Optimizing Tensor Computations: From Applications to Compilation and Runtime Techniques

Matthias Boehm  
TU Berlin  
Berlin, Germany

Matteo Interlandi  
Microsoft GSL  
Los Angeles, CA, USA

Chris Jermaine  
Rice University  
Houston, TX, USA

SIGMOD 2023  
Fri Jun 23, 9am – 10.30am
Who We Are

Matthias Boehm
TU Berlin
Berlin, Germany

Matteo Interlandi
Microsoft
Los Angeles, CA, USA

Chris Jermaine
Rice University
Houston, TX, USA

SystemML → SystemDS

SimSQL, MCDB
TRA

DAPHNE
Agenda

• A Case for Tensor Computations [15min]
• Selected Applications [35min]
  • Query Processing and Data Analytics
  • Data Science Lifecycle Tasks
  • Simulation and Sampling
• Selected Runtime Backends [35min]
  • Tensor Query Processor (TQP)
  • Apache SystemML/SystemDS
  • Tensor Relational Algebra
A Case for Tensor Computations
Motivation: A Historic Perspective

• Two Key Drivers of DB Research
  • New analysis workloads (NLP, key/value, RDF/graphs, documents, time series, ML) and applications
  • New HW/infrastructure (multi-/many-core, cloud/serverless, scale-up/out, NUMA/HBM, RDMA, SSD/NVM, FPGA/GPU/ASIC)

• Past: Waves of General-purpose and Specialized Systems
  • Goal #1: Avoid boundary crossing ➔ General-purpose
  • Goal #2: New workload + Performance ➔ Specialized systems
  • Some Examples

New Workloads
System
New HW/Env

ML Systems
RDBMS

OODBMS  XML  Docs  OLAP  MR/Spark  RDF/graphs  NLP  Time
OR  Hybrid  JSON Datatype  HTAP  SQL on Hadoop  In-DB alternatives

In-DB alternatives
Motivation: Tensor Computations

• **Present:** Narrow Focus (on Mini-batch SGD)
  - Increasingly narrow training/inference focus on deep neural networks (DNN) mini-batch stochastic gradient descent (SGD)
  - Parameter servers and similar distribution strategies
  - Communication, security, acceleration primitives for narrow focus

• **Future:** Broader Focus (on General Tensor Computations)
  - General linear algebra programs and tensor computations
  - Different architectures (parameter servers, data- & task-task parallel, hybrid, recursive)
  - Wide variety of applications and workload characteristics
# OLAP Queries / Data Frames / ML Systems

<table>
<thead>
<tr>
<th></th>
<th>DBMS</th>
<th>Data Frames</th>
<th>ML Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language Abstraction</strong></td>
<td>SQL</td>
<td>Relational Algebra ++</td>
<td>Linear Algebra ++</td>
</tr>
<tr>
<td><strong>Workload</strong></td>
<td>Repeated Queries</td>
<td>Explorative Operations</td>
<td>Iterative Algorithms</td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td>Server (but RAWdb, DuckDB)</td>
<td>Stateless library (embedded)</td>
<td>Stateless library (embedded, but Serving)</td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>Join ordering, rewrites Phy. op selection Query compilation</td>
<td>mmchain opt, rewrites Phy. op selection Operation fusion/codegen</td>
<td></td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td>Scans, large-small joins, aggregations; vectorization</td>
<td>Coarse-grained frame operations</td>
<td>Coarse-grained tensor operations vectorization / mini-batches</td>
</tr>
<tr>
<td></td>
<td>Variety of value types</td>
<td></td>
<td>Increasing number of specialized value types</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>Page layouts w/ SMAs and compression</td>
<td>Arrays, open formats (e.g., Arrow, Parquet)</td>
<td>Dense / sparse / compressed blocks</td>
</tr>
</tbody>
</table>

➤ *Increasing Convergence of Optimization and Runtime Techniques*
Long-term Benefits

• **#1 Simplicity**
  - Coarse-grained frame/matrix/tensor data structures and operations
  - Reduced system infrastructure complexity (boundary crossing)

• **#2 Reuse of Compiler/Runtime Techniques**
  - Focused work and reuse of commonly used compiler/runtime techniques
  - Generality over hand-crafted specialized systems and algorithms

• **#3 Performance and Scalability**
  - Leverage HW Accelerators and distributed runtime backends ➔ Increasing specialization and rapid evolution
  - Homogeneous arrays and simple parallelization strategies

Build Libraries for Tensor Ops on HW X once and reuse
Selected Applications

Query Processing and Data Analytics
Data Science Lifecycle Tasks
Simulation and Sampling

[35 min]
Query Processing and Data Analytics

Tensor Computation Runtimes

Neural Networks

CPU  GPU  TPU  Mobile  Browser
Query Processing and Data Analytics

Tensor Computation Runtimes

Neural Networks
Relational (SQL)

CPU
GPU
TPU
Mobile
Browser

TQP
[PVLDB’22]
Query Processing and Data Analytics

Tensor Computation Runtimes

- Neural Networks
- Relational (SQL)
- Classical ML

CPU
GPU
TPU
Mobile
Browser

TQP
PVLDB’22
Hummingbird
OSDI’22
Query Processing and Data Analytics

Questions:
1. How do we represent tabular data as tensors?
2. How can we map SQL operations into tensor programs?
3. How can we map traditional ML model into tensor computations?
Tensor data representation

**Def Tensor:**
A multidimensional matrix that is a cornerstone data structure in AI

<table>
<thead>
<tr>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>saleid</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Numeric as 1-d tensor
Dates as 1-d numeric tensor
Strings as UTF-8 2-d tensor (N x max_len)
**Tensor** data representation

**Def Tensor:**

A multidimensional matrix that is a cornerstone data structure in AI.
Tensor data representation

**Def Tensor:**
A multidimensional matrix that is a cornerstone data structure in AI
Implementing SQL operators using tensor ops

WHERE L_QUANTITY < 24

Opt 1:
1. mask = torch.lt(l_quantity, 24)
2. output = torch.masked_select(l_quantity, mask)

Opt 2:
1. mask = torch.lt(l_quantity, 24)
2. idx = torch.nonzero(mask)
3. output = torch.index_select(l_quantity, dim=0, idx)
Traditional ML models are composed by: **featurizers** and **ML models**

- **Each featurizer** is defined by an **algorithm**
  - e.g., compute the one-hot encoded version of the input feature
- **Each trained model** is defined by a **prediction** function
  - Prediction functions can be either **algebraic** (e.g., linear regression) or **algorithmic** (e.g., decision tree models)
  - Algebraic models are easy to translate: just implement the same formula in tensor algebra!
Translating Trees

$x \in \mathbb{R}^6$

Internal node: $n_*$
Leaf node: $l_*$
Translating Trees

$x \in \mathbb{R}^6$

$n_1
\begin{cases} x_3 > 5.1 \\ T \end{cases}

n_2
\begin{cases} x_1 > 1.8 \\ F \end{cases}

n_3
\begin{cases} x_3 > 2.4 \\ F \end{cases}

n_4
\begin{cases} x_5 > 0.4 \\ T \end{cases}

n_5
\begin{cases} x_5 > 0.4 \\ F \end{cases}

l_1
\begin{cases} +1 \\ 10 \end{cases}

l_2
\begin{cases} +1 \\ 20 \end{cases}

l_3
\begin{cases} +1 \\ 30 \end{cases}

l_4
\begin{cases} +1 \\ 40 \end{cases}

l_5
\begin{cases} +1 \\ 50 \end{cases}

b = \begin{pmatrix} -5.1 \\ -1.8 \\ -2.4 \\ -0.4 \end{pmatrix}

l_v = \begin{pmatrix} 10 \\ 20 \\ \vdots \\ 50 \end{pmatrix}

n_*, l_* \in [0, 1]
Translating Trees

Evaluate all conditions together

Evaluate all conditions together

$\mathbf{x} \in \mathbb{R}^6$

$x_1 > 1.8$

$n_1$

$x_3 > 5.1$

$n_2$

$x_1 > 1.8$

$\mathbf{n} \mathbf{1} = -5.1$

$\mathbf{n} \mathbf{2} = -1.8$

$\mathbf{n} \mathbf{3} = -2.4$

$\mathbf{n} \mathbf{4} = -0.4$

$\mathbf{b} = (10, 20, \cdots, 50)$

$l_1 \in [0, 1]$

$n_*, l_* \in [0, 1]$
Translating Trees

Evaluate all conditions together

Evaluate all paths together
DS Lifecycle: Data Cleaning & Deduplication

• Key Observation
  • State-of-the-art data cleaning based on ML (ML for DI, DI for ML)
  • Examples: Data extraction, schema alignment, entity resolution, data validation, data cleaning, outlier detection, missing value imputation, semantic type detection, data augmentation, feature selection, feature engineering, feature transformations

• Outliers
  • Winsorizing
  • Outlier by standard dev / quantiles

• Missing Value Imputation
  • Imputation by mean / mode [for MCAR]
  • Imputation by mice() [for MAR]

# compute quantiles for lower and upper
ql = quantile(X, 0.05);
qu = quantile(X, 0.95);
# replace values outside [ql,qu] w/ ql and qu
Y = min(qu, max(ql, X));

[Jose Cambronero, John K. Feser, Micah Smith, Samuel Madden: Query Optimization for Dynamic Imputation. PVLDB 2017]

[Xin Luna Dong, Theodoros Rekatsinas: Data Integration and Machine Learning: A Natural Synergy. SIGMOD 2018]
DS Lifecycle: Data Cleaning & Deduplication, cont.

- **Entity Resolution Blocking**
  - Compute word/attribute embeddings + tuple embeddings
  - **Locality-Sensitive Hashing (LSH)** for blocking
  - K hash functions $h(t) \rightarrow k$-dim hash-code
  - L hash tables, each k hash functions

- **Deep Learning for ER**
  - Automatic representation learning from text (avoid feature engineering)
  - Leverage pre-trained word embeddings for semantics (no syntactic limitations)
  - **Examples:** DeepER, Magellan

\[
\begin{align*}
v[t1] &= [0.45, 0.8, 0.85] & \quad v[t2] &= [0.4, 0.85, 0.75] \\
h1 &= [-1, 1, 1] & \quad h2 &= [1, 1, 1] \\
h3 &= [-1, -1, 1] & \quad h4 &= [-1, 1, -1], \\
\end{align*}
\]

\[
\begin{align*}
v \% \% H & \quad v[t1] \% \% H & \quad v[t2] \% \% H \\
\end{align*}
\]

\[
\begin{align*}
[1.2, 2.1, -0.4, -0.5] & \quad [1.2, 2.0, -0.5, -0.3] & \quad [1, 1, -1, -1] & \rightarrow & \quad [1, 1, -1, -1] & \rightarrow & \quad \text{Hash} & \quad \text{[12] bucket}
\end{align*}
\]

[Muhammad Ebraheem et al: Distributed Representations of Tuples for Entity Resolution. *PVLDB 2018*]

[Saravanan Thirumuruganathan et al. Deep Learning for Blocking in Entity Matching [...]. *PVLDB 2021*]

**Distributed Tuple Repr.**

Abs Difference / Hadamard Prod.
DS Lifecycle: Data Cleaning Pipelines

- **Automatic Generation of Cleaning Pipelines**
  - Library of robust, parameterized data cleaning primitives
  - Enumeration of DAGs of primitives & hyper-parameter optimization (GA, HB)

### Other Systems:
- BoostClean,
- HoloClean,
- Raha-Baran,
- Learn2Clean,
- DiffPrep

### Rules/Objectives

#### Data Samples

#### Target App

#### Requirements/Outcomes

### Physical

#### Logical

### Data- and Task-parallel Computation

**P_1**: gmm → imputeFD → mergeDup → delML

**P_n**: outlierBySd → mice → delDup → voting

#### Debugging

#### Outlier Detection → MVI → Deduplication → Resolve Mislabels

<table>
<thead>
<tr>
<th>University</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>TU Graz</td>
<td>Austria</td>
</tr>
<tr>
<td>TU Graz</td>
<td>Austria</td>
</tr>
<tr>
<td>TU Graz</td>
<td>Germany</td>
</tr>
<tr>
<td>IIT</td>
<td>India</td>
</tr>
<tr>
<td>IIT</td>
<td>India</td>
</tr>
<tr>
<td>IIT</td>
<td>Pakistan</td>
</tr>
<tr>
<td>SIBA</td>
<td>Pakistan</td>
</tr>
<tr>
<td>SIBA</td>
<td>null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>University</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>TU Graz</td>
<td>Austria</td>
</tr>
<tr>
<td>TU Graz</td>
<td>Austria</td>
</tr>
<tr>
<td>TU Graz</td>
<td>Austria</td>
</tr>
<tr>
<td>IIT</td>
<td>India</td>
</tr>
<tr>
<td>IIT</td>
<td>India</td>
</tr>
<tr>
<td>IIT</td>
<td>India</td>
</tr>
<tr>
<td>IIT</td>
<td>India</td>
</tr>
<tr>
<td>SIBA</td>
<td>Pakistan</td>
</tr>
<tr>
<td>SIBA</td>
<td>Pakistan</td>
</tr>
<tr>
<td>SIBA</td>
<td>null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.77</td>
<td>0.80</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.96</td>
<td>0.12</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.66</td>
<td>0.09</td>
<td>null</td>
<td>1</td>
</tr>
<tr>
<td>0.23</td>
<td>0.04</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0.91</td>
<td>0.02</td>
<td>17</td>
<td>null</td>
</tr>
<tr>
<td>0.21</td>
<td>0.38</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0.31</td>
<td>null</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0.75</td>
<td>0.21</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>0.19</td>
<td>0.61</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>0.64</td>
<td>0.31</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.77</td>
<td>0.80</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.96</td>
<td>0.12</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.66</td>
<td>0.09</td>
<td>null</td>
<td>1</td>
</tr>
<tr>
<td>0.23</td>
<td>0.04</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0.91</td>
<td>0.02</td>
<td>17</td>
<td>null</td>
</tr>
<tr>
<td>0.21</td>
<td>0.38</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0.31</td>
<td>null</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0.75</td>
<td>0.21</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>0.19</td>
<td>0.61</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>0.64</td>
<td>0.31</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>
DS Lifecycle: Data Augmentation

• Data Augmentation Overview
  • Complex ML models / deep NNs need lots of labeled data to avoid overfitting ➔ expensive
  • Augment training data by synthetic labeled data
  • #1: Movement/selection (translation, rotation, reflection, cropping)
  • #2: Distortions (stretching, shearing, lens distortions, color, mixup of images)
  ➔ Clean mapping to linear algebra operations

• Effects of Data Augmentation
  • #1 Regularization for reduced generalization error, not always training error
  • #2 Invariance increase by averaging features of augmented data points
  ➔ Data Augmentation as a Kernel
  • Kernel metric for augmentation selection
  • Affine transforms on approx. kernel features

AlexNet


DS Lifecycle: Graph Processing

- **Connected Components**
  - Compute connected components (subgraphs)
  - Vertex-centric processing
  - Propagate \( \max(\text{current}, \text{msgs}) \)
    if \( \neq \) current to neighbors, terminate if no msgs

- **Connected Components in Linear Algebra**

- **Other Examples**
  - Page Rank, Shortest Paths

```r
# initialize state with vertex ids
c = seq(1,nrow(G));
diff = Inf; iter = 1;

# iterative computation of connected components
while( diff > 0 & (maxi==0 | iter<=maxi) ) {
  u = \max(\text{rowMaxs}(G * t(c)), c);
diff = \sum(u \neq c);
c = u; # update assignment
  iter = iter + 1;
}
```

[[SIGMOD'10, SIGMOD'20 Test of Time]]
DS Lifecycle: Training Decision Trees

- **Input**: \( X \) (recoded/binned)
- **Main Algorithm**
  
  \[
  X_2 = \text{encode}(X); \\
  \text{while} (\text{length}(\text{queue}) > 0) \{ \\
  \text{node}0 = \text{remove}(\text{queue}, 1); \\
  \text{[...]} = \text{findBestSplit}(X_2, \ldots) \\
  \text{if}(\text{validSplit}) \\
  M[, 2*nID-1:2*nID] <- (f, v); \\
  \text{else} \\
  M[,2*nID] = \text{labelLeaf}(\ldots); \\
  \text{putInQueueCond}(\text{ILeft}); \\
  \text{putInQueueCond}(\text{IRight}); \}
  \]
- **Output**: linearized tree

\[
\text{Task-parallel over features} \\
\text{Optional value sampling } O(\log m) \\
\text{Vectorized } \leq \text{split evaluation (matmult)} \\
\text{Vectorized information gain computation}
\]
DS Lifecycle: Model Debugging

- **Problem Formulation**
  - Intuitive slice scoring function
  - Exact top-k slice finding
  - $|S| \geq \sigma \land sc(S) > 0$
  - $\alpha \in (0,1]$

- **Properties & Pruning**
  - Monotonicity of slice sizes, errors
  - Upper bound sizes/errors/scores → pruning & termination

- **Linear-Algebra-based Slice Finding**
  - Recoded matrix $X$, error vector $e$
  - Vectorized implementation in linear algebra (join & eval via sparse-sparse matrix multiply per lattice level)
  - Local and distributed task/data-parallel execution

$$sc = \alpha \left( \frac{\bar{e}(S)}{\bar{e}(X)} - 1 \right) - (1 - \alpha) \left( \frac{|X|}{|S|} - 1 \right)$$

- Slice error
- Slice size

| Credit: sliceline, Silicon Valley, HBO |
--- | --- |
| DS Lifecycle: Model Debugging | [Sliceline @ SIGMOD’21c] |
Fairness and Explainability

• Fairness Problem Formulation
  • A **fairness specification** given by a triplet \((g, f, \varepsilon)\) induces \((|g(D)| \text{choose } 2)\) fairness constraints on pairs of groups
  • A fairness specification is satisfied by a classifier \(h\) on \(D\) iff all induced fairness constraints are satisfied, i.e., \(\forall g_i, g_j \in g(D), |f(h, g_i) - f(h, g_j)| \leq \varepsilon\)

• Unconstrained optimization problem

• Explainability via LIME
  • **Sample perturbations** of prediction input (e.g., hide parts of image, attribute values)
  • **Locally weighted regression**


Sampling and Simulations

- Classical data analysis assumes database data are correct
  - Ex: I have a customer $c$ who places an order $x_c$
  - $x_c[i]$ represents the dollars spent on the $i$th inventory item purchased
  - We would typically assume that the $x_c[i]$ value recorded in a database reflects reality
  - But does it always? No!
  - Can we afford to ignore the possibility $x_c[i]$ is incorrect?

  Even if correct on average, can’t afford to ignore the implications of the extremes...
To Quantify Risk: Use a Statistical Model

To generate a purchase vector:

1) Sample \( \mathbf{x} \sim \text{Normal}(\mu, \Sigma) \)
2) For \( i \in \{1...4\} \)
3) With probability 0.2
4) \( x[i] \leftarrow 0 \)

\[
\mu = \begin{bmatrix} 15.6 \\ 5.1 \\ 3.2 \\ 6.9 \end{bmatrix}
\]

\[
\Sigma = \begin{bmatrix} 10.2 & 2.5 & 3.8 & 7.1 \\ 2.5 & 3.8 & 7.1 & 4.5 \\ 3.8 & 7.1 & 4.5 & 2.2 \\ 7.1 & 4.5 & 2.2 & 1.6 \end{bmatrix}
\]

\[
\mathbf{x} = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 0 \end{bmatrix}
\]

Strong correlation between Item 1 and item 4

Vector of std devs

Correlation matrix

Simulates the case where the 8.2 was never recorded
Given Such a Model

- We can look at an observed purchase vector and “correct” it
  - The first entry is high (22.8 vs mean of 15.6)
  - But the last entry is low (0 vs mean of 6.9)
  - How is this possible with a correlation of 0.95?
  - According to our model: most likely the 0 is wrong
  - But we don’t know the correct vector
  - Go Bayesian! Use Bayes’ rule to sample from $p(x|\bar{x}, \mu, \Sigma)$

$$\bar{x} = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 0 \end{bmatrix}$$

$$x = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 6.9 \end{bmatrix}$$

$$x = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 7.5 \end{bmatrix}$$

$$x = \begin{bmatrix} 0 \\ 1.6 \\ 4.5 \\ 0 \end{bmatrix}$$
How to Facilitate in a Database?

• Need the ability to move from relations to tensors
  • Math uses tensors (vectors, matrices, etc.), not relations!
• And the ability to sample random values from distributions

Assume sales data are stored relationally:

```sql
CREATE VIEW vecs AS
SELECT VECTORIZE (label_scalar (prod_id, amt))
AS sale_vec, cust_id, sale_date
FROM sale
GROUP BY cust_id, sale_date
```

Agg function to create a vector from (int, numerical) pairs of type (the dim)

Linear Algebra on an RDBMS [ICDE17]
Now We Have a Set of Vectors

- **Compute observed mean and covariance on a per-customer basis**
  - Will serve as an empirical prior for each customer
  - So-called “empirical Bayes”

We now have a view containing sales vectors

vecs (sale_vec, cust_id, sale_date)

And now one that has the empirical covariance matrix:
It’s easy to create a table having the mean vector for each customer:
CREATE VIEW covers AS
SELECT (v.sale_vec - m.avg_sale, v.sale_vec - m.avg_sale) AS covar,
v.cust_id
FROM vecs AS v, means AS m
WHERE v.cust_id = m.cust_id
GROUP BY v.cust_id
We Are Ready To Create a Simulated Table

• **Invoke a special UDF called a “VG function”**
  • In our case, `ResampleVec` encapsulates an MCMC algorithm to “fix” sales vector

Here are the views we’ve created:

vecs (sale_vec, cust_id, sale_date)
means (avg_sale, cust_id)
covars (covar, cust_id)

It’s easy to create a table having the mean vector for each cust:

```
CREATE TABLE simulated_sales AS
FOR EACH v IN vecs
WITH sale_vec AS ResampleVec (v.sale_vec,
(SELECT m.avg_sale FROM means AS m WHERE m.cust_id = v.cust_id),
(SELECT c.covar FROM covars AS c WHERE c.cust_id = v.cust_id))
SELECT v.cust_id, v.sale_date, s.value
FROM sale_vec s
```

Now, queries over this simulated table return a **distribution** of results. Ex:

```
SELECT AVG (s.value)
FROM simulated_sales s,
cust c
WHERE s.cust_id = c.cust_id AND c.region = 'northeast' AND s.sale_date = '12-23-22'
```

VG function accepts old vec
Subquery to get the mean
For each sale, use VG function to sample new vec
`ResampleVec` is a VG function that simulates the sale
Subquery to get the covariance
Then stitch together output tuples
Stochastic Simulations

- Can use these tools to build database-valued Markov chains
  - Facilitates very large scale Bayesian machine learning
  - Can illustrate with a toy example

This table has a single tuple:

people (start_loc)

This table has info on cities:

city (city_id, transition_probs)

**Goal**: implement a simple Markov chain that has people moving from city to city according to the specified probabilities

Initialize the simulation:

```
CREATE TABLE locations[0] (cur_loc) AS
  SELECT * FROM people
```

Simulate the transitions out of each city:

```
CREATE TABLE transitions[i] (next_pos) AS
  FOR EACH c in city
    WITH
      next_pos AS Multinomial (  
        (SELECT l.cur_loc[c.city_id]  
         FROM locations[i-1] AS l),  
        c.transition_probs)  
  SELECT n.value  
  FROM next_pos s
```

And aggregate over all of the source cities to find the number of people in each destination:

```
CREATE TABLE locations[i] AS
  SELECT SUM (next_pos)  
  FROM transitions[i]
```

Note how use of vectors makes this quite simple!
References


[SIGMOD'23] Peng Li, Zhiyi Chen, Xu Chu, Kexin Rong: DiffPrep: Differentiable Data Preprocessing Pipeline Search for Learning over Tabular Data, SIGMOD 2023


[SIGMOD'18] Xin Luna Dong, Theodoros Rekatsinas: Data Integration and Machine Learning: A Natural Synergy. SIGMOD 2018


[SIGMOD'10] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski: Pregel: a system for large-scale graph processing. SIGMOD 2010
Selected Runtime Backends

Tensor Query Processor (RDBMS → MLSys)
Apache SystemDS (MLSys)
Tensor Relational Algebra (MLSys → RDBMS)

[35 min]
SQL to Tensor conversion: Tensor Query Processor (TQP)

SQL Query

Parser and Optimizer

Spark SQL Query Plan

Converter

Primitive-based IR (PIR)

Tensorizer

Tensor program

Executor

Data Ingestion

CPU  AMD GPU  NVIDIA GPU

Hardware

convert time in the 10s of milliseconds (w/o Spark parsing and optimization time)

SELECT MAX(p_supplycost) AS price, s_name AS supp FROM supplier JOIN partsupp ON ps_suppkey=s_suppkey GROUP BY supplier.s_name ORDER BY price DESC;

Tensor program for Sort
Tensor program for Join
Tensor program for Filter

TQP [PVLDB’22]

Spark SQL
Query Plan

Sort Operator

Sort PIR Operator

TQP

Parser and Optimizer

Converter

Primitive-based IR (PIR)

Tensorizer

Executor

Data Ingestion

CPU  AMD GPU  NVIDIA GPU

Hardware
Pros and Con of Tensor Query Processor

**Pro:**
- Scalable Approach (no need to reimplement the query processor for each new hardware)
- Leverage the massive investments in special HW
- Low engineering effort: TQP + HB in 20k lines of Python code
- Great performance

**Limitations:**
- Can only target devices supported by PyTorch (e.g., no Xbox)
- Sub-operators require many data materializations
- Cannot take advantage of advanced neural network compilers techniques (e.g., no operator fusion, no kernel tuning)
- Some operators are hard to implement efficiently
SQL to Tensor conversion: Tensor Query Processor (TQP)
Extending TQP

Antares is a Cross-compiling Engine for Microsoft 1st/2nd party devices

https://github.com/microsoft/antares
Interesting HW configuration: APU design where CPU and GPU share HBM (no PCI-e)

Predictable usage pattern: gamers mostly play during the evenings (AKA dark time)
TPCH on Xbox (SF 10, P100)

GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

<table>
<thead>
<tr>
<th></th>
<th>Xeon E5-2690</th>
<th>P100</th>
<th>Xbox Series X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth (GB/s)</td>
<td>154</td>
<td>732</td>
<td>560</td>
</tr>
<tr>
<td>Unidirectional PCIE3 (GB/s)</td>
<td>-</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>Theoretical TFLOps</td>
<td>1.4</td>
<td>9.5</td>
<td>12.0</td>
</tr>
</tbody>
</table>
TPCH on Xbox (SF 10, P100)

GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

<table>
<thead>
<tr>
<th></th>
<th>Xeon E5-2690</th>
<th>P100</th>
<th>Xbox Series X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth (GB/s)</td>
<td>154</td>
<td>732</td>
<td>560</td>
</tr>
<tr>
<td>Unidirectional PCIe (GB/s)</td>
<td>-</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>Theoretical TFLOps</td>
<td>1.4</td>
<td>9.5</td>
<td>12.0</td>
</tr>
</tbody>
</table>
TPCH on Xbox (SF 10, P100)

GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

---

<table>
<thead>
<tr>
<th></th>
<th>Xeon E5-2690</th>
<th>P100</th>
<th>Xbox Series X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth (GB/s)</td>
<td>154</td>
<td>732</td>
<td>560</td>
</tr>
<tr>
<td>Unidirectional PCIe (GB/s)</td>
<td>-</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>Theoretical TFLOps</td>
<td>1.4</td>
<td>9.5</td>
<td>12.0</td>
</tr>
</tbody>
</table>
TPCH on Xbox (SF 10, P100)

GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

<table>
<thead>
<tr>
<th></th>
<th>Xeon E5-2690</th>
<th>P100</th>
<th>Xbox Series X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth (GB/s)</td>
<td>154</td>
<td>732</td>
<td>560</td>
</tr>
<tr>
<td>Unidirectional PClv3 (GB/s)</td>
<td>-</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>Theoretical TFLOps</td>
<td>1.4</td>
<td>9.5</td>
<td>12.0</td>
</tr>
</tbody>
</table>
AI-Centric Database System

Broader implications of having a DBMS on an ML runtime backend

Integration with AI ecosystem
AI-Centric Database System

Broader implications of having a DBMS on an ML runtime backend

Multi-modal data support

```
SELECT
    input AS images,
    image_text_similarity_model("KFC Receipt", input) AS score
FROM attachments
ORDER BY score DESC
LIMIT 1
```
AI-Centric Database System

Broader implications of having a DBMS on an ML runtime backend

Automatic Differentiation

Example Query Inputs

Example Query Outputs

SELECT Digit, Size, COUNT(*)
FROM parseImageToTable(Image)
GROUP BY Digit, Size

Trainable Table Value Function

Example Query Outputs

<table>
<thead>
<tr>
<th>Digit</th>
<th>Size</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Large 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Small 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Large 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Small 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Large 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Small 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Large 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Small 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Large 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Small 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Large 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Small 2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Large 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Small 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Large 2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Small 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Large 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Small 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Large 0</td>
<td>0</td>
</tr>
</tbody>
</table>
AI-Centric Database System

Broader implications of having a DBMS on an ML runtime backend

Tensorframes as accelerator for Pandas Dataframes
Apache SystemDS

**APIS:** Command line, JMLC, Spark MLContext, Spark ML, (20+ Scalable Algorithms)

- [SIGMOD’15,’17,’19,’21abc,’23abc,’24]
- [PVLDB’14,’16ab,’18,’22]
- [ICDE’11,’12,’15]
- [CIDR’17,’20]
- [VLDB’18]
- [CIKM’22]
- [DEBull’14]
- [PPoPP’15]

**In-Memory Single Node** (scale-up)

- GPUs since 2014/16
- Other Prototype Backends:
  - Netezza
  - Apache Flink

**Hadoop or Spark Cluster** (scale-out)

- since 2010/11
- since 2012
- since 2015

**Federated** (LA progs, PS)

- since 2019

- Write Once, Run Anywhere

- DML Scripts

- Language

- Compiler

- Runtime

- 07/2020 Renamed to SystemDS
- 05/2017 Apache Top-Level Project
- 11/2015 Apache Incubator Project
- 08/2015 Open Source Release
LinregDS (Direct Solve)

X = read($1);
y = read($2);
intercept = $3;
lambda = 0.001;

... if(intercept == 1) {
    ones = matrix(1, nrow(X), 1);
    X = append(X, ones);
}

I = matrix(1, ncol(X), 1);
A = t(X) %*% X + diag(I)*lambda;
b = t(X) %*% y;
beta = solve(A, b);
... write(b, $4);

Scenario:
X: $10^8 \times 10^3, 10^{11}$
y: $10^8 \times 1, 10^8$

HOP DAG (after rewrites)

Cluster Config:
• driver mem: 20 GB
• exec mem: 60 GB

→ Distributed Matrices
• Fixed-size matrix blocks
• Data-parallel operations

Other Systems:
In-RDBMS, RIOT

Hybrid Runtime Plans:
• Size propagation / memory estimates
• Integrated CP / Spark runtime
• Dynamic recompilation during runtime
Apache SystemDS: Rewrites

- **Example Static Rewrites** (size-independent)
  - Common Subexpression Elimination
  - Constant Folding / Branch Removal / Block Sequence Merge
  - Static Simplification Rewrites
  - Right/Left Indexing Vectorization
  - For Loop Vectorization
  - Spark checkpoint/repartition injection

- **Example Dynamic Rewrites** (size-dependent)
  - Dynamic Simplification Rewrites
  - Matrix Mult Chain Optimization

---

**Example Static Rewrites**

- \( \text{trace}(X\%\%Y) \rightarrow \text{sum}(X*t(Y)) \)
  - \( O(n^3) \)
  - \( Y \)

- \( \text{sum}(\lambda*X) \rightarrow \lambda*\text{sum}(X) \)
  - \( X \)

- \( \text{sum}(X+Y) \rightarrow \text{sum}(X) + \text{sum}(Y) \)
  - \( O(n^2) \)
  - \( X^* Y^\top \)

---

**Example Dynamic Rewrites**

- \( \text{rowSums}(X) \rightarrow X, \iff ncol(X)=1 \)
- \( \text{sum}(X^2) \rightarrow X\%\%t(X), \iff ncol(X)=1 \)

---

**Sparsity Estimation & Sparse DP Enum**

- TensorFlow, PyTorch

---

**Other Systems:**

- DEBull’14, CIDR’17, SIGMOD’19
- MNC @ SIGMOD’19

---

**Size propagation and sparsity estimation**

- 2,002 MFLOPs
- 4 MFLOPs
Apache SystemDS: Operator Fusion & Codegen

- **Motivation:** DAGs of linear algebra (LA) operations and statistical functions with materialized intermediates → **ubiquitous fusion opportunities**

- **Examples**

  a) Intermediates
  
  b) Single-Pass
  
  c) Multi-Aggregates

  d) Sparsity Exploitation

- **Other Systems:**
  - BTO, Tupleware, Kasen, Weld, TACO, Julia, TF XLA, JAX, TVM, DAPHNE, PyTorch, Triton
SystemDS: Operator Fusion & Codegen, cont.

- **MLogreg Inner Loop** (main expr on feature matrix X)

1: \( Q = P[, 1:k] \ast (X \%\% v) \)
2: \( H = t(X) \%\% (Q - P[, 1:k] \ast \text{rowSums}(Q)) \)

```java
public final class TMP25 extends SpoofRow {
    public TMP25() {
        super(RowType.COL_AGG_B1_T, true, 5);
    }
    protected void genexecDense(double[] a, int ai, SideInput[] b, double[] c, ...
        int len) {
        double[] TMP11 = getVector(b[1].vals(rix), ...);
        double[] TMP12 = vectMatMult(a, b[0].vals(rix), ...);
        double[] TMP13 = vectMult(TMP11, TMP12, 0, 0, ...);
        double TMP14 = vectSum(TMP13, 0, TMP13.length);
        double[] TMP15 = vectMult(TMP11, TMP14, 0, ...);
        double[] TMP16 = vectMinus(TMP13, TMP15, 0, 0, ...);
        vectOuterMultAdd(a, TMP16, c, ai, 0, 0, ...);
    }
    protected void genexecSparse(double[] avals, int[] aix, int ai, SideInput[] b,
        int len) {...}
}
```
Apache SystemDS: Lineage-based Reuse

• **Lineage as Key Enabling Technique**
  - Trace lineage of ops (incl. non-determinism), **dedup for loops/funcs**
  - Model versioning, **data reuse**, incr. maintenance, autodiff, debugging

• **Full Reuse of Intermediates**
  - Before executing instruction, probe output lineage in cache
    Map<Lineage, MatrixBlock>
  - Cost-based/heuristic caching and eviction decisions (compiler-assisted)

• **Partial Reuse of Intermediates**
  - **Problem:** Often partial result overlap
  - Reuse partial results via dedicated rewrites (compensation plans)

Other Systems:
COLUMBUS, KeystoneML, Helix, PRETZEL, MISTIQUE, Alpine Meadow, Collaborative Optimizer

Example:

```
for( i in 1:numModels )
  R[,i] = lm(X, y, lambda[i], ...)
```

```
m_lmDS = function(...) {
  l = matrix(reg, ncol(X), 1)
  A = t(X) %*% X + diag(l)
  b = t(X) %*% y
  beta = solve(A, b) ...
}
m_steplm = function(...) {
  while( continue ) {
    parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
        Xi = cbind(Xg, X[,i])
        B[,i] = lm(Xi, y, ...)
      }
    }
    # add best to Xg (AIC)
  }
} 
```
Apache SystemDS: Workload-aware CLA

- **Lossless Matrix Compression**
  - Improved general applicability (adaptive compression time, new compression schemes, new kernels, intermediates, workload-aware)
  - Sparsity → **Redundancy exploitation** (data redundancy, structural redundancy)

- **Workload-aware Compression**
  - Workload summary → compression
  - Compressed Representation → execution planning

Other Systems:
TOC, Spark/Flink, NetCDF/HDF5, SciDB, TileDB, Sprintz, Grammar, Factorized

User Script:
```r
X = read("data/X")
y = read("data/y")
X = scale(X,TRUE,TRUE)
w = 12svm(X,y,TRUE, 1e-9,1e-3,100)
write(w,"data/wXy")
```

Workload Tree

Built-in Functions:
```r
if(shift) X = X - colMeans(X)
if(scale) X = X / colSds(X)
```

Cost Summary
```
\begin{tabular}{c|c|c|c|c|c}
|   | 0 | 100 | 10 | 10 | 105 | 0 |
\end{tabular}
```
Runtime for Tensor-Relational Computations

• **Tensor-based computations can easily be specified relationally**
  - Why? Consider Einstein summation notation

• **These are all relational computations!**
  - Always an (optional) aggregation
  - On top of a projection
  - On top of an (optional) join tree

• **Ex: MatMul**
  - First join A and B on $j$ index
  - Then projection to multiply matched entries
  - Then aggregate, grouping on $i$ and $k$ indices

\[
\forall_{i,k} C_{i,k} \leftarrow \sum_j A_{i,j} \times B_{j,k} - \sum_{\emptyset} \text{ReLU}(A_i) - \sum_j W_{i,j}^{(2)} \times B_j
\]

\[
D_\emptyset \leftarrow \sum_i \exp(C_i)
\]

\[
E_i \leftarrow \sum_{\emptyset} \frac{\exp(C_i)}{D_\emptyset}
\]

General Relativity [Einstein16]
But Pure Relational Can Be Slow

- **Relational is good**: we know how to scale relational computations
  - But pure relational (each entry in a tensor is a tuple) will not perform well
  - Why? Relational mult of two 80K by 80K matrices produces 512 trillion intermediate tuples
  - Small overhead associated with each tuple means performance is poor

- **Tensor relations** allow the best of both

Consider the matrix:

\[
\begin{bmatrix}
  1.4 & 2.2 & 1.2 & 2.1 \\
  2.3 & 2.6 & 1.1 & 2.2 \\
  1.4 & 1.0 & 1.1 & 1.4 \\
  1.1 & 1.4 & 2.5 & 2.3 
\end{bmatrix}
\]

Decompose into a set of \((\text{rowID}, \text{colID}, \text{chunk})\) triples:

\[
\tilde{R} = \left\{ \left( 1, 1 \right), \left[ \begin{array}{ccc} 1.4 & 1.2 \\ 2.3 & 2.6 \end{array} \right] \right), \left( 1, 2 \right), \left[ \begin{array}{ccc} 1.2 & 2.1 \\ 1.1 & 2.2 \end{array} \right] \right), \left( 2, 1 \right), \left[ \begin{array}{ccc} 1.4 & 1.0 \\ 1.1 & 1.4 \end{array} \right] \right), \left( 2, 2 \right), \left[ \begin{array}{ccc} 1.1 & 1.4 \\ 2.5 & 2.3 \end{array} \right] \right) \}
\]
Tensor Relations

- These are relations
  - So distributing to multiple machines/devices (GPUs) is easy
  - Scaling to very large computations (operands don’t fit in GPU RAM) is also easy
  - But actual data manipulations over sub-tensors done with efficient CPU/GPU kernels

- Example: MatMul over tensor relations

```
SELECT lhs.rowID, rhs.colID
    mat_sum (mat_mul (lhs.chunk, rhs.chunk))
FROM A AS lhs, B AS rhs
WHERE lhs.colID = rhs.rowID
GROUP BY lhs.rowID, rhs.colID
```

Imagine two 80K by 80K matrices decomposed into 1K by 1K chunks

Now only 80 X 80 X 80 = 512K intermediate tuples from join; low overhead, but plenty of parallelism
Tensor-Relational Computations: Use a DBMS?

- DBMS engine can be improved. Consider MatMul on multiple machines/devices

Imagine two 80K by 80K matrices decomposed into 6400, 1K by 1K chunks; perform compute on 256 machines/devices

```
SELECT lhs.rowID, rhs.colID
    mat_sum (mat_mul (lhs.chunk, rhs.chunk))
FROM A AS lhs, B AS rhs
WHERE lhs.colID = rhs.rowID
GROUP BY lhs.rowID, rhs.colID
```

Join produces 6400 x 80 result chunks
Hash partitioned into 6400 buckets to aggregate

Join can only keep 80 machines/devices busy (just 80 unique join keys)
Agg requires transferring approximately 6400 x 80 = 512000 chunks across machines/devices

However, there’s an algorithm that can do the mult with 3 x 8 x 6400 = 153600 xfers, all 256 busy
Can Do Much Better

Each proc. gets $\frac{6400}{64} = 100$ chunks of A

Can Do Much Better

Optimal 3D algorithm

Replicate A $m^{1/3}$ times

Replicate B $m^{1/3}$ times

Reduce: $m^{1/3}$ xfers of partial outputs

Asymptotically less comm than relational by a $m^{1/3}$ factor

Each processor performs pairwise mult of local chunks
What Is the Issue?

• **DBMS treats all relational ops as separate**
  • Best way to do the join: hash A, hash B—just xfer A and B once
  • But this sets you up for a terrible aggregation...
  • Xfer equivalent to 80x the size of input matrix in our example
  • Compare to 8x in the case of the 3D in our example

• **3D multiply xfers A and B eight times during join---so join is more expensive**
  • But then aggregation is set up nicely, so much faster
  • Trade-off: expensive join for inexpensive agg is not available to a DBMS
Tensor-Relational: New Engine Needed

- Such computations are fundamentally different from classical relational
  - Few tuples, but very large
  - Computationally intensive part is running the kernels, not linking tuples
  - Time required to transfer subtensors across machines dominates

- How should a tensor-relational engine work?
  - Use relational lineage as input to an optimization problem
  - Where to place kernels
    - So as to minimize communication, while ensuring load balancing
  - Can plan everything, down to swaps in and out of device memory!
  - Then execute on a specialized dataflow engine

Given this
- First create variants of A and B that have tensor IDs, dropping tensors
- Execute the computation over tuples with tensors removed, but only collect lineage
How Well Does This Work?

• **Simple comparison**
  - Tensor-relational runtime with lineage-based planning vs. broadcast matrix multiply
  - CPU cluster, Amazon EC2
  - Multiply non-square matrices of size $N \times K$ and $K \times M$, split into 1K by 1K chunks

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>K</th>
<th>BMM</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1K</td>
<td>1K</td>
<td>1000K</td>
<td>17.19s</td>
<td>5.03s</td>
</tr>
<tr>
<td>1K</td>
<td>40K</td>
<td>40K</td>
<td>16.98s</td>
<td>7.49s</td>
</tr>
</tbody>
</table>

Cluster with 2 workers

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>K</th>
<th>BMM</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1K</td>
<td>1K</td>
<td>1000K</td>
<td>16.62s</td>
<td>2.99s</td>
</tr>
<tr>
<td>1K</td>
<td>40K</td>
<td>40K</td>
<td>16.87s</td>
<td>3.15s</td>
</tr>
</tbody>
</table>

Cluster with 7 workers

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>K</th>
<th>BMM</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1K</td>
<td>1K</td>
<td>1000K</td>
<td>18.40s</td>
<td>2.50s</td>
</tr>
<tr>
<td>1K</td>
<td>40K</td>
<td>40K</td>
<td>19.78s</td>
<td>2.57s</td>
</tr>
</tbody>
</table>

Cluster with 16 workers
References

[DaMoN’23] Wei Cui, Qianxi Zhang, Spyros Blanas, Jesús Camacho-Rodriguez, Brandon Haynes, Yinan Li, Ravi Ramamurthy, Peng Cheng, Rathijit Sen, Matteo Interlandi: Query Processing on Gaming Consoles. DaMoN 2023


[SIGMOD’23] Sebastian Baunsgaard, Matthias Boehm: AWARE: Workload-aware, Redundancy-exploiting Linear Algebra. SIGMOD 2023

[CIDR’22] Patrick Damme et al.: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. CIDR 2022

[CIKM’22] Sebastian Baunsgaard, Matthias Boehm, Kevin Innerebner, Mito Kehayov, Florian Lackner, Olga Ovcharenko, Arnab Phani, Tobias Rieger, David Weissteiner, Sebastian Benjamin Wrede: Federated Data Preparation, Learning, and Debugging in Apache SystemDS. CIKM 2022


References, cont.


Botong Huang, Matthias Boehm, Yuanyuan Tian, Berthold Reinwald, Shirish Tatikonda, Frederick R. Reiss: Resource Elasticity for Large-Scale Machine Learning. SIGMOD 2015


Conclusions

• **Future:** Broader Focus (on General Tensor Computations)
  • General linear algebra programs and tensor computations
  • Different architectures (parameter servers, data-/task-task parallel)
  • Wide variety of applications and workload characteristics

• **Long-term Benefits**
  • Simplicity
  • Reuse of Compilation and Runtime Techniques
  • Performance and Scalability

• Lots of **Open Challenges** and Research Opportunities