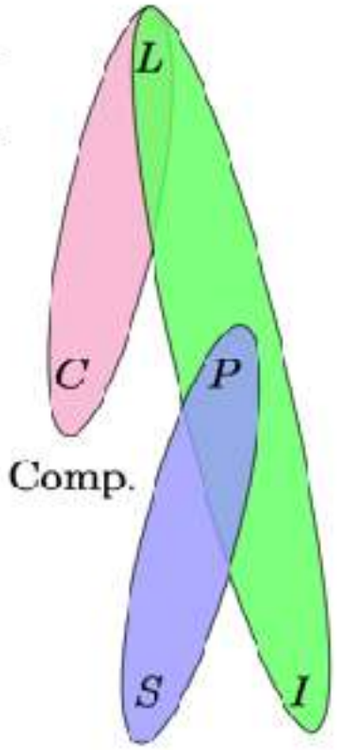
Sales	
P	S
p_1	s_1
p_1	s_2
p_2	s_3
p_2	s_4
p_3	s_5

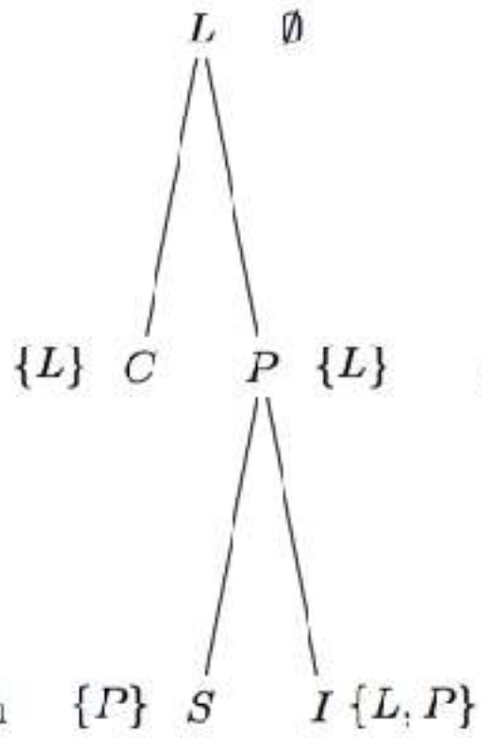
Branch		
L	P	I
l_1	p_1	i_1
l_1	p_1	i_2
l_1	p_2	i_3
l_2	p_2	i_4
l_2	p_3	i_5

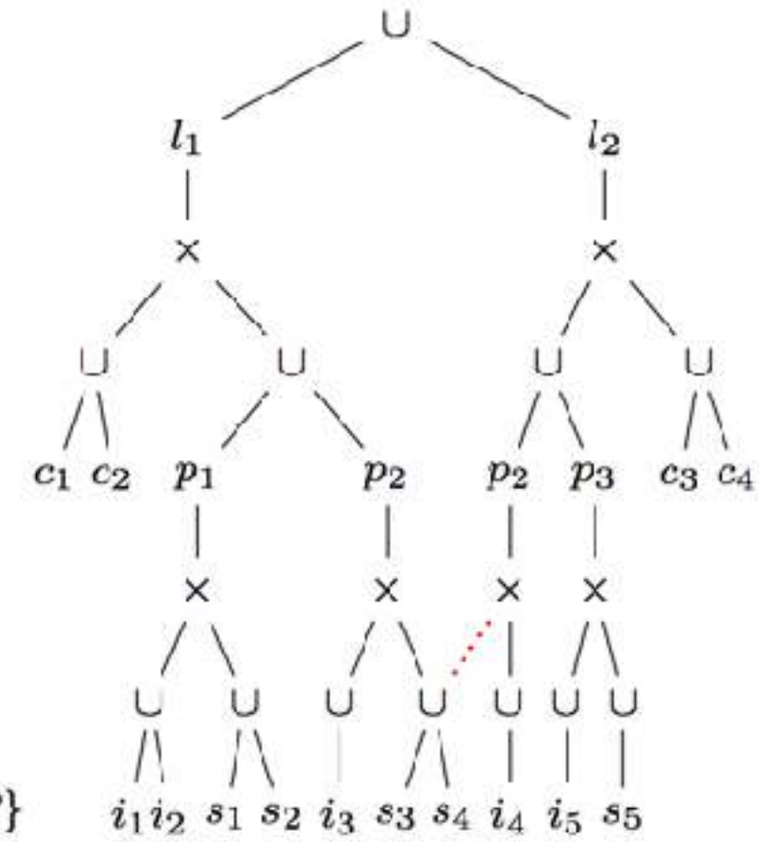
Competition	
L	C
l_1	c_1
l_1	c_2
l_2	c_3
l_2	c_4

Natural Join

L	C	P	I	S
l_1	c_1	p_1	i_1	s_1
l_1	c_1	p_1	i_1	s_2
l_1	c_1	p_1	i_2	s_1
l_1	c_1	p_1	i_2	s_2
l_1	c_1	p_2	i_3	s_3
l_1	c_1	p_2	i_3	s_4
.....				
above block for c_2				
.....				
l_2	c_3	p_2	i_4	s_3
l_2	c_3	p_2	i_4	s_4
l_2	c_3	p_3	i_5	s_5
.....				
above block for c_4				







SRL; Deep RDBMS Integration

SRL combines statistical learning with logic-based rules/constraints

"Non-IID" ML models
(MVDs, EMVDs, JDs)

NIPS'12 tutorial by Lise Getoor
Book with Ben Taskar

Inference and learning often perform joins internally!

Scalable grounding using RDBMS: **Tuffy [VLDB'10]**
Incremental maintenance: **IncrementalDeepDive [VLDB'15]**

Increasing interest in deeper integration of ML into DBMS kernel!

SAP HANA SLACID: Linear algebra kernels in an RDBMS **[SSDBM'14]**
New compressed sparse row/col. representations
Integrated API for basic access patterns and lin. alg. ops
OpenMP-based shared memory parallelism in DBMS task scheduler

References: Part 3

Columbus [SIGMOD'14]: Materialization Optimizations for Feature Selection Workloads
DeepDive [DataEng'14]: Feature Engineering for Knowledge Base Construction
FDB-F [SIGMOD'16]: Learning Linear Regression Models over Factorized Joins
IncrementalDeepDive [VLDB'15]: Incremental Knowledge Base Construction Using DeepDive
Morpheus [VLDB'17]: Towards Linear Algebra over Normalized Data
Orion [SIGMOD'15]: Learning Generalized Linear Models Over Normalized Data
Santoku [VLDB'15]: Demonstration of Santoku: Optimizing Machine Learning over Normalized Data
SLACID [SSDBM'14]: SLACID - Sparse Linear Algebra in a Column-Oriented In-Memory Database System
Tuffy [VLDB'10]: Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS

Backup Slides

Statistical Relational Learning Systems

Captures logical dependencies between between entities/variables

"Non-IID" ML models
(MVDs, EMVDs, JDs)

PODS tutorial by Lise Getoor on Tue!
(also NIPS'12; book with Taskar)

Example: Markov Logic Network (MLN); used by DeepDive

Schema		Λ Markov Logic Program		Evidence
paper(PaperID, URL)	weight	rule		wrote('Joe', 'P1')
wrote(Author, Paper)	5	cat(p, c1), cat(p, c2) => c1 = c2	(F_1)	wrote('Joe', 'P2')
refers(Paper, Paper)	1	wrote(x, p1), wrote(x, p2), cat(p1, c) => cat(p2, c)	(F_2)	wrote('Jake', 'P3')
cat(Paper, Category)	2	cat(p1, c), refers(p1, p2) => cat(p2, c)	(F_3)	refers('P1', 'P3')
	+∞	paper(p, u) => ∃x. wrote(x, p)	(F_4)	cat('P2', 'DB')
	-1	cat(p, 'Networking')	(F_5)	...

MLN inference (MAP) computes "most probable world" by plugging values of variables to predict

Grounding + Search

Involves joins!

Scalable grounding using RDBMS: **Tuffy [VLDB'10]**

Scalable Gibbs sampling: **Elementary [SIGMOD'13]**

Incremental maintenance: **IncrementalDeepDive [VLDB'15]**

Deep RDBMS Integration

Integrating linear algebra kernels into an RDBMS: SAP HANA

SLACID [SSDBM'14]: Mutable columnar layout for sparse matrices

Compressed sparse row/col. representation + incr. delta

Integrated API for basic access patterns and lin. alg. ops

OpenMP-based shared memory parallelism in DBMS task scheduler

Time series-specific systems: Fa, F2DB

Fa [VLDB'07]: "Declarative forecasting" queries for time series

Projection and shift-based time series feature transformations

Feature ranking and subset selection heuristics

Lin. reg., Bayesian networks, SVM, CART, Random Forest

Both one-time and continuous forecasting

Part 4: Rewrites, Operator Selection, and Operator Fusion

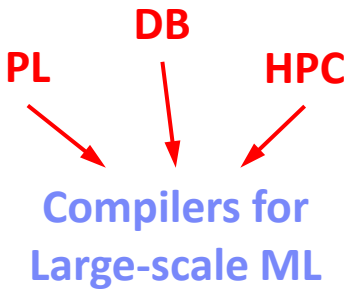
Matthias Boehm

IBM Research – Almaden
San Jose, CA, USA

SIGMOD 2017

Overview Optimizing Compilers for ML Algorithms

- **Comparison Query Optimization**
  - Rule- and cost-based rewrites and operator ordering
  - Physical operator selection and query compilation
  - Linear algebra / other ML operators, DAGs, control flow, sparse/dense formats



- **#1 Interpretation** (operation at-a-time)
  - Examples: Morpheus [PVLDB'17]
- **#2 Lazy Expression Compilation** (DAG at-a-time)
  - Examples: RIOT [CIDR'09], Mahout Samsara [MLSystems'16]
  - Examples w/ control structures: Weld [CIDR'17], OptiML [ICML'11], Emma [SIGMOD'15]
- **#3 Program Compilation** (entire program)
  - Examples: SystemML [PVLDB'16], Cumulon [SIGMOD'13], Tupleware [PVLDB'15]

Optimization Scope

```
1:  X = read($1); # n x m matrix
2:  y = read($2); # n x 1 vector
3:  maxi = 50; lambda = 0.001;
4:  intercept = $3;
5:  ...
6:  r = -(t(X) %*% y);
7:  norm_r2 = sum(r * r); p = -r;
8:  w = matrix(0, ncol(X), 1); i = 0;
9:  while(i<maxi & norm_r2>norm_r2_trgt)
10: {
11:    q = (t(X) %*% X %*% p)+lambda*p;
12:    alpha = norm_r2 / sum(p * q);
13:    w = w + alpha * p;
14:    old_norm_r2 = norm_r2;
15:    r = r + alpha * q;
16:    norm_r2 = sum(r * r);
17:    beta = norm_r2 / old_norm_r2;
18:    p = -r + beta * p; i = i + 1;
19: }
20: write(w, $4, format="text");
```

Logical Simplification Rewrites

- **Traditional PL Rewrites** (e.g., TensorFlow, OptiML, SystemML)
  - CSE, constant folding, branch removal

- **Algebraic Simplification Rewrites** (e.g., SystemML, FAQ [PODS'16])
  - `t(X) %*% y` → `t(t(y) %*% X)`
  - `trace(X %*% Y)` → `sum(X * t(Y))`
  - `sum(X + Y)` → `sum(X) + sum(Y)`
  - `sum(X^2)` → `t(X) %*% X, iff ncol(X)=1`

- **Loop Vectorization** (e.g., OptiML, SystemML)

```
for(i in a:b)
   X[i,1] = Y[i,2] + Z[i,1]
```
→
```
X[a:b,1] = Y[a:b,2] + Z[a:b,1]
```

- **Incremental Computations**
  - Delta update rules (e.g., **LINVIEW** [SIGMOD'14], factorized [CoRR'17])
  - Incremental iterations (e.g., Flink)
  - Update-in-place (e.g., SystemML)

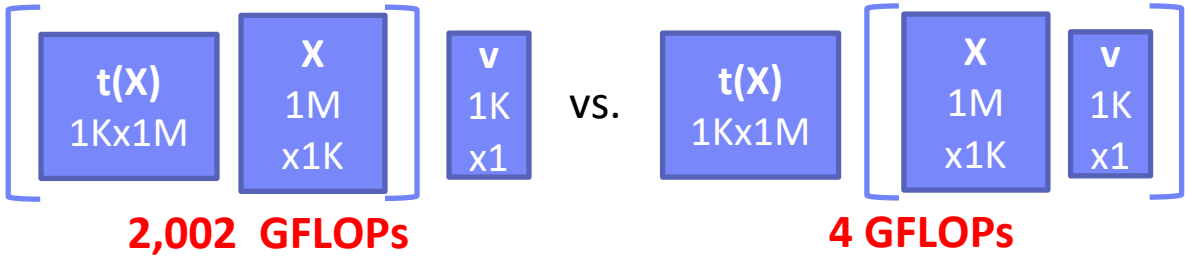
```
A = t(X) %*% X + t(ΔX) %*% ΔX
b = t(X) %*% y + t(ΔX) %*% Δy
```

Logical Simplification Rewrites
Matrix Multiplication Chain Optimization

- **Optimization Problem**
  - Matrix multiplication chain of n matrices M_1, M_2, …M_n (associative)
  - Optimal parenthesization of the product $M_1 M_2 \ldots M_n$

Example

t(X) %*% X %*% v

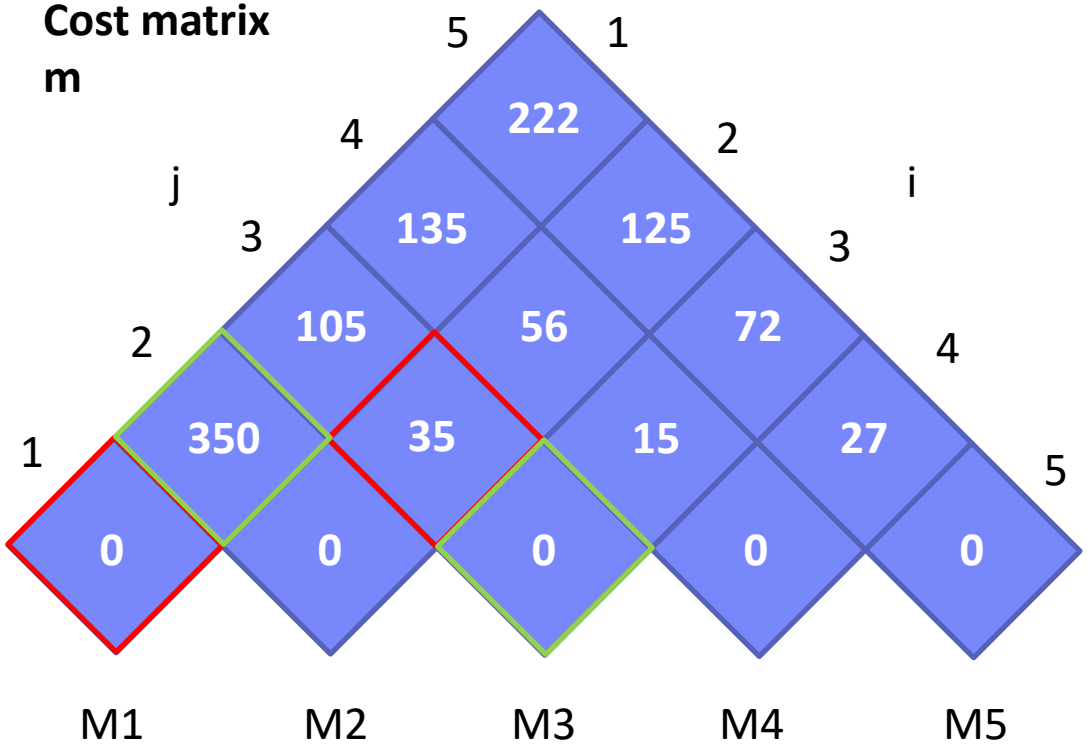


- **Search Space Characteristics**
  - Naïve exhaustive: Catalan numbers → $\Omega(4^n / n^{3/2})$
  - DP applies: (1) optimal substructure, (2) overlapping subproblems
  - Textbook DP algorithm [MIT Press'09]: $\Theta(n^3)$ time, $\Theta(n^2)$ space
    - Examples: SystemML [Data Eng. Bull. '14], RIOT (including I/O costs), SpMachO (including sparsity for intermediates) [EDBT'15],
  - Best known algorithm: O(n log n)

Matrix Multiplication Chain Optimization

M1	M2	M3	M4	M5
10x7	7x5	5x1	1x3	3x9

Cost matrix m



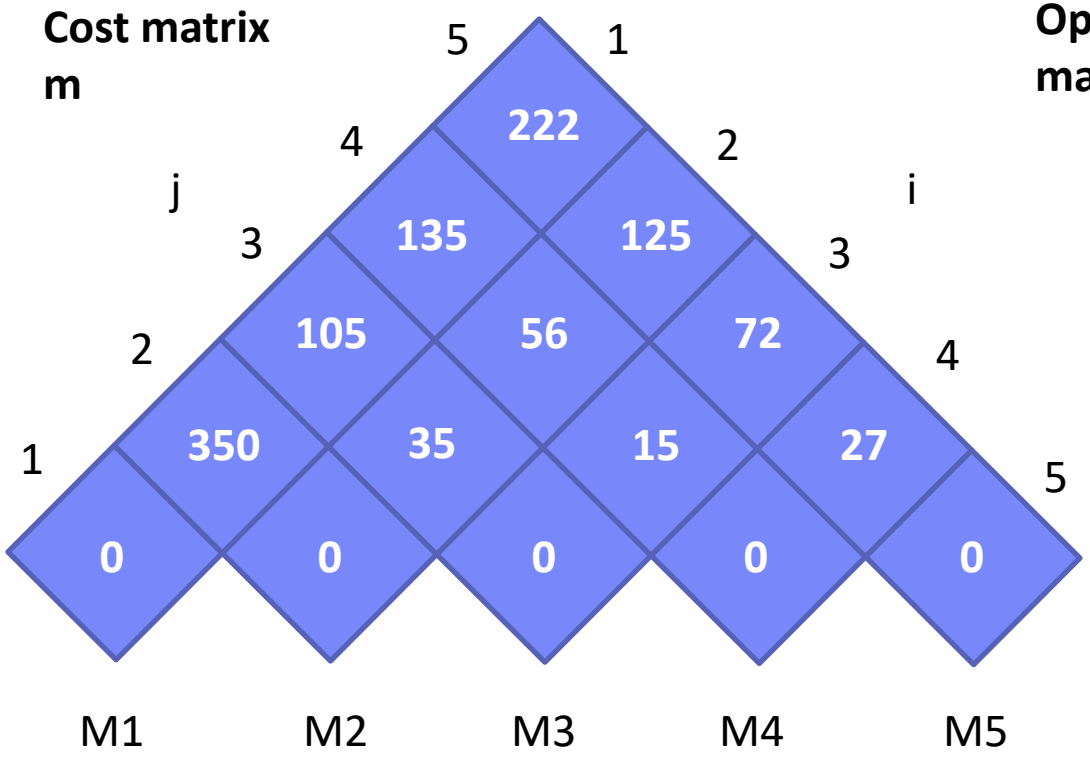
m[1,3] = min(
m[1,1] + m[2,3] + p1p2p4,
m[1,2] + m[3,3] + p1p3p4)

= min(= min(
0 + 35 + 10*7*1, **105,**
350 + 0 + 10*5*1) 400)

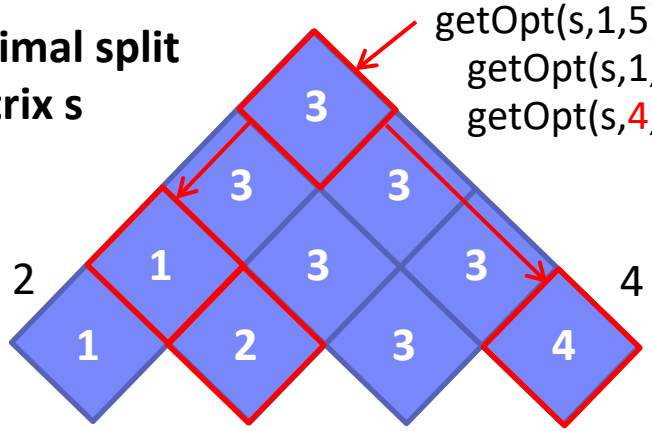
Matrix Multiplication Chain Optimization

M1	M2	M3	M4	M5
10x7	7x5	5x1	1x3	3x9

Cost matrix m



Optimal split matrix s



(M1 M2 M3 M4 M5)

((M1 M2 M3) (M4 M5))

((M1 (M2 M3)) (M4 M5))

➔ **((M1 (M2 M3)) (M4 M5))**

➔ **Open questions: DAGs; other operations, joint opt w/ rewrites, CSE, fusion, and physical operators**

Physical Rewrites and Optimizations

- **Distributed Caching**
  - Redundant compute vs. memory consumption and I/O
  - **#1 Cache intermediates w/ multiple refs** (Emma)
  - **#2 Cache initial read and read-only loop vars** (SystemML)
- **Partitioning**
  - Many frameworks exploit co-partitioning for efficient joins
  - **#1 Partitioning-exploiting operators** (SystemML, Emma, Samsara)
  - **#2 Inject partitioning to avoid shuffle per iteration** (SystemML)
  - **#3 Plan-specific data partitioning** (SystemML, Dmac [SIGMOD'15], Kasen [PVLDB'16])
- **Other Data Flow Optimizations** (Emma)
  - **#1 Exists unnesting** (e.g., filter w/ broadcast → join)
  - **#2 Fold-group fusion** (e.g., groupByKey → reduceByKey)
- **Physical Operator Selection**

Physical Operator Selection

- **Common Selection Criteria**
  - **Data and cluster characteristics** (e.g., data size/shape, memory, parallelism)
  - **Matrix/operation properties** (e.g., diagonal/symmetric, sparse-safe ops)
  - **Data flow properties** (e.g., co-partitioning, co-location, data locality)
- **#0 Local Operators**
  - SystemML mm, tsmm, mmchain; Samsara/Mllib local linalg
- **#1 Special Operators** (often fused operators)
  - Special patterns (SystemML **tsmm**, tsmm2, mapmmchain, pmm; Samsara AtA
  - Sparsity exploiting (SystemML wdivmm, **wsloss**, wcemm; Cumulon maskMult)
- **#2 Broadcast-Based Operators** (aka broadcast join)
  - SystemML **mapmm, mapmmchain**
- **#3 Co-Partitioning-Based Operators** (aka improved repartition join)
  - SystemML zipmm; Emma, Samsara OpAtB
- **#4 Shuffle-Based Operators** (aka repartition join)
  - SystemML cpmm, rmm; Samsara OpAB

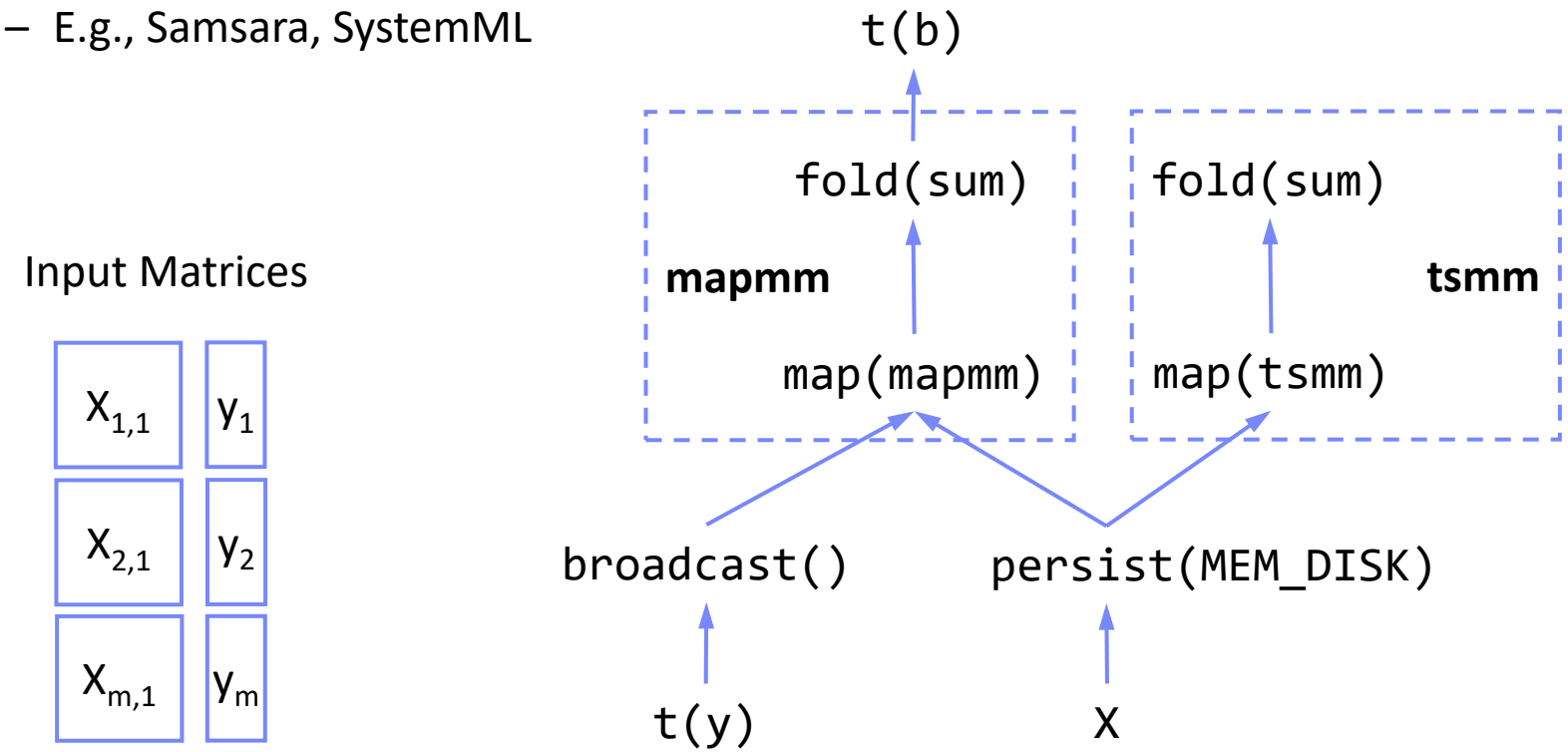
Selection Preference



Example Physical Operators

- **Example Linear Regression Direct Solve**
  - Transpose-self for t(X)%*%X
  - Broadcast-based for t(X) %*% y
  - Logical and physical rewrites
  - E.g., Samsara, SystemML

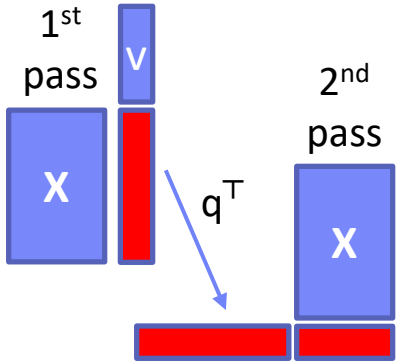
```
A = t(X) %*% X
b = t(X) %*% y
w = solve(A, b)
```



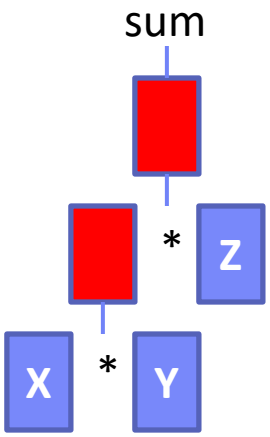
Fused Operators

- **Motivation**
  - Problem: **Memory-bandwidth-bound operations** (I/O)
  - Goal: Reduce number of **scans and intermediates**

- **Matrix-Vector Chains:** `t(X) %*% (X%*%v)`
  - Fused single-pass operator: **mmchain** [PPoPP'15]
  - Row-aligned creation/consumption



- **Ternary Aggregates:** `sum(X*Y*Z)`
  - Fused aggregation operator
  - Avoid materialized intermediates

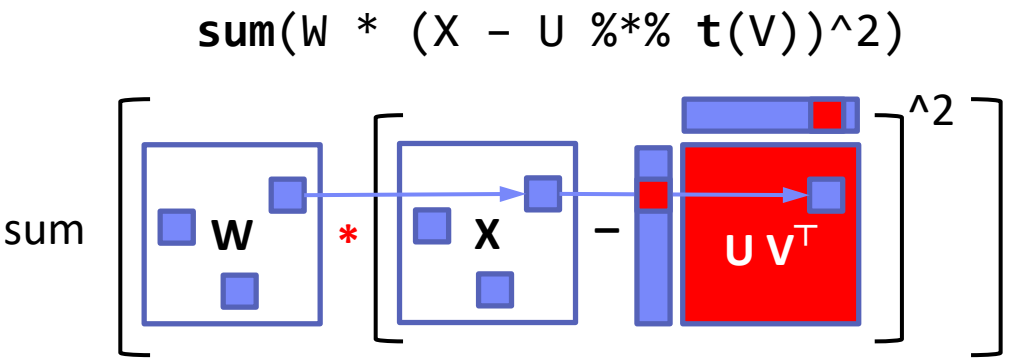


- **Other ML-Specific Operators**
  - Sample proportion: `X * (1-X)`
  - Sigmoid: `1 / (1 + exp(-X))`
  - Axpy: `X + s*Y`, `X - s*Y`

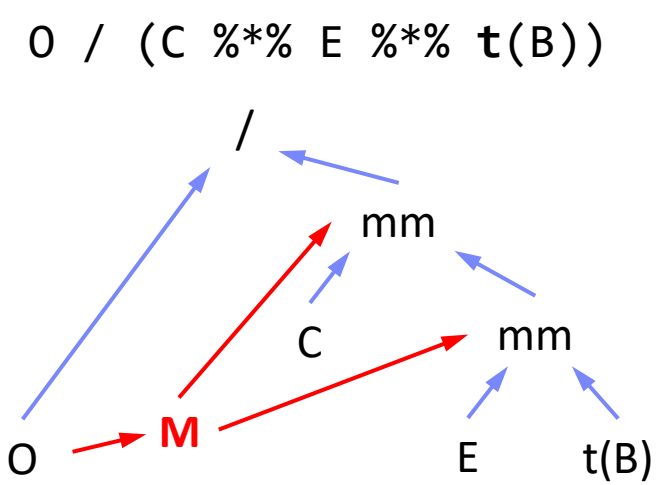
Sparsity-Exploiting Fused Operators

- **Goal:** Avoid dense intermediates and unnecessary computation

- **#1 Fused Physical Operators**
  - E.g., SystemML [PVLDB'16] wsloss, wcemm, wdivmm
  - Selective computation over non-zeros of "sparse driver"



- **#2 Masked Physical Operators**
  - E.g., Cumulon MaskMult [SIGMOD'13]
  - Create mask of "sparse driver"
  - Pass mask to single masked matrix multiply operator



➔ **Open questions: NaN handling, automatic operator fusion (codegen)**

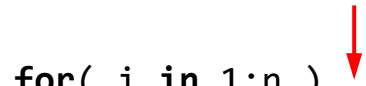
Automatic Operator Fusion

- **Motivation**
  - Large development effort for hand-coded fused operators
  - UDF-centric systems w/o pre-defined operators

- **General Approach: Fuse by Access Pattern**
  - **#1 Loop fusion** (OptiML, Tupleware, Weld, TensorFlow XLA [github'17])
  - **#2 Templates** (Kasen, SPOOF [CIDR'17])
  - Scope: expression or program compilation

```
R = (A + s*B) * C
```

```
for( i in 1:n )
   tmp[i] = s*B[i]
for( i in 1:n )
   tmp[i] = A[i]+tmp[i]
for( i in 1:n)
   tmp[i] = tmp[i]*C[i]
```



```
for( i in 1:n )
   tmp[i] = (A[i]+s*B[i]) * C[i]
```

- **Additional Techniques**
  - Tupleware: **Micro optimizations** (tile-at-a-time, predicates, result allocation)
  - Weld: **Cross-library optimizations** (via common IR of basic operations)
  - SystemML-SPOOF: **sparsity-exploiting fused operators**

➔ **Open question: Optimization of fusion plans for DAGs (redundant compute vs materialization, access patterns)**

Runtime Adaptation (see AQP)

- **Problem of Unknown/Changing Size Information**
  - Dimensions/sparsity required for **cost comparisons/valid plans**
  - Unknowns → **conservative fallback plans**
- **Challenges**
  - Conditional control flow, function call graphs, UDFs
  - Data-dependent ops (e.g., sampling, group by classes, output sparsity)
  - Computed size expressions, changing dimensions/sparsity
- **Approaches**
  - **#1 Lazy expression optimization** (RIOT, OptiML, Emma, Weld, Samsara)
    - Optimize on triggering actions (unconditional scope)
  - **#2 Dynamic inter-DAG recompilation** (SystemML)
    - Split/mark DAGs, recompile DAGs/functions w/ exact stats

→ **Open questions:**

  - **Estimating the size and sparsity of intermediates**
  - **Adaptive query processing and storage**

References for Part 4

- T. C. Hu and M. T. Shing: Computation of Matrix Chain Products. Part II. SIAM J. Comput. 13(2): 228-251, 1984.
- T. H. Cormen, et al. Introduction to Algorithms, Third Edition, The MIT Press, pages 370-377, 2009.
- Y. Zhang et al. RIOT: I/O-Ecient Numerical Computing without SQL. In CIDR, 2009.
- A. K. Sujeeth et al. OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning. In ICML, 2011.
- S. Ewen et al. Spinning Fast Iterative Data Flows. PVLDB, 5(11), 2012.
- B. Huang et al. Cumulon: Optimizing Statistical Data Analysis in the Cloud. In SIGMOD, 2013.
- M. Nikolic et al. LINVIEW: Incremental View Maintenance for Complex Analytical Queries. In SIGMOD, 2014.
- M. Boehm et al. SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. IEEE Data Eng. Bull., 37(3), 2014.
- D. Kernert et al. SpMacho - Optimizing Sparse Linear Algebra Expressions with Probabilistic Density Estimation. In EDBT, 2015.
- A. Ashari et al.: On optimizing machine learning workloads via kernel fusion. In PPoPP, 2015.
- A. Alexandrov et al. Implicit Parallelism through Deep Language Embedding. In SIGMOD, 2015.
- Lele Yu et al. Exploiting Matrix Dependency for Efficient Distributed Matrix Computation. In SIGMOD, 2015.
- M. Abo Khamis et al.: FAQ: Questions Asked Frequently. In PODS, 2016.
- A. Crotty et al. An Architecture for Compiling UDF-centric Workflows. PVLDB, 8(12), 2015.
- M. Boehm et al. SystemML: Declarative Machine Learning on Spark. PVLDB, 9(13), 2016.
- M. Zhang et al. Measuring and Optimizing Distributed Array Programs. PVLDB, 9(12), 2016.
- S. Schelter et al. Samsara: Declarative Machine Learning on Distributed Dataflow Systems. NIPS Workshop MLSystems, 2016.
- T. Elgamal et al. SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning. In CIDR, 2017.
- S. Palkar et al. Weld: A Common Runtime for High Performance Data Analysis. In CIDR 2017.
- TensorFlow XLA, https://www.tensorflow.org/performance/xla/, 2017.
- M. Nikolic and D. Olteanu: Incremental Maintenance of Regression Models over Joins, CoRR, 2017.
- L. Chen et al. Towards Linear Algebra over Normalized Data. PVLDB, to appear, 2017.

Part 5: Compression, Scan Sharing, and Index Structures

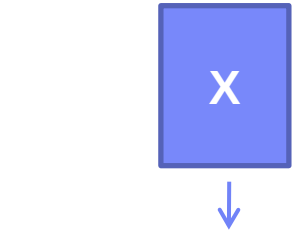
Matthias Boehm

IBM Research – Almaden
San Jose, CA, USA

SIGMOD 2017

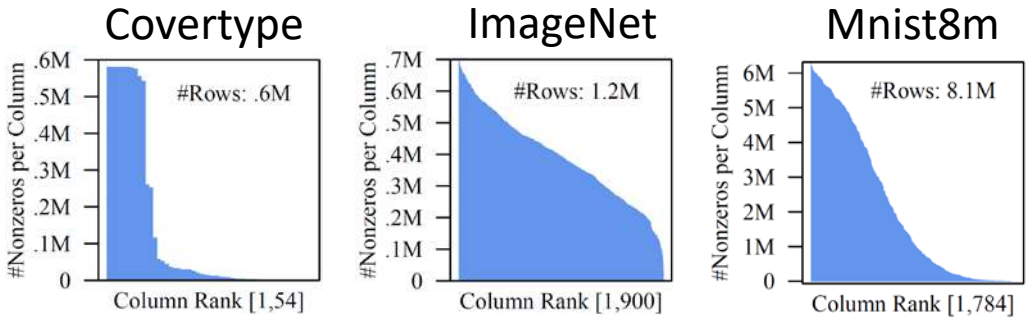
Motivation: Workload Characteristics

- **Memory-Bandwidth-Bound Operations**
  - Iterative ML algorithms w/ read-only data access
  - **#1:** I/O-bound matrix vector products
    ➔ **Crucial to fit matrix into memory**
    (single node, distributed, GPU)
    ➔ **Avoid unnecessary scans**
  - **#2:** Matrix and vector intermediates
    ➔ **Reduce number of reads and writes**




```
while(!converged) {
  … q = X %*% v …
}
```

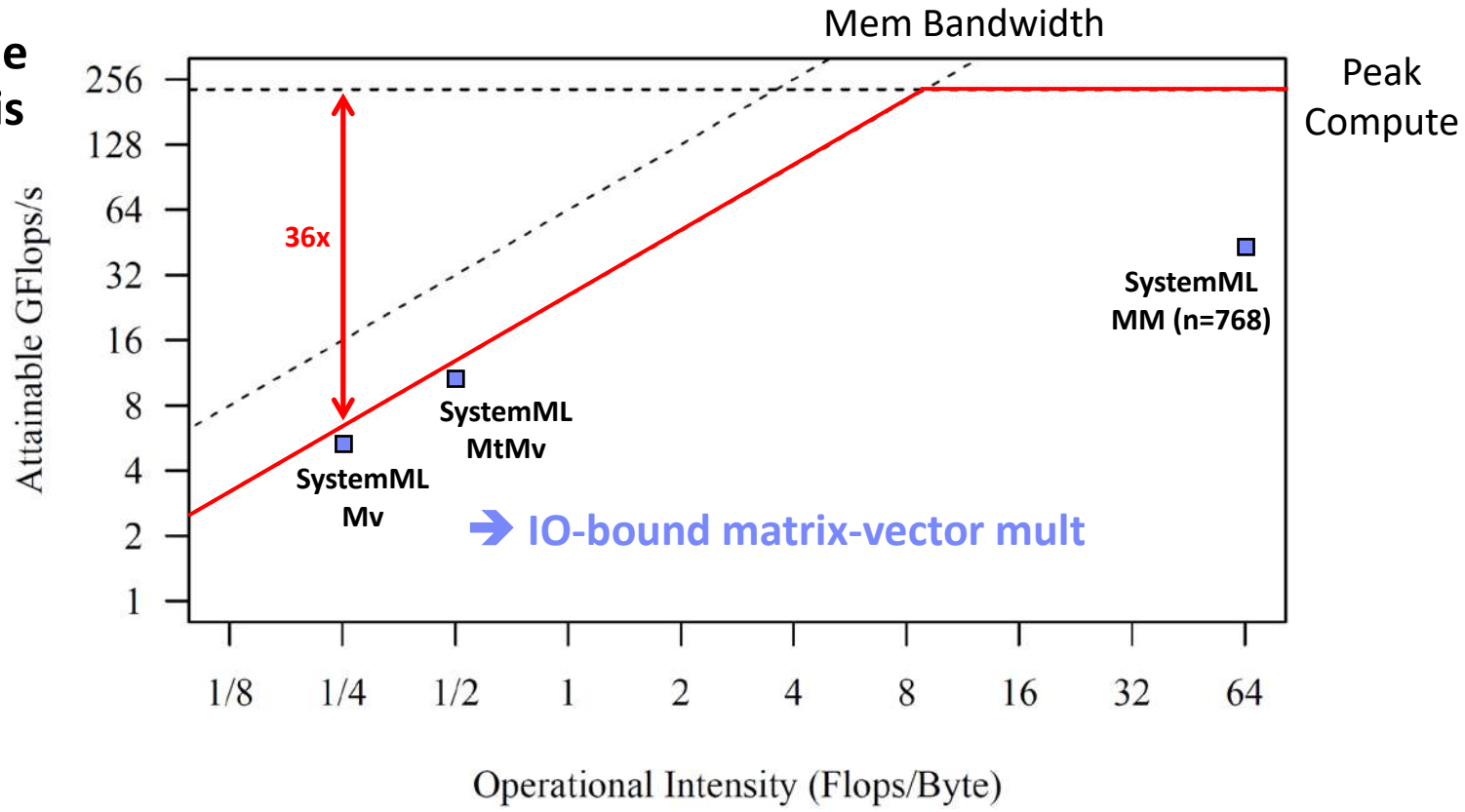
- **Common Data Characteristics**
  - Tall & skinny matrices (#row >> #columns)
  - Non-uniform sparsity
  - Low column cardinality
  - Column correlations



Motivation: Workload Characteristics

- **Single Node:** 2x6 E5-2440 @2.4GHz–2.9GHz, DDR3 RAM @1.3GHz (ECC)
  - Peak memory bandwidth: **2 x 32GB/s** (local), **2 x 12.8GB/s** (remote QPI)
  - Peak compute bandwidth: **2 x 115.2GFlops/s**

- **Roofline Analysis**



Background: Block Partitioning and Layouts

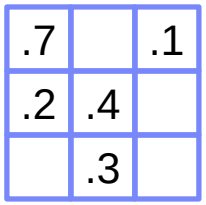
- **Blocked Matrix Representations**
  - Blocks, a.k.a. “tiles”, “chunks”, or “pages”
  - **#1 Logical (fixed-size) blocking** (→ var. physical size)
  - **#2 Physical blocking** (→ fixed physical size)
  - Blocks encoded independently (dense/sparse)
  - Local matrices → single block

Logical blocking
3,400x2,700 matrix
(w/ B_c=1,000)



- **Common Block Representations**
  - **Dense** (linearized arrays)
  - **CSR** (compressed sparse rows)
  - **CSC** (compressed sparse columns)
  - MCSR (modified CSR)
  - **COO** (Coordinate matrix)
  - …

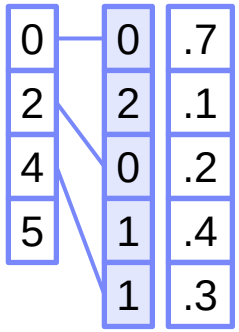
Example
3x3 Matrix



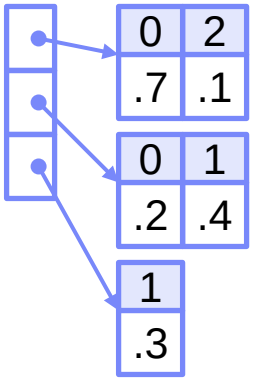
Dense (row-major)



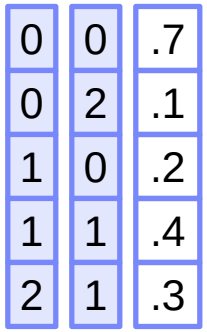
CSR



MCSR

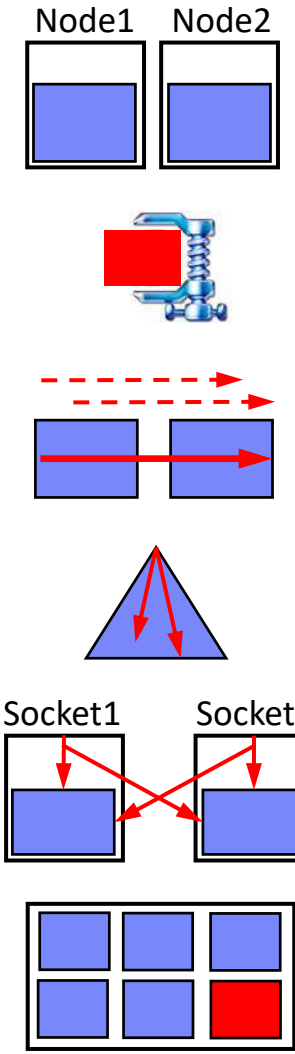


COO



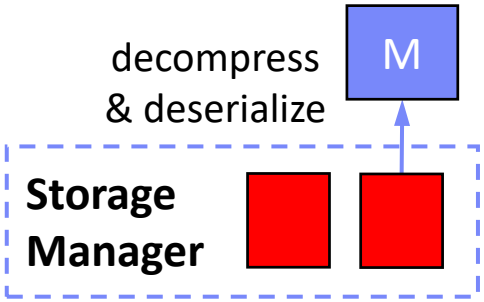
Overview Techniques for Data-Intensive Machine Learning

- **#1 (Distributed) Caching**
  - Keep read only feature matrix in (distributed) memory
- **#2 Compression**
  - Fit larger datasets into available memory
- **#3 Scan Sharing (and operator fusion)**
  - Reduce the number of scans as well as read/writes
- **#4 Index Structures**
  - Out-of-core data, I/O-aware ops, updates
- **#5 NUMA-Aware Partitioning and Replication**
  - Matrix partitioning / replication → data locality
- **#6 Buffer Pool Management**
  - Graceful eviction of intermediates, out-of-core ops

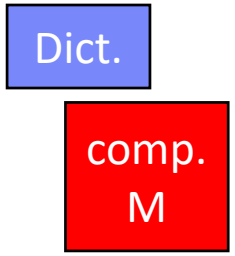


Compression Techniques

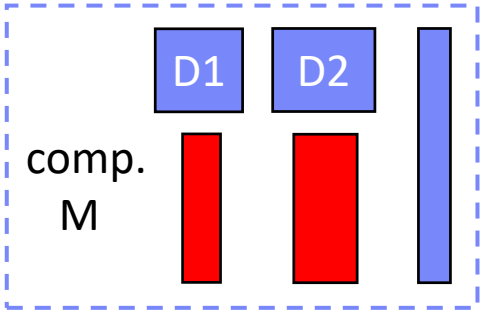
- **#1 Block-Level General-Purpose Compression**
  - Heavyweight or lightweight compression schemes
  - Decompress matrices block-wise for each operation
  - E.g.: Spark RDD compression (Snappy/LZ4), SciDB SM [SSDBM'11], TileDB SM [PVLDB'16]



- **#2 Block-Level Matrix Compression**
  - Compress matrix block with common encoding scheme
  - Perform LA ops over compressed representation
  - E.g.: CSR-VI (dict) [CF'08], cPLS (grammar) [KDD'16], TOC (LZW w/ trie) [CoRR'17]

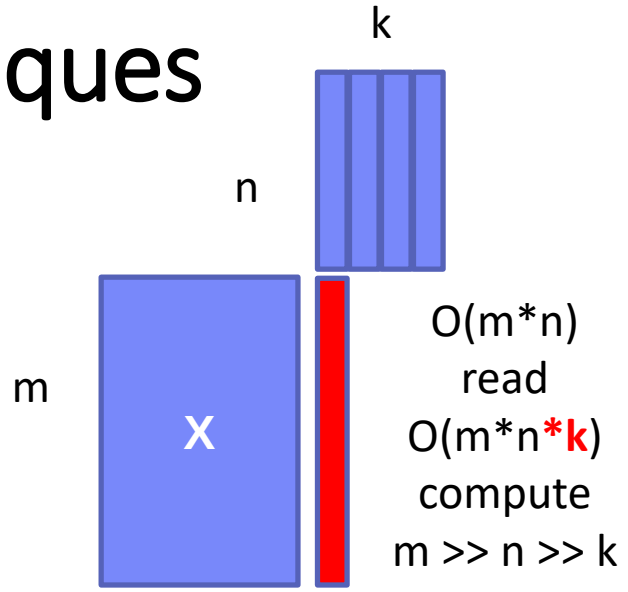


- **#3 Column-Group-Level Matrix Compression**
  - Compress column groups w/ heterogenous schemes
  - Perform LA ops over compressed representation
  - E.g.: SystemML CLA (RLE, OLE, DDC, UC) [PVLDB'16]



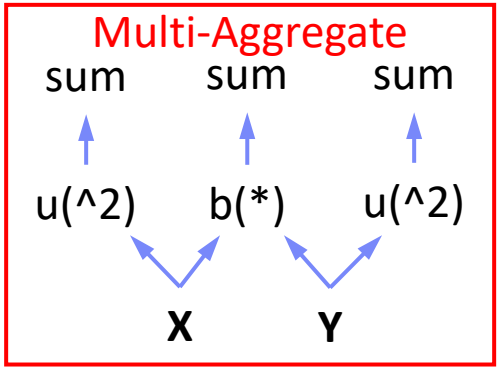
Scan Sharing Techniques

- **#1 Batching**
  - One-pass evaluation of multiple configurations
  - Use cases: EL, CV, feature selection, hyper parameter tuning
  - E.g.: **TUPAQ** [SoCC'16], **Columbus** [SIGMOD'14]



- **#2 Fused Operator DAGs**
  - Avoid unnecessary scans, (e.g., part 4 mmchain)
  - Avoid unnecessary writes / reads
  - Multi-aggregates, redundancy
  - E.g.: **SystemML codegen**

```
a = sum(X^2)
b = sum(X*Y)
c = sum(Y^2)
```



- **#3 Runtime Piggybacking**
  - Merge concurrent data-parallel jobs
  - "Wait-Merge-Submit-Return"-loop
  - E.g.: **SystemML parfor** [PVLDB'14]

```
parfor( i in 1:numModels )
  while( !converged )
    q = X %*% v; ...
```

Index Structures and NUMA Awareness

- **Goals:** Out-of-core operations and data placement

- **Index Structures**
  - Tree structures of blocks w/ user-defined/fixed linearization functions
  - **LAB-Tree** (Linearized Array B-tree, RIOT) [PVLDB'11]
    - Leaf-splitting strategies, and update batching via flushing policies
  - **TileDB Storage Manager** [PVLDB'16]
    - Two-level blocking and update batching via fragments
  - **AT MATRIX** (Adaptive Tile Matrix, SAP HANA) [ICDE'16]
    - Two-level blocking and NUMA-aware range partitioning

- **NUMA-Aware Model/Data Replication**
  - **DimmWitted:** HW vs statistical efficiency [PVLDB'14]
  - Model: PerCore, PerNode, PerMachine
  - Data: partitioning (sharding), full replication

➔ **Open questions: Heterogenous hardware, cache coherence, etc**

References for Part 5

- K. Kourtis et al. Optimizing Sparse Matrix-Vector Multiplication Using Index and Value Compression. In CF, 2008.
- S. Williams et al.: Roofline: An Insightful Visual Performance Model for Multicore Architectures. Comm. ACM 52(4) 2009.
- Y. Zhang et al. RIOT: I/O-Ecient Numerical Computing without SQL. In CIDR, 2009.
- M. Stonebraker et al. The Architecture of SciDB. In SSDBM, 2011.
- Y. Zhang et al. Storing Matrices on Disk: Theory and Practice Revisited. PVLDB, 4(11), 2011.
- T. Kraska et al. MLbase: A Distributed Machine-learning System. In CIDR, 2013.
- C. Zhang and C. Re. Towards High-throughput Gibbs Sampling at Scale: A Study Across Storage Managers. In SIGMOD, 2013.
- C. Zhang et al. Materialization Optimizations for Feature Selection Workloads. In SIGMOD, 2014.
- M. Boehm et al. Hybrid Parallelization Strategies for Large-Scale Machine Learning in SystemML. PVLDB, 7(7), 2014.
- C. Zhang and C. Re. DimmWitted: A Study of Main-Memory Statistical Analytics. PVLDB, 7(12), 2014.
- E. R. Sparks et al. Automating Model Search for Large Scale Machine Learning. In SoCC, 2015.
- D. Kernert et al. Topology-Aware Optimization of Big Sparse Matrices and Matrix Multiplications on Main-Memory Systems. In ICDE, 2016.
- A. Elgohary et al. Compressed Linear Algebra for Large-Scale Machine Learning. PVLDB, 9(12), 2016.
- M. Boehm et al. SystemML: Declarative Machine Learning on Spark. PVLDB, 9(13), 2016.
- Stavros Papadopoulos et al. The TileDB Array Data Storage Manager. PVLDB 10(4), 2016.
- T. Elgamal et al. SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning. CIDR, 2017.
- F. Li et al. When Lempel-Ziv-Welch Meets Machine Learning: A Case Study of Accelerating Machine Learning using Coding. CoRR, 2017.

Backup: Compressed Linear Algebra (CLA)

[PVLDB 2016]

- **Overview compression framework**
  - Column-wise matrix compression (values + offset lists / references)
  - Column co-coding (column groups encoded as single unit)
  - Heterogeneous column encoding formats (OLE, RLE, DDC, UC)

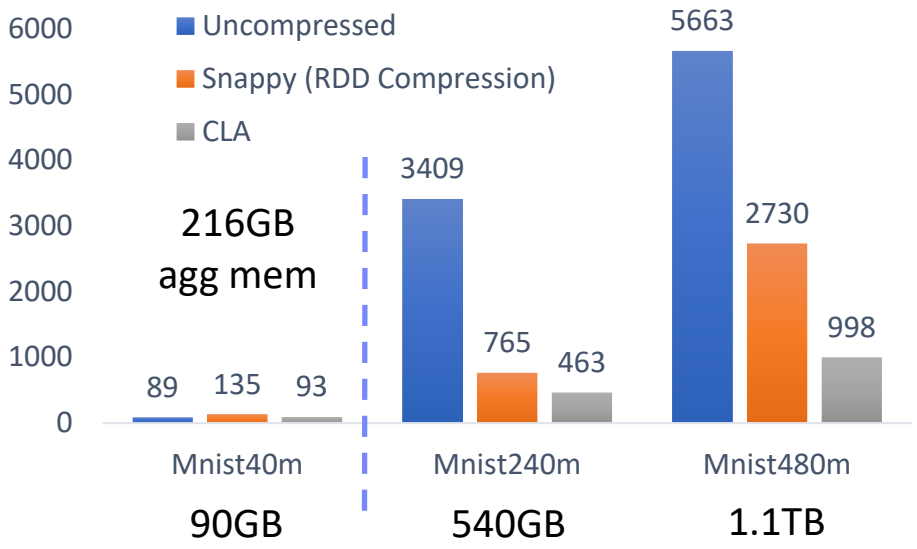
  } **Automatic Planning**

- **Experiments**
  - **LinregCG**, 10 iterations, SystemML 0.14
  - 1+6 node cluster, Spark 2.1

Compression Ratios

Dataset	Gzip	Snappy	CLA
Higgs	1.93	1.38	**2.17**
Census	17.11	6.04	**35.69**
Covtype	10.40	6.13	**18.19**
ImageNet	5.54	3.35	**7.34**
Mnist8m	4.12	2.60	**7.32**
Airline78	7.07	4.28	**7.44**

End-to-End Performance [sec]



Backup: Index Structures

- **Overview Common Indexing Techniques**
  - Physical blocking w/ leaf splitting strategies
  - Dense and sparse leaf blocks w/ contiguous ranges of cells
  - Batching of updates (deferred insertion)

- **LAB-Tree (Linearized Array B-tree, RIOT)** [PVLDB 2011]
  - Operations: get, scan (iterator w/ given order), left/right indexing (on disk)
  - B-tree w/ physical blocking (sparse/dense), leaves have assigned ranges
  - Array linearization via UDFs (e.g., row/column major, Z-order, etc)
  - **Leaf splitting strategies:** split-in-middle, split-aligned, split-off-dense, split-defer-next, split-balanced-ratio
  - **Flushing policies for update batching:** flush-all, least-recently-used, smallest-page, largest-page, largest-page-probabilistically, largest-group

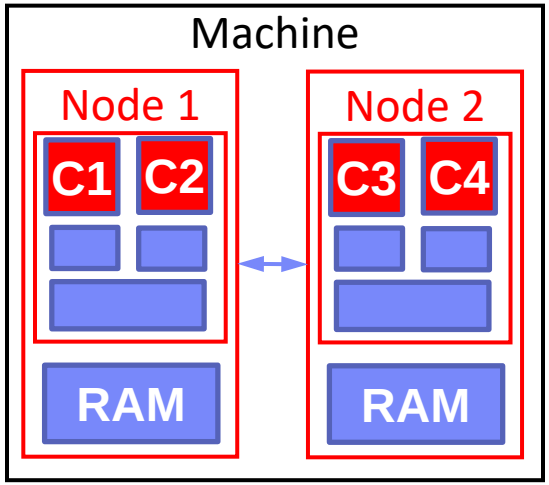
Backup: Index Structures, cont.

- **AT MATRIX (Adaptive Tile Matrix, SAP HANA)** [ICDE 2016]
  - Operations: matrix multiplication ATMult (in-memory)
  - Two-level blocking: **Adaptive variable-sized tiles** (dense or sparse w/ CSR), composed of atomic squared blocks
  - Two-dimensional quad-tree, w/ Z-order as linearization function
  - **Horizontal partitioning across NUMA nodes**

- **TileDB Storage Manager** [PVLDB 2016]
  - Operations: init, write, read, consolidate, finalize (on disk)
  - Two-level blocking: space tiles (fixed size), data tiles (variable size for sparse)
  - Two-level linearization: cell order and tile order (row/column major)
  - **Fragments for update batching**
    ("a timestamped snapshot of a batch of array updates")

Backup: NUMA-Aware Partitioning and Replication

- **AT MATRIX (Adaptive Tile Matrix)**
  - Recursive NUMA-aware partitioning into dense/sparse tiles
  - Inter-tile (worker teams) and intra-tile (threads in team) parallelization
  - Job scheduling framework from SAP HANA (horizontal range partitioning, socket-local queues with task-stealing)



- **NUMA-Aware Model and Data Replication**
  - **DimmWitted: HW vs statistical efficiency**
  - Model Replication
    - PerCore, PerMachine
    - PerNode (hybrid)
  - Data Replication
    - Partitioning (sharding)
    - Full replication

➔ **Open questions: Heterogenous hardware, cache coherence, etc.**

Backup: Buffer Pool Management

- **#1 Intermediates of LA Programs**
  - Hybrid runtime plans of in-memory and distributed operations
  - **Graceful eviction of intermediates at granularity of variables**
  - Example: SystemML
    - Soft references for in-memory matrices and broadcasts
    - LRU, FIFO buffer replacement strategies
- **#2 Operation/Algorithm-Specific Buffer Management**
  - Operations/algorithms over out-of-core matrices and factor graphs
  - **Page-level storage layout and buffer replacement policies**
  - Example #2a: RIOT
    - Chains of matrix multiplications
    - Operation-aware I/O schedules
  - Example #2b: Elementary
    - LR, CRF, LDA over out-of-core factor graphs
    - Materialization strategies and MRU/LFU buffer replacement

Part 6: Resource Elasticity

"Intelligence is the ability to adapt to change."

Stephen Hawking (?)



Jun Yang

Duke University
Durham, NC, USA

SIGMOD 2017

http://picshype.com/rubber-band-ball/amazon.com-:-acco-rubber-band/37225

Rise of Cloud

- **Cluster computing for big data is easier than ever**
  - Clouds allow you to get a cluster on demand, and pay as you go
  - There is a growing ecosystem of platforms and tools for data analysis

Challenges

- **Maddening array of "knobs"**
  - Hardware provisioning, software configuration, program tuning
- **"Elastic" environment**
  - Multi-tenant clusters, fluctuating markets, failures
  - Particularly hard for large-scale, long-running ML workloads

Roadmap

- **Provisioning (& scheduling): what do I need (& when)?**
- **Recovery: what do I do when what I need fails?**
- **Working with markets**

☞**These problems are not limited to DB & ML workloads, but we shall see how DB & ML add twists**

Provisioning: Example Decisions

- **Given an ML program, what types of machines to acquire, and how many**
  Cumulon **[SIGMOD'13+follow-up]**
  - A bigger cluster may get results faster, but cost more
  - No perfect speedup, so big clusters may not give good cost/time trade-off
- **Given a cluster, how to configure the execution of an ML program**
  - What's the appropriate degree of parallelism for an execution step?
    SystemML **[DEBull'14,PVLDB'16]**
    ScalOps **[DeBull'12]**
    - Overhead of parallelism isn't always justified
  - How much memory do we allocate to master and work processes?
    SystemML **[SIGMOD'15]**
    - Optimal allocation depends on computation and data access patterns

☞**Decisions interact with optimizations discussed earlier**

- Cluster configuration affects degree of parallelism and memory allocation, as well as optimal execution strategies

Provisioning/Scheduling: Techniques

Depend on the level of abstraction:

- **Program is a black box**
  - First observe, and then decide; can leverage past execution profiles
- **Program is broken down into a workflow with clear input/output for each unit, e.g., *MapReduce*, *Spark***
  - More effective profiling and optimization on a per-unit basis
- **Program is specified declaratively, DB-style**
  - Reusable and composable cost models
  - Bigger search space through rewrites
  - Cost-based what-if analysis

***SystemML* [ICDE'11+follow-up]**
***Cumulon* [SIGMOD'13+follow-up]**

- **Program follows a specific template**
  - Even more opportunities arise; e.g., scheduling parameter updates/synchronization in ***parameter servers* [VLDB'10,OSDI'14]** + resource provisiong in ***Dolphin* [MLSys'16]** + adapting learning rate by update staleness in ***DynSGD* [SIGMOD'17]**

☞**Adaptation is always key, regardless of abstraction level**

Recovery: General Techniques

Depend on the level of abstraction:

- **Program is a black box**
  - Checkpointing VM state in reliable/redundant storage
- **Program is a workflow with clear input/output for each unit**
  - Write input/output to reliable storage + rerun failed units, e.g., ***Hadoop/MapReduce***
  - Intermediate results can be in memory and lost + recover using lineage ***Spark RDD*** **[NSDI'12]**
- **Program is specified declaratively, DB-style**
  - Finer-grained lineage-based recovery using knowledge of operators + intelligent selective checkpointing ***Cümülön*** **[PVLDB'15]**

Recovery: Algorithm-Specific

- **Many ML algorithms can tolerate missing input or errors by design**
  - Instead of recovering to a state where as if failures never occurred, convert failures into “soft” ones that algorithms can handle themselves
- **Example: distributed batch gradient descent**

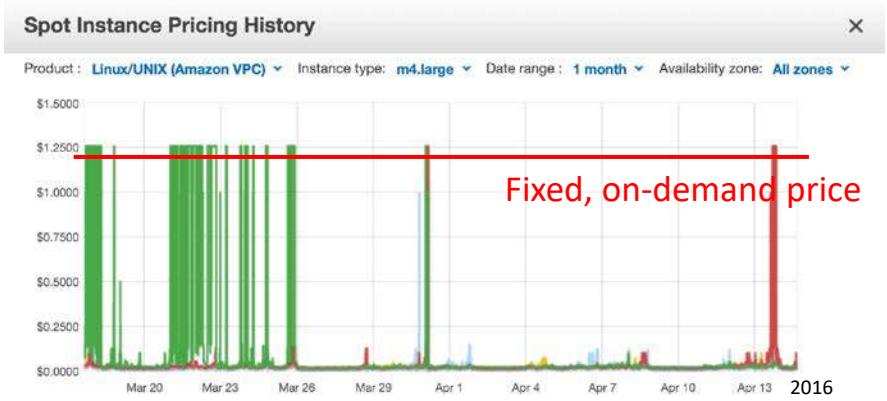
  Narayanamurthy+ (*REEF*) [BigLearn'13]
  - In an iteration, if a task fails to calculate the contribution from one partition of data, simply use an approximation (from the previous iteration)
  - Algorithm still converges

☞**Generalized to user-defined, algorithm-specific “compensations”**

Schelter+ [CIKM'13]

Working with Markets

- **"On-demand" (regular) instances: fixed price, guaranteed**
- **"Spot" instances: availability/price vary over time; e.g; on Amazon:**
  - You set a bid price, and get instances if bid price ≥ market price
  - You pay market price (@hour start), by hours
  - You lose the instances if market price rises above your bid, but your last hour will be free
- **Price can depend on machine type, region, and time**



☞**How do we leverage markets effectively?**

- Pop quiz: would you ever bid higher than the fixed price?
  - Yes! Less chance of losing them, yet still lower cost on average

Working with Markets: Techniques

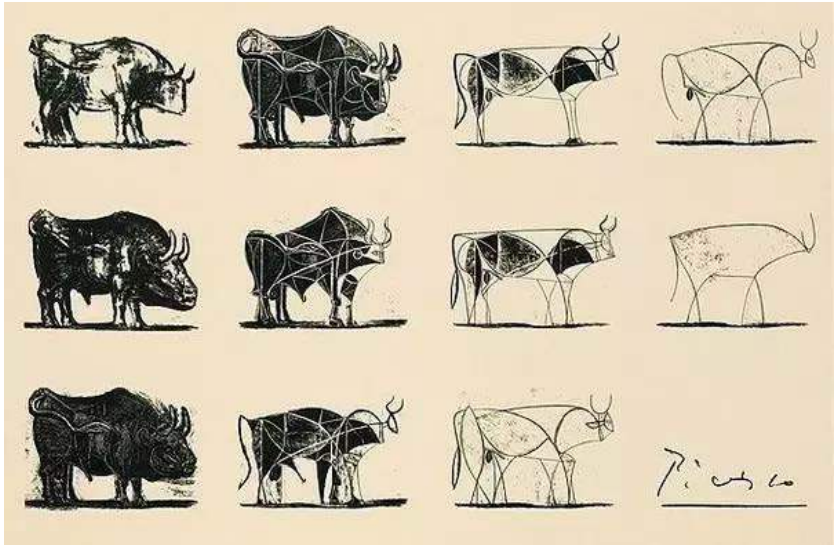
- **Diversify your portfolio: consider instances with different types, across regions**
  - If one market is too expensive, turn to others, e.g., ***Dyna*** **[TCC'16]**
  - A heterogeneous cluster may be best for mixed workloads, e.g., **Zhang+ [PER'15]**
- **Minimizing expected cost is often not enough; need to control risk**
  - Model the market to quantify uncertainty, e.g., ***Cümülön(-D)*** **[PVLDB'15,'17]**
- **Zafer+ [Cloud'12] squeezes entire execution on spots in an hour; retries with a higher bid price if you lose them**
  - Losing spots within an hour incurs no cost with Amazon
- ***Dyna*** **[TCC'16] tries faster spots before falling back to on-demand**
  - But only if doing so improves the execution time distribution
- ***Cümülön*** **[PVLDB'15] picks the optimal mix of spot/on-demand instances**
  - To minimize expected cost while meeting deadline/budget with high probability
  - Recovers and re-optimizes if spots are lost
- ***Cümülön-D*** **[PVLDB'17] adapts proactively dynamically and proactively**
  - Bids/terminates as needed, based on execution progress and market condition
  - Solves the optimization problem as a Markov Decision Process (MDP) and pre-compiles a "cookbook" to apply at run time

Summary

- **Large-scale ML is increasingly being done in a cloud**
- **Challenges of elasticity are not unique to DB & ML**

- **Lots of uncertainty, but adaption & stochastic modeling come to rescue**

- **Different levels of abstraction lead to different opportunities—declarative (DB-style) ML enables smarter, more effective solutions**



https://www.quora.com/What-is-the-difference-between-abstract-art-and-modern-art

References for Part 6: Resource Elasticity

- ***Cumulon* [SIGMOD'13]** Huang et al. “Cumulon: optimizing statistical data analysis in the cloud.” SIGMOD 2013
- ***Cümülön* [PVLDB'15]** Huang et al. “Cümülön: matrix-based data analytics in the cloud with spot instances.” PVLDB 2015
- ***Cümülön-D* [PVLDB'17]** Huang & Yang. “Cümülön-D: data analytics in a dynamic spot market.” PVLDB 2017
- ***Dolphin* [MLSys'16]** Zhou et al. “Dolphin: Runtime Optimization for Distributed Machine Learning.” ML Systems Workshop, 2016
- ***Dyna* [TCC'16]** Zhou et al. “Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds.” TCC 4(1), 2016
- ***DynSGD* [SIGMOD'17]** Jiang et al. “Heterogeneity-aware Distributed Parameter Servers.” SIGMOD 2017
- **Narayanamurthy+ (*REEF*) [BigLearn'13]** Narayanamurthy et al. “Towards Resource-Elastic Machine Learning.” BigLearn 2013
- ***Parameter Server* [VLDB'10]** Smola & Narayanamurthy. “An architecture for parallel topic models.” VLDB 2010
- ***Parameter Server* [ODSI'14]** Li et al. “Scaling Distributed Machine Learning with the Parameter Server.” OSDI 2014
- **Schelter+ [CIKM'13]** Schelter et al. “All Roads Lead to Rome: Optimistic Recovery for Distributed Iterative Data Processing.” CIKM 2013
- ***ScalOps* [DeBull'12]** Borkar et al. “Declarative systems for large-scale machine learning.” IEEE Data Eng. Bulletin, 35(2), 2012
- ***Spark RDD* [NSDI'12]** Zaharia et al. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing .” NSDI 2017
- ***SystemML* [ICDE'11]** Ghoting et al. “SystemML: Declarative machine learning on MapReduce.” ICDE 2011
- ***SystemML* [SIGMOD'15]** Huang et al. “Resource elasticity for large-scale machine learning.” SIGMOD 2015
- ***SystemML* [VLDB'16]** Boehm et al. “SystemML: Declarative machine learning on Spark.” PVLDB 9(13), 2016
- **Zafer+ [Cloud'12]** Zafer et al. “Optimal bids for spot VMs in a cloud for deadline constrained jobs.” Cloud Computing, 2012
- **Zhang+ [PER'15]** Zhang et al. “Exploiting Cloud Heterogeneity to Optimize Performance and Cost of MapReduce Processing.” Performance Evaluation Review, 42(4), 2015

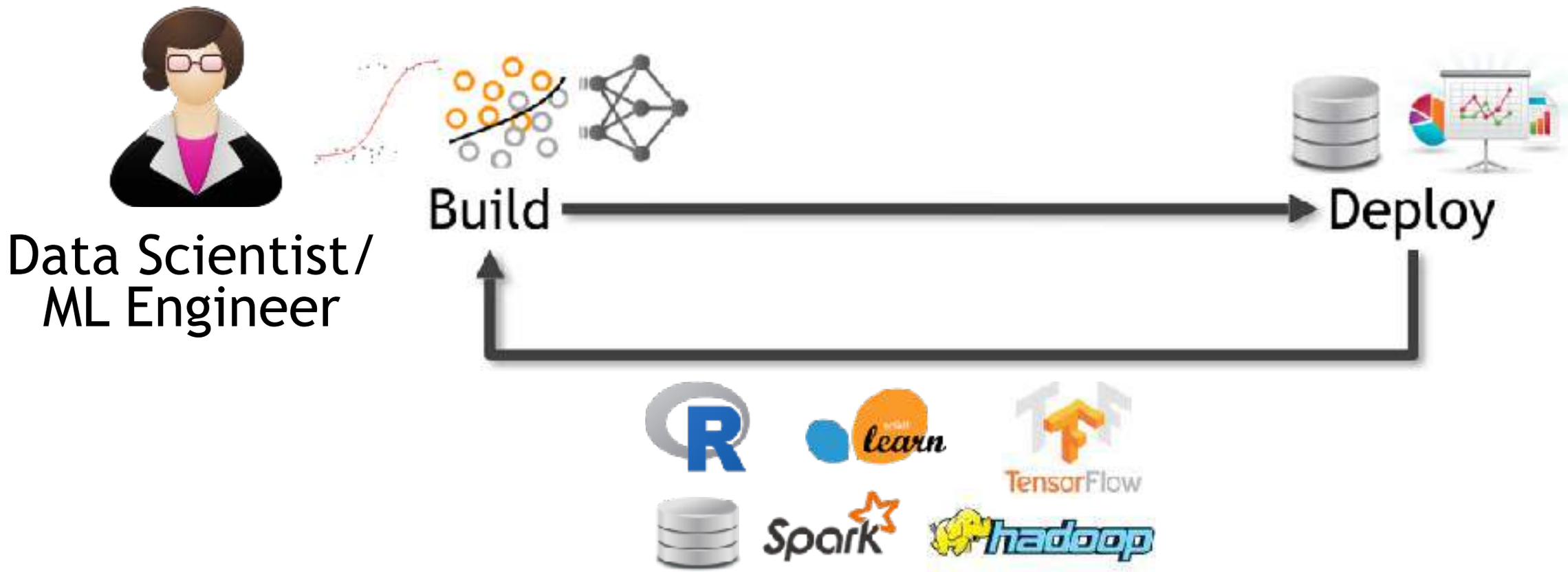
Part 7: ML Lifecycle Systems

Arun Kumar

UC San Diego
La Jolla, CA, USA

SIGMOD 2017

Overview: ML Lifecycle Issues



Data sourcing
Feature engineering
and model selection

Model serving
Tighter loop between
inference and learning

Model management

Feature Engineering

***Q**: What is feature engineering (FE)?*

The process of obtaining a formal representation of the data-generating process as structured signals (features) for an ML model

***Q**: Why is it important from a data management perspective?*

High-quality features are the "secret sauce" of applied ML

FE operations are basically data transformations!

Often "brushed under the carpet" by ML community

***Q**: What sort of operations constitute feature engineering?*

Depends on the data type!

Structured data: Whitening, feature selection/ranking, joins, PCA, etc.

Text: Bag-of-words, Parsing-based, Domain-specific, Word2Vec, etc.

Deep CNNs and RNNs for images, audio, video, time series, etc.

Feature Engineering Systems

Feature selection:

Obtain a subset of features to improve accuracy and/or interpretability

Columbus [SIGMOD'14]:

Often not a single algorithm but a human-in-the-loop dialogue process

Data scientist explores multiple subsets based on domain insights

Understanding customer churn

CustID	Churn?	Age	Income	Gender	City	…
…	…	…	…	…	…	…



Evaluate error with all features in chosen set

Drop demographic features and re-evaluate

Add Gender back in and so on …

A few such common steps encoded as "declarative" ops in DSL

Impl. on top of R/Python; optimizing code-gen middleware

Batching/materialization; QR decomposition; coresets; warm start

Feature Engineering Systems

Treating FE as a dataflow-oriented process; DB-style optimizations:

Brainwash [CIDR'13] / **DeepDive** [DataEng'14]

Workflows of UDFs; feature recommendations

KeystoneML [ICDE'17]

Alternative phy. impl. of solvers; cost-based op. selection

Reducing amount of work for feature coding/evaluation:

Zombie [ICDE'16]

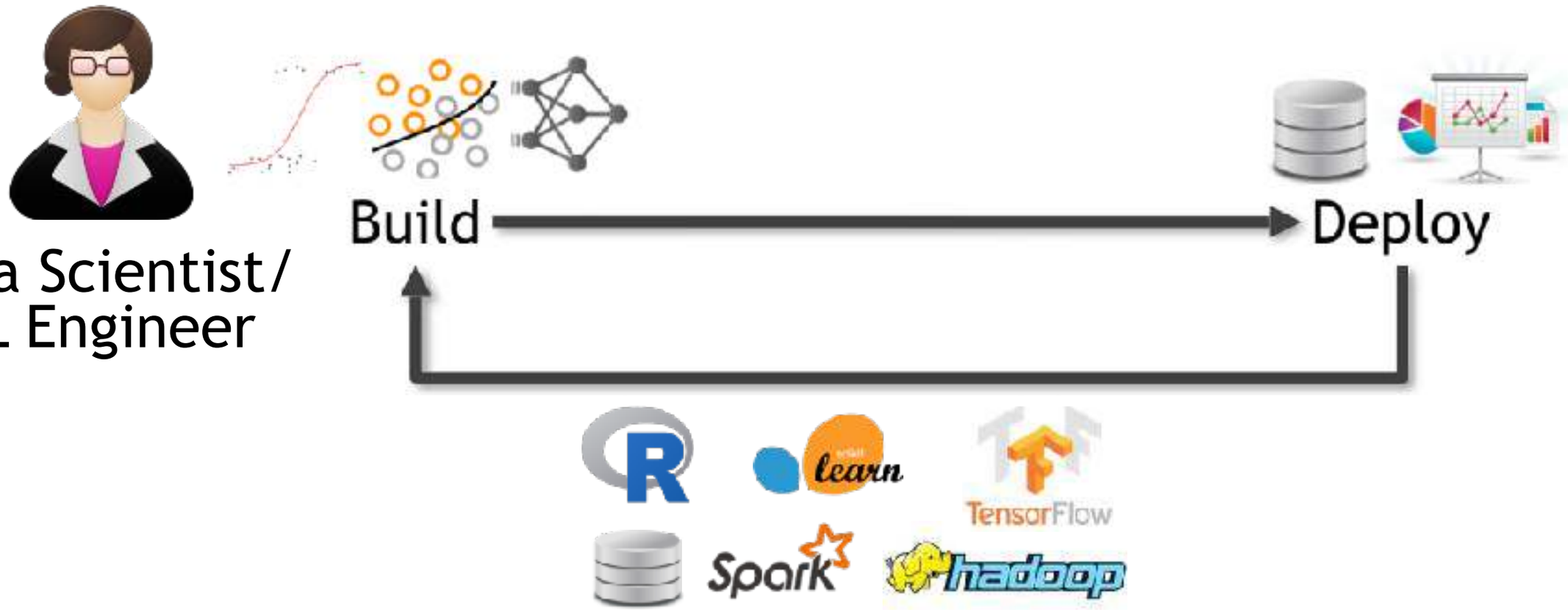
Index structure to sub select relevant data; bandit techniques

Applying learning theory to skip features and help with sourcing tables:

Hamlet [SIGMOD'16]

More open questions remain in *systematizing* feature engineering

Overview: ML Lifecycle Issues



Data sourcing
Feature engineering
and model selection

Model serving
Tighter loop between
inference and learning

Model management

Model Selection

***Q**: What is model selection (MS)?*

The process of obtaining a prediction function to capture a data-generating process using data generated by that process

MSMS [SIGMODRec'15]

Model Selection Triple (MST) (FE, AS, PT)

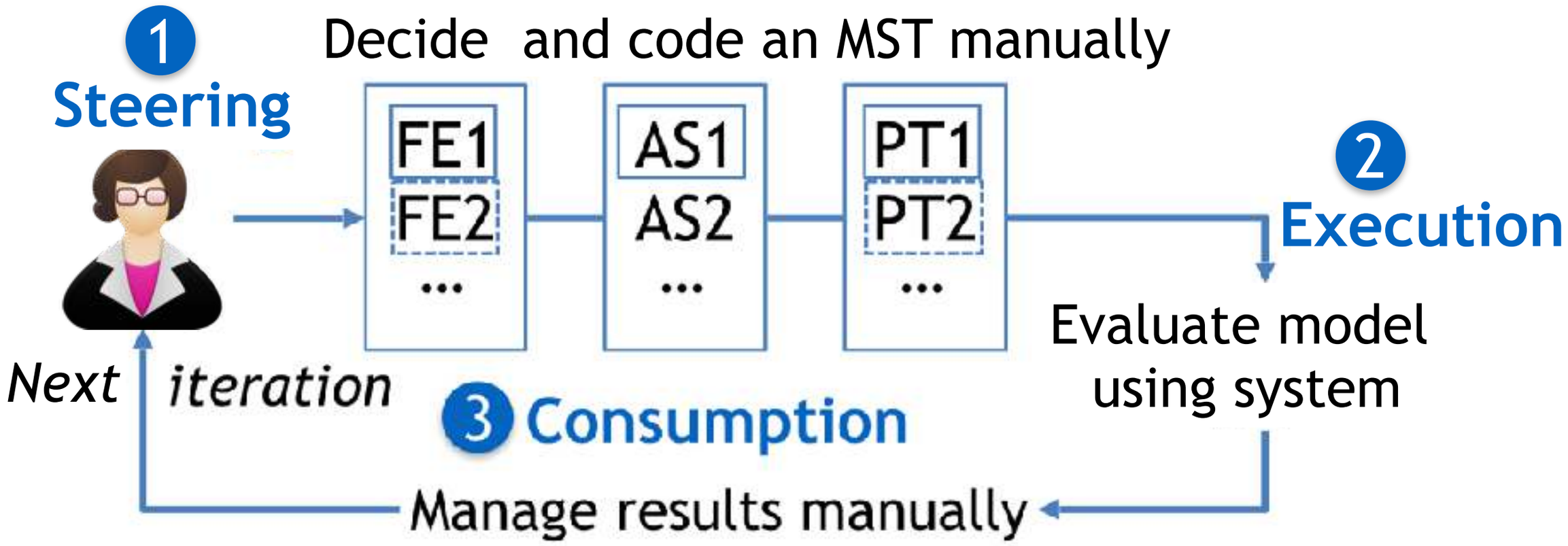
FE: Feature Engineering
AS: Algorithm Selection
PT: (Hyper-)Parameter Tuning

***Q**: Why is it important from a data management perspective?*

FE, AS, and PT often access the dataset (or subsets) repeatedly
A lot of opportunities to improve efficiency with DB-style opt.
FE, AS, and PT are *inter-dependent* and together constitute MS

Model Selection Process

MSMS [SIGMODRec'15] Model Selection Triple (MST): (FE, AS, PT)



Data scientists typically think at a higher level of abstraction

Automation essentially groups MSTs en masse

MS abstractions can help capture intermediate points

Model Selection Process

MSMS [SIGMODRec'15] Model Selection Triple (MST): (FE, AS, PT)

1 "Declarative" interfaces

Group a set of "logically related" MSTs



{FE1, FE2} x AS1 x
{PT1, PT2}
…

Code Generation

2 Optimization

Evaluate models using system

3 Provenance management

Manage results

Next iteration

Many old and recent MS abstractions can be "retro-fitted"

Several new MS abstractions can be introduced to co-exist

Model Selection Management Systems (MSMS)

MSMS [SIGMODRec'15]

<u>The Higher Layers</u>: Declarative Interfaces (some in hindsight!)

Autotuned functions	Columbus	MLBase	*<u>New Abstractions</u>*
E.g., glmnet() in R FE x AS x {PT}	E.g., StepAdd() {FE} x AS x PT	E.g., doClassify() FE x {AS1 x {PT}, AS2 x {PT}}	… {FE} x {AS x PT}, …

{ {FE} x {AS} x {PT} }

<u>The Narrow Waist:</u>
A set of logically related Model Selection Triples (MST)


In-memory


In-RDBMS


Others

<u>The Lower Layers</u>: Optimized Implementations

Model Selection Systems

Automation of AS and PT search with pre-defined search space:

MLbase [CIDR'13] / **TuPAQ** [SoCC'15]

Declarative ML tasks (e.g., "DoClassify"); fixed set of algorithms

Data batching; bandit techniques for explore-exploit search

Hemingway [MLSys'16]

Joint AS and cluster sizing for optimization algorithms

Observe-and-adapt approach for convergence properties

DB-style optimizations for PT and general meta-learning:

SystemML [ICDE'15]; **GLADE** [DanaC'12]

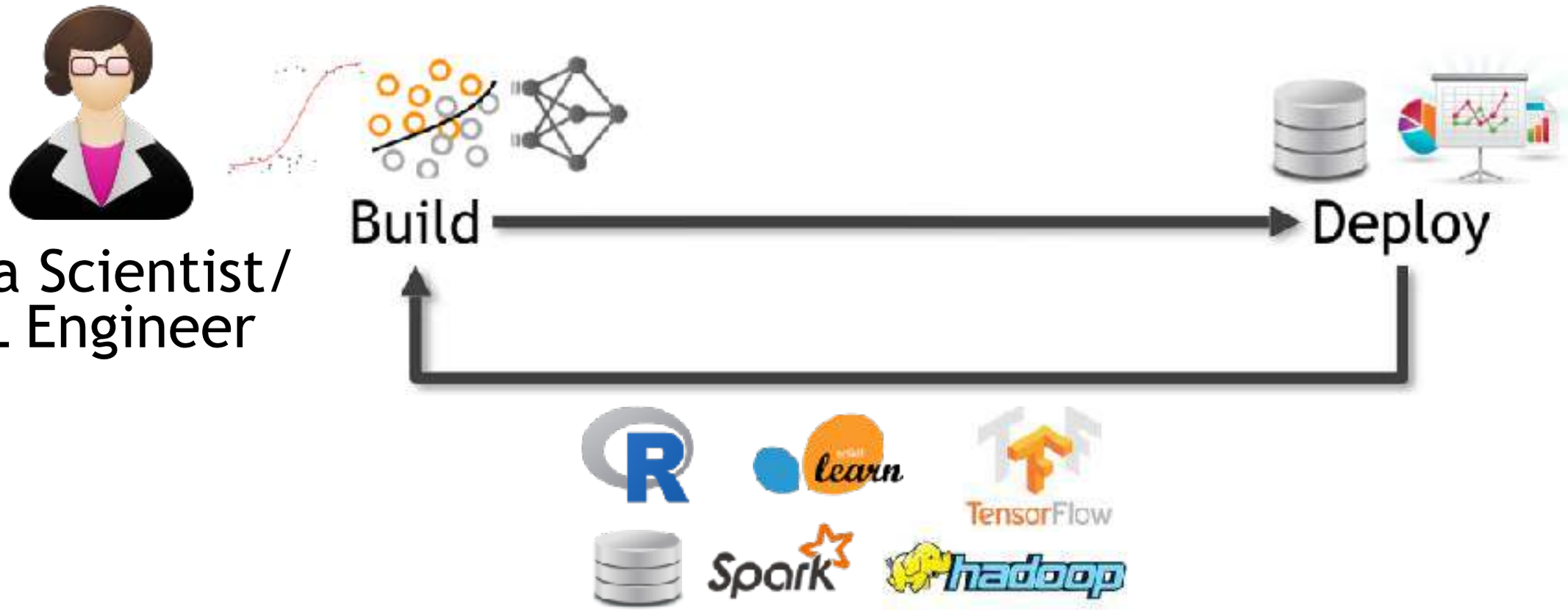
Many open questions remain on optimizing/improving model selection

Interactions of PT with AS and FE

Exploiting redundancy across and within MSTs; cost models

Incorporating constraint/approximations and visualizations, etc.

Overview: ML Lifecycle Issues



Data sourcing
Feature engineering
and model selection

Model serving
Tighter loop between
inference and learning

Model management

Model Management Systems

***Q:** What is model management?*

Treating trained models as data themselves (store, query, debug, etc.)

Integrating ML models with SQL querying: **LongView** [CIDR'11]

Iterative ML debugging: **MindTagger** [VLDB'15], **PALM** [HILDA'17]

Specialized storage engines and custom optimizations:

ModelHub [ICDE'17]

Versioned storage/retrieval of CNNs (sets of float matrices)

Optimizations for reducing storage footprint

Many open questions on managing large space of MSTs, especially for large models (DNNs/trees); ML provenance and debugging

Other ML Lifecycle Issues

Model Serving: High-throughput/low-latency inference/re-learning

MacroBase [SIGMOD'17]

Clipper [NSDI'17] / **Velox** [CIDR'15]

Integrating data-driven applications with reinforcement learning

Data Sourcing: Modeling labeling process; ML+cleaning; ML+pricing

Snorkel [NIPS'16]

ActiveClean [VLDB'16]

Model-Based Pricing [DEEM'17]

Interactive Model Building: Human-in-the-loop interfaces

Ava [CIDR'17]

Vizdom [VLDB'15]

References: Part 7

ActiveClean [VLDB'16]: ActiveClean: Interactive Data Cleaning For Statistical Modeling
Ava [CIDR'17]: Ava: From Data to Insights Through Conversation
Brainwash [CIDR'13]: Brainwash: A Data System for Feature Engineering
Clipper [NSDI'17]: Clipper: A Low-Latency Online Prediction Serving System
Hamlet [SIGMOD'16]: To Join or Not to Join? Thinking Twice about Joins before Feature Selection
Hemingway [MLSys'16]: Hemingway: Modeling Distributed Optimization Algorithms
KeystoneML [ICDE'17]: KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics
Longview [CIDR'11]: The Case for Predictive Database Systems: Opportunities and Challenges
MacroBase [SIGMOD'17]: MacroBase: Prioritizing Attention in Fast Data
MindTagger [VLDB'15]: MindTagger: A Demonstration of Data Labeling in Knowledge Base Construction
MLbase [CIDR'13]: MLbase: A Distributed Machine-learning System
Model-Based Pricing [DEEM'17]: Model-based Pricing: Do Not Pay for More than What You Learn!
ModelDB [HILDA'16]: MODELDB: A System for Machine Learning Model Management
ModelHub [ICDE'17]: ModelHub: Towards Unified Data and Lifecycle Management for Deep Learning
MSMS [SIGMODRec'15]: Model Selection Management Systems: The Next Frontier of Advanced Analytics
PALM [HILDA'17]: PALM: Machine Learning Explanations For Iterative Debugging
Snorkel [NIPS'16]: Data Programming: Creating Large Training Sets, Quickly
SystemML [ICDE'15]: Efficient Sample Generation for Scalable Meta Learning
TuPAQ [SoCC'15]: Automating Model Search for Large Scale Machine Learning
Velox [CIDR'15]: The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox
Vizdom [VLDB'15]: Vizdom: Interactive Analytics through Pen and Touch
Zombie [ICDE'16]: Input Selection for Fast Feature Engineering

Part 8: Open Problems and Conclusions

Arun Kumar
UC San Diego
San Diego, CA, USA

Matthias Boehm
IBM Research – Almaden
San Jose, CA, USA

Jun Yang
Duke University
Durham, NC, USA

SIGMOD 2017

Open Problems: Optimizer and Runtime

- **#1 Size and Sparsity Estimation**
  - Fundamental building block for cost comparisons / valid plan generation
  - Issues: function calls, UDFs, data-dependent operators, changing sizes
- **#2 Convergence Estimation**
  - Number of iterations until convergence unknown
  - Required for cost comparisons and progress estimation
- **#3 Adaptive Query Processing and Storage**
  - Unknown or changing workloads → adaptive query processing
  - Currently limited to inter-DAG recompilation and expression optimization
- **#4 Automatic Rewrites and Operator Fusion**
  - Huge potential for simplification rewrites and operator fusion
  - Challenging in presence of new access methods, compression, etc.
- **#5 Special Value Handling**
  - Special values such as NaN, INF, -0 ignored by most systems → **incorrect results**
  - Support these special values w/o sacrificing performance

Open Problems: End-to-End Lifecycle

- **#6 Integrating Relational and Linear Algebra**
  - Seamless optimizer / runtime integration in holistic framework
  - Including data transformations, training and prediction

- **#7 Seamless Feature Engineering and Model Selection**
  - (Semi-)automating feature engineering and model selection
  - Including abstractions, meta-algorithms, and system architectures

- **#8 ML System Benchmarks**
  - Existing benchmarks limited to ML tasks in terms of reference implementations of large-scale ML libraries or SQL-centric workloads
  - Broader range of benchmarks at various abstraction levels

Conclusions

- **Summary**
  - Compelling arguments for integrating **ML → DB** and **DB → ML**
  - ML in data systems, DB-inspired ML systems, ML lifecycle systems

- **#1 Existing Work to Build Upon**
  - Awareness of existing systems and techniques
  - Survey of effective optimization and runtime techniques

- **#2 Where the Data Management Community Can Help**
  - Integrating ML into existing data systems
  - Optimizer and runtime techniques for large-scale ML systems
  - Tools and systems to simplify/improve the end-to-end ML lifecycle

  ➔ Many open technical problems