MatFast

IN-MEMORY DISTRIBUTED MATRIX COMPUTATION PROCESSING AND OPTIMIZATION BASED ON SPARK SQL

Mingjie Tang
Yanbo Liang

Oct, 2017

HORTONWORKS®
POWERING THE FUTURE OF DATA™

About Authors

- Yongyang Yu
  - Machine learning, Database system, Computation Algebra
  - PhD students at Purdue University
- Mingjie Tang
  - Spark SQL, Spark ML, Database, Machine Learning
  - Software Engineer at Hortonworks
- Yanbo Liang
  - Apache Spark committer, Spark MLlib
  - Staff Software Engineer at Hortonworks
- … All Other Contributors

Agenda

Motivation

Overview of MatFast

Implementation and optimization

Use cases

Motivation

- Many applications rely on efficient processing of queries over big matrix data:
  - Recommender systems
  - Social network analysis
  - Predict traffic data flow
  - Anti-fraud and spam detection
  - Bioinformatics

Motivation

- Recommender Systems

  Netflix's user-movie rating table (sample)

movies users					
Alice	4	?	3	5	4
Bob	?	5	4	?	?
Cindy	3	?	?	?	2

Problem: Predict the missing entries in the table

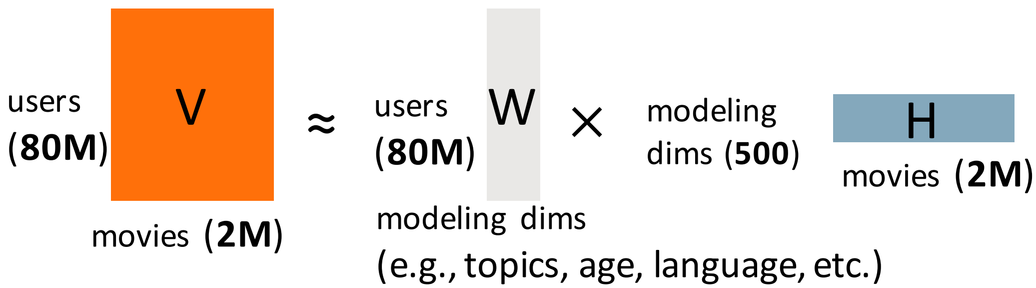
Input: User-movie rating table with missing entries

Output: Complete user-movie rating table with predictions

For Netflix, #users = 80 million, #movies = 2 million

Motivation

- Gaussian Non-negative Matrix Factorization (GNMF)
  - Assumption: $V_{u \times m} \approx W_{u \times t} \times H_{t \times m}$



huge volume

dense/sparse storage

iterative execution

```
Initialize W and H
for i = 1 to nIter do
    H = H * (W^T × V) / (W^T × W × H)
    W = W * (V × H^T) / (W × H × H^T)
end
```

Matrix operation for GNMF Algorithm

Motivation

- User-Movie Rating Prediction with GNMF

```
val p = 200 // number of topics
val V = loadMatrix("in/V") // read matrix
val max_niter = 10 // max number of iteration
W = RandomMatrix(V.nrows, p)
H = RandomMatrix(p, V.ncols)
for (i <- 0 until max_niter) {
    H = H * (W.t %*% V) / (W.t %*% W %*% H)
    W = W * (V %*% H.t) / (W %*% H %*% H.t)
}
(H %*% W).saveToHive()
```


HORTONWORKS

State of the art solution in Spark ecosystem

- Alternative Least Square approach in Spark (ALS)
  - Experiment on Spotify data
  - 50+ million users x 30+ million songs
  - 50 billion ratings For rank 10 with 10 iterations
  - ~1 hour running time

- How to extend ALS to other matrix computation?
  - SVD
  - PCA
  - QR

Observation

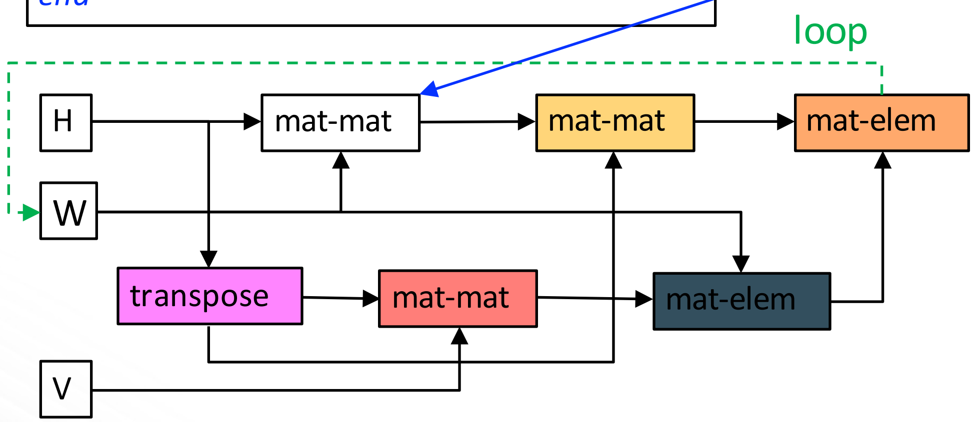
Initialize W and H

for i = 1 *to* nIter *do*

$$H = H * (W^T \times V) \;/\; (W^T \times W \times H)$$
$$W = W * \; V \times H^T \;/\; \big((W \times H) \times H^T\big)$$

end

intermediate result cost:
8 X 10^{16}



Matrix computation evaluation pipeline

Observation

Initialize W and H

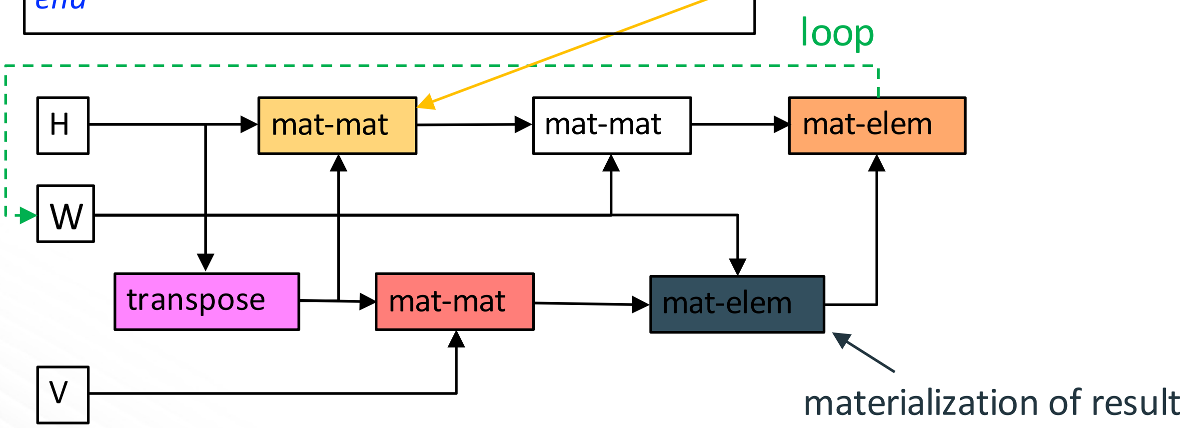
for i = 1 *to* nIter *do*

$$H = H * (W^T \times V) \,/\, (W^T \times W \times H)$$

$$W = W * V \times H^T / (W \times (H \times H^T))$$

end

intermediate result cost:
5 X 10^{11}



Matrix computation evaluation pipeline

HORTONWORKS

Observation

Initialize W and H

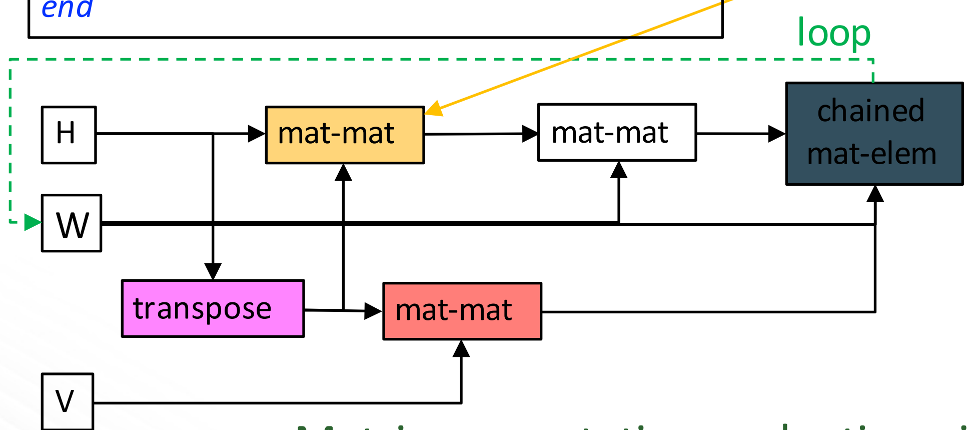
for i = 1 *to* nIter *do*

$$H = H * (W^T \times V) \,/\, (W^T \times W \times H)$$

$$W = W * (V \times H^T) \,/\, (W \times (H \times H^T))$$

end

intermediate result size:
5 X 10^{11}



Matrix computation evaluation pipeline

Overview of MatFast

HORTONWORKS®

Matrix operators

- Unary operator
  - Transpose: $\mathbf{B} = \mathbf{A}^T$
- Binary operators
  - $\mathbf{B} = \mathbf{A} + \beta$; $\mathbf{B} = \mathbf{A} * \beta$;
  - $\mathbf{C} = \mathbf{A} \bigstar \mathbf{B}$, $\bigstar \in \{+, *, /\}$;
  - $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ (**A** %*% **B**)

matrix-matrix multiplication

- Others
  - return a matrix: `abs`(**A**), `pow`(**A**, p)
  - return a vector: `rowSum`(**A**), `colSum`(**A**)
  - return a scalar: `max`(**A**), `min`(**A**)

HORTONWORKS

Optimization targets

- MATFAST generates a computation- and communication-efficient execution plan:
  - Optimize a *single matrix operator* in an expression ✓
  - Optimize *multiple operators* in an expression
  - Exploit *data dependency* between different expressions ✓

HORTONWORKS

Comparison with other systems

	Single	Distributed w. multiple nodes				
	R	ScaLAPACK	SciDB	SystemML	MLlib	DMac
huge volume.		✔	✔	✔	✔	✔
sparse comp.	✔		∼	✔	∼	∼
multiple operators	✔	✔	✔	✔	✔	✔
partition w. dependency						✔
opt. exec. plan				✔		
interface	R script	C/Fortran	SQL-like	R-like	Java/Scala	Scala
fault tolerance			✔	✔	✔	✔
open source	✔	✔	∼	✔	✔	

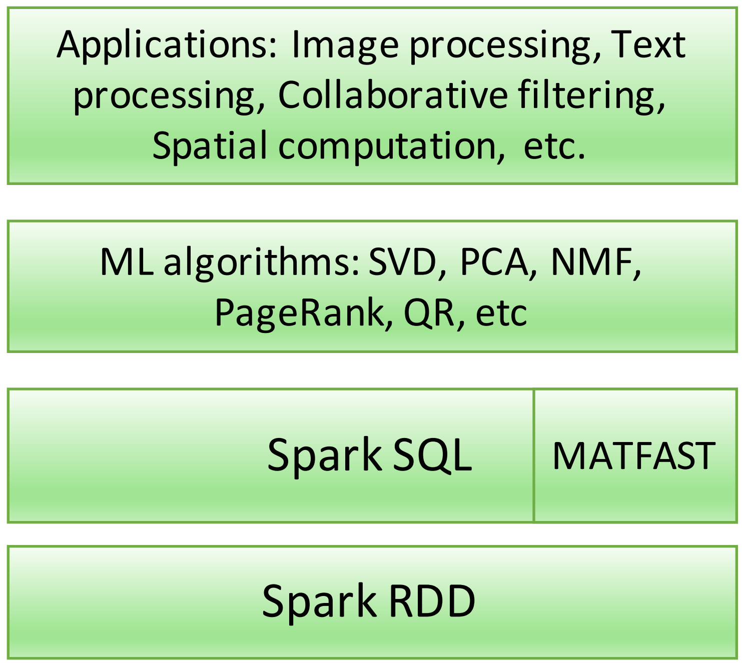
HORTONWORKS

Compare with Spark SQL

	Matrix operators	SQL relational query
Data type	matrix	relational table
Operators	transpose, mat-mat, mat-scalar, mat-elem	join, select, group by, aggregate
Execution scheme	iterative	acyclic

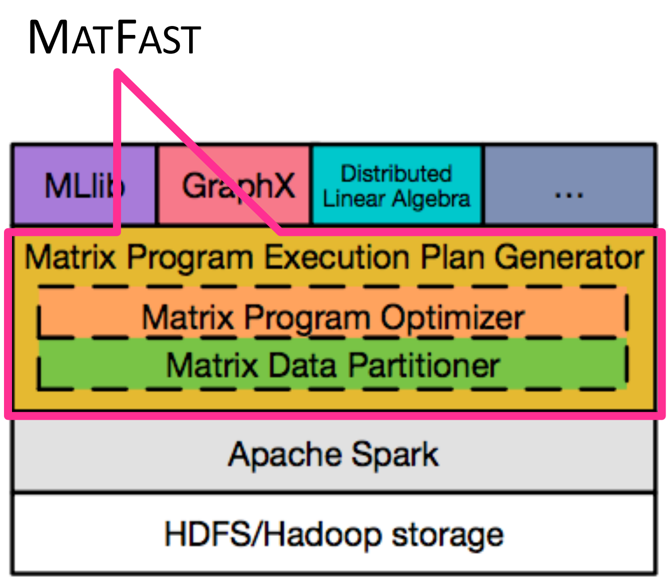
HORTONWORKS

System framework

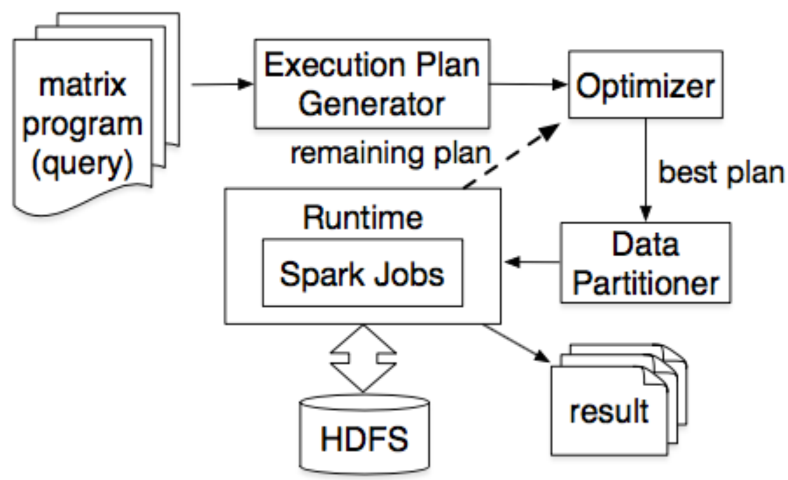


HORTONWORKS®

System framework



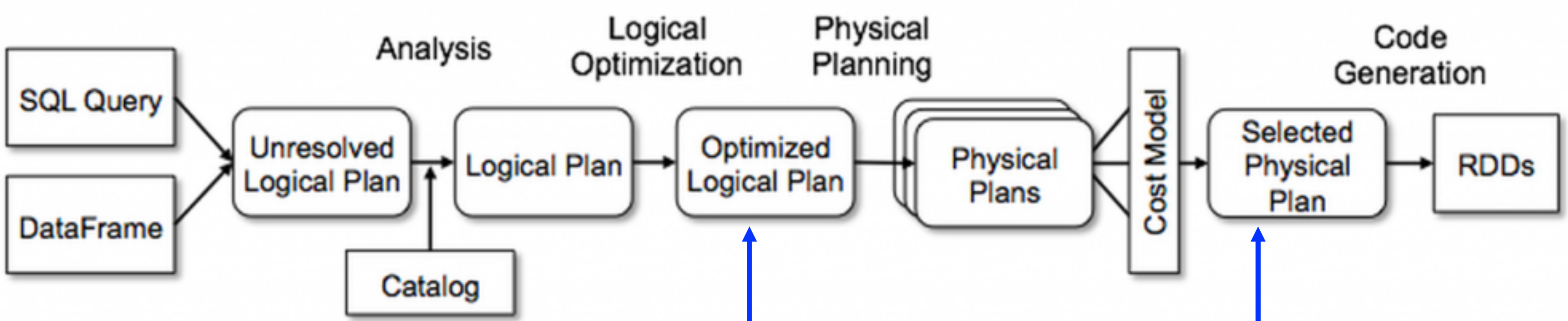
Components



Architecture

MatFast within Spark Catalyst

- Extend Spark Catalyst



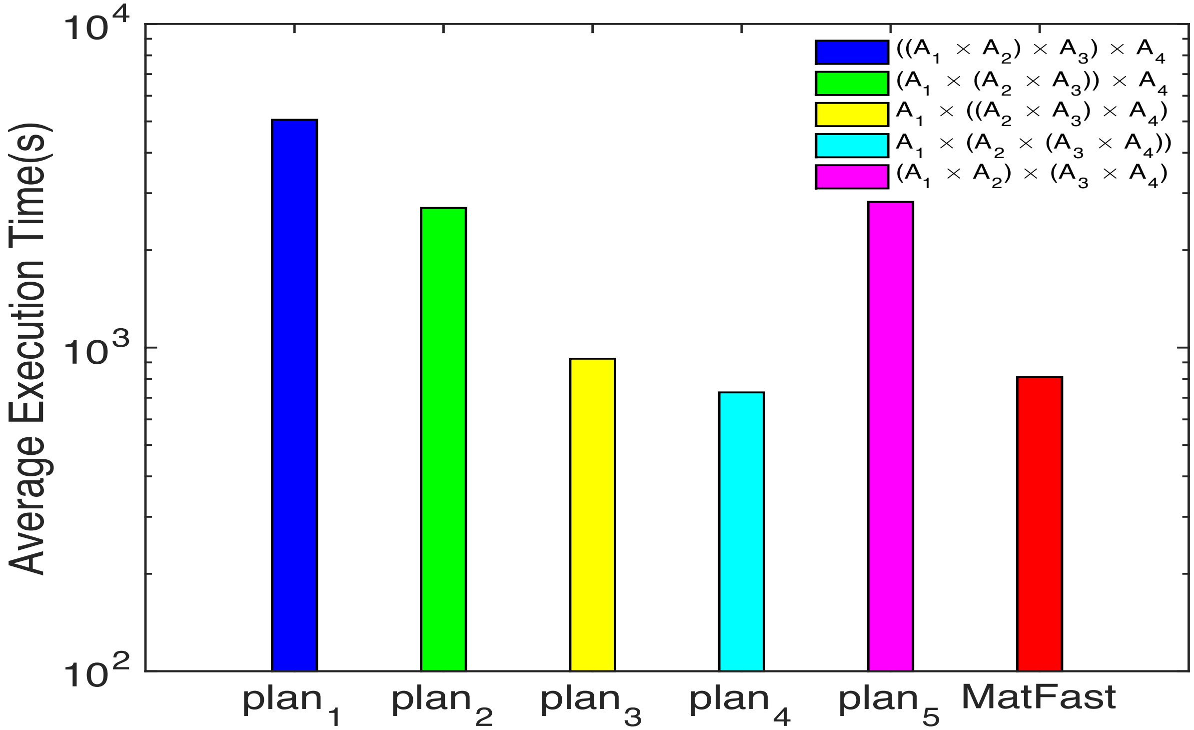
Rule based optimization (single matrix operators, multiple matrix operators)

Cost based optimization (optimizing data partitioning)

Implementation and optimization

HORTONWORKS®

Optimization 1: a Single Operator - Cost Based Optimization



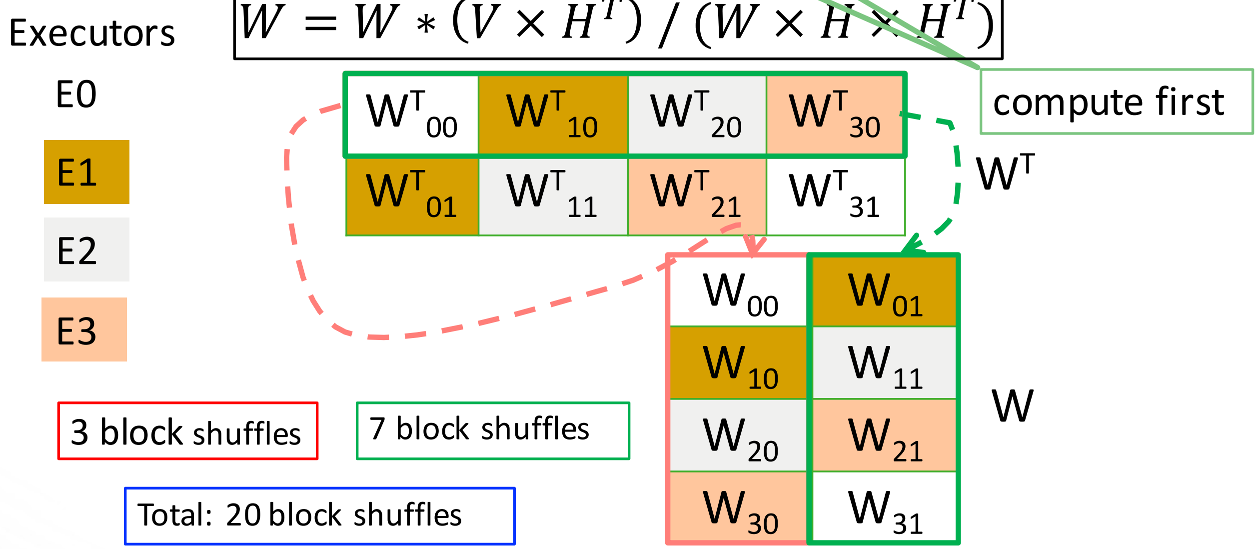
Optimization 2: optimizing data partitioning in pipeline

- Distribute matrix data over a set of workers
- How to determine the data partitioning scheme for a matrix such that minimum shuffle cost is introduced for the entire pipeline?
- Partitioning schemes
  - Row scheme (“*r*”)
  - Column scheme (“*c*”)
  - Block-Cyclic scheme (“*b-c*”)
  - Broadcast scheme (“*b*”)

Optimization 2: optimizing data partitioning in pipeline

- How to determine the data partitioning scheme for a matrix such that minimum shuffle cost is introduced for the entire pipeline?

$$H = H * (W^T \times V) \,/\, (\boxed{W^T \times W} \times H)$$
$$W = W * (V \times H^T) \,/\, (W \times H \times H^T)$$

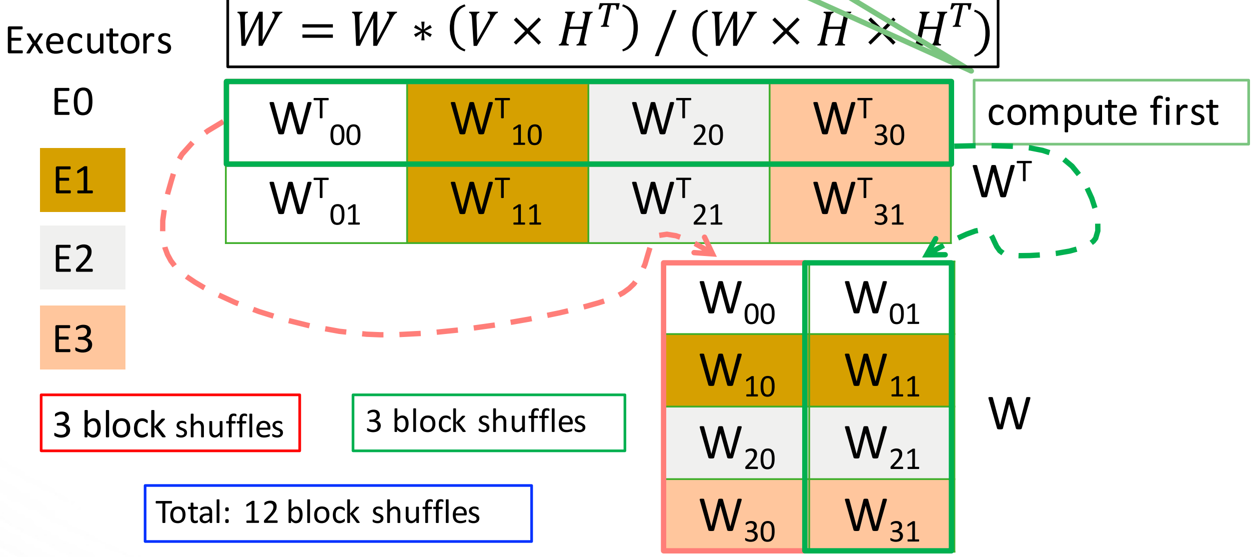


Hash-based partition, $(i + j)\ \%\ N$

Optimization 2: optimizing data partitioning in pipeline

- How to determine the data partitioning scheme for a matrix such that minimum shuffle cost is introduced for the entire pipeline?

$$H = H * (W^T \times V) \,/\, (\boxed{W^T \times W} \times H)$$
$$W = W * (V \times H^T) \,/\, (W \times H \times H^T)$$



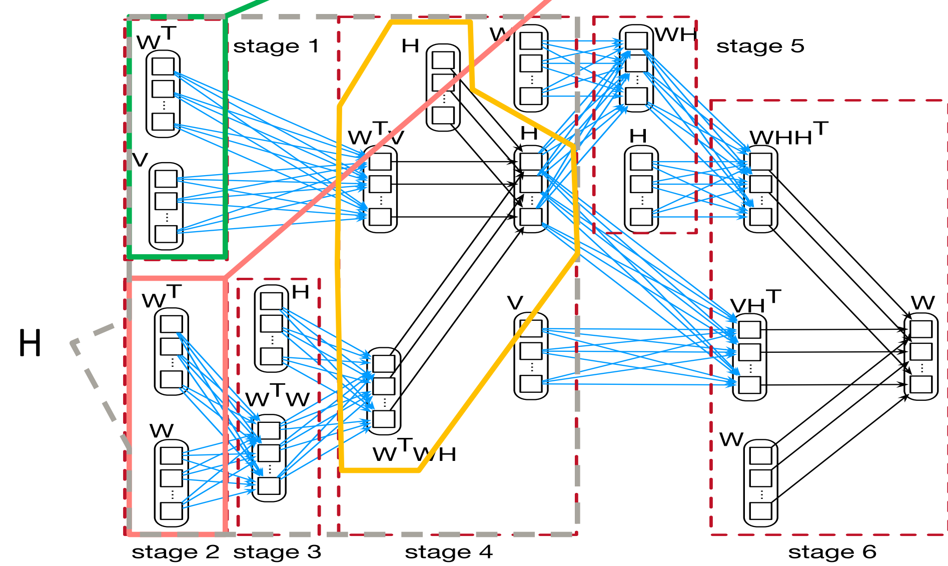
Row-based partition

HORTONWORKS

Optimization 2: optimizing data partitioning in pipeline

- We need an optimized plan to determine an optimized data partitioning scheme for each matrix such that minimum shuffle overhead is introduced for the entire pipeline.
- For example, with hash-based data partitioning, the computation pipeline involves multiple shuffles for aligning the data blocks.

$$H = H * (W^T \times V) \,/\, (W^T \times W \times H)$$
$$W = W * (V \times H^T) \,/\, (W \times H \times H^T)$$

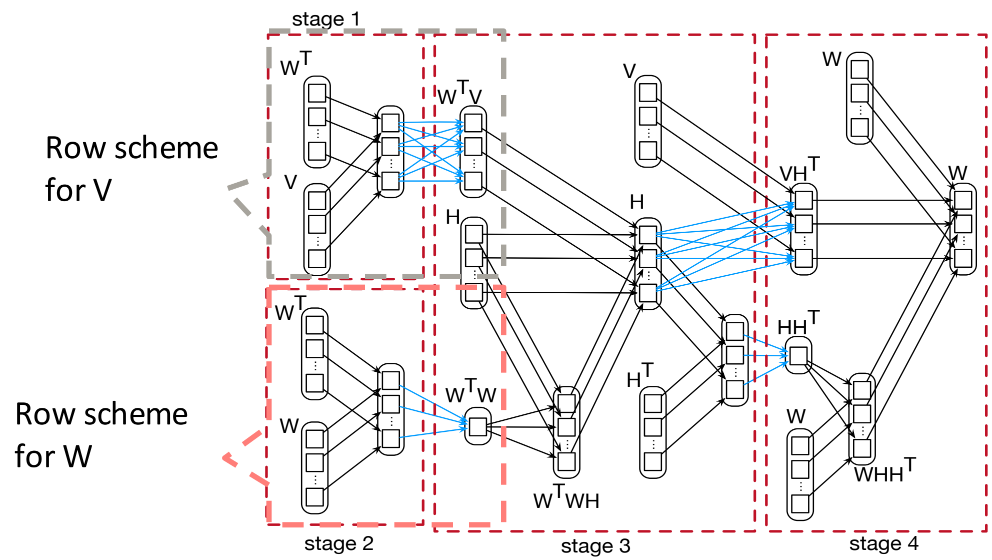


HORTONWORKS

Optimization 2: optimizing data partitioning in pipeline

- MatFast determines the partitioning scheme for an input matrix with min shuffle cost according to the cost model.
- ***Greedily*** optimizes each operator

$$s_{i1(i2)} \leftarrow \underset{s_{i1(i2)}}{\operatorname{argmin}} C_{comm}(op, s_{i1}[, s_{i2}], s_o)$$



- Physical execution plan with optimized data partitioning

Case studies

HORTONWORKS®

Experiments

- Dataset APIs
  - Code examples link

- Compare with state-of-the-art systems
  - Spark MLlib (provided matrix operation)
  - SystemML (Spark)
  - ScaLAPACK
  - SciDB

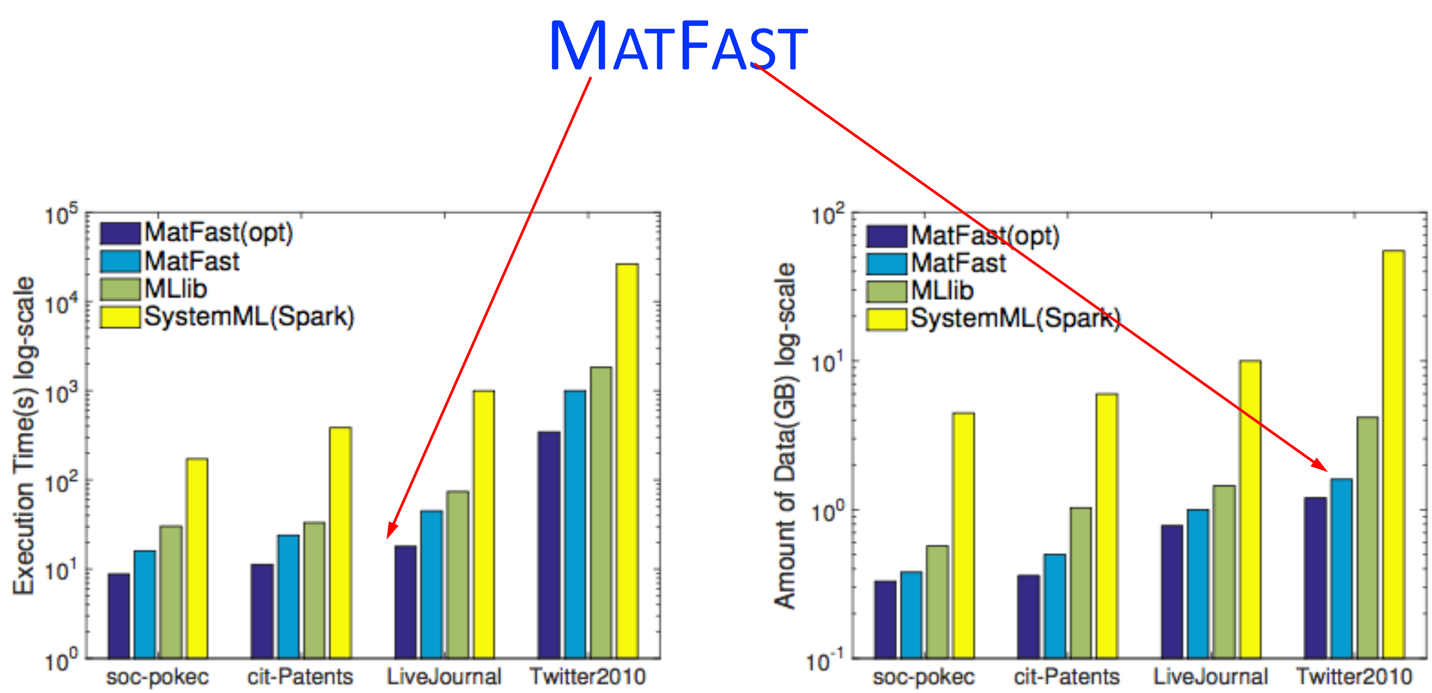
- Netflix data
  - 100,480,507 ratings
  - 17,770 movies from 480,189 customers

- Social network data

Graph	#nodes	#edges
soc-pokec	1,632,803	30,622,564
cit-Patents	3,774,768	16,518,978
LiveJournal	4,847,571	68,993,773
Twitter2010	41,652,230	1,468,365,182

TABLE V: Statistics of the social network datasets

PageRank on different datasets

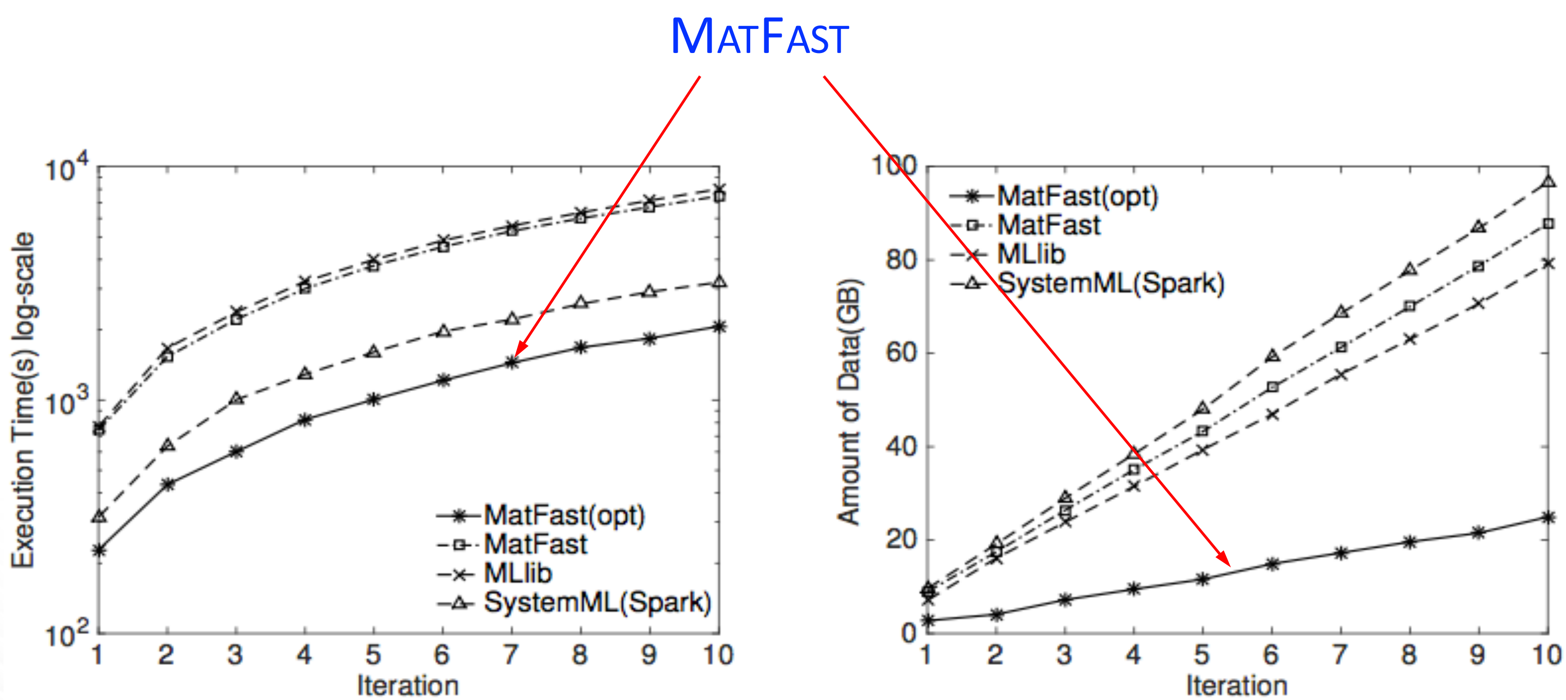


(a) Execution time

(b) Communication cost

HORTONWORKS

GNMF on the Netflix dataset



(a) Execution time

(b) Communication cost

HORTONWORKS®

Future plan

- More user friend APIs
- Advanced plan optimizer
- Python and R interface
- Vertical applications

Conclusion

- Proposed and realized MatFast, an in-memory distributed platform that optimizes query pipelines of matrix operations

- Take advantage of dynamic cost-based analysis and rule-based heuristics to generate a query execution plan

- Communication-efficient data partitioning scheme assignment

HORTONWORKS

Reference

– Yongyang Yu, MingJie Tang, Walid G. Aref, Qutaibah M. Malluhi, Mostafa M. Abbas, Mourad Ouzzani:
In-Memory Distributed Matrix Computation Processing and Optimization. ICDE 2017: 1047-1058

HORTONWORKS

Thanks

Q & A

mtang@hortonworks.com