Scalable Linear Algebra on a Relational Database System

...

Pavel Grechanuk

Authors: Shangyu Luo, Zakai J, Gao, Michael Gubanov, Luis L. Perez, and Christopher Jermaine

Introduction

- Machine learning and large scale statistical processing in an important application domain and require linear algebra

Introduction

- Machine learning and large scale statistical processing in an important application domain and require linear algebra
- Many distributed linear algebra computations have closely corresponding distributed relational algebra computations

Introduction

- Machine learning and large scale statistical processing in an important application domain and require linear algebra
- Many distributed linear algebra computations have closely corresponding distributed relational algebra computations
- Given this the authors argue that it is natural to build a distributed linear algebra on top of a relational database

Why would we want to do this?

- Relational databases reap the benefits of decades of research targeted at building efficient systems

Why would we want to do this?

- Relational databases reap the benefits of decades of research targeted at building efficient systems
- Relational systems already have components such as a cost based query optimizer to aid in performing efficient computations

Why would we want to do this?

- Relational databases reap the benefits of decades of research targeted at building efficient systems
- Relational systems already have components such as a cost based query optimizer to aid in performing efficient computations
- Much of the work that goes into developing a scalable linear algebra system requires implementing functionality that is similar to a database query optimizer

What are the benefits?

- Most of the worlds data sits in relational databases -> if the databases could do linear algebra -> the data would be sitting in linear algebra systems meaning:

What are the benefits?

- Most of the worlds data sits in relational databases -> if the databases could do linear algebra -> the data would be sitting in linear algebra systems meaning:

1. It would eliminate the extract-transform-reload workflow that is expensive to do and requires transfering the data from one system to another

What are the benefits?

- Most of the worlds data sits in relational databases -> if the databases could do linear algebra -> the data would be sitting in linear algebra systems meaning:

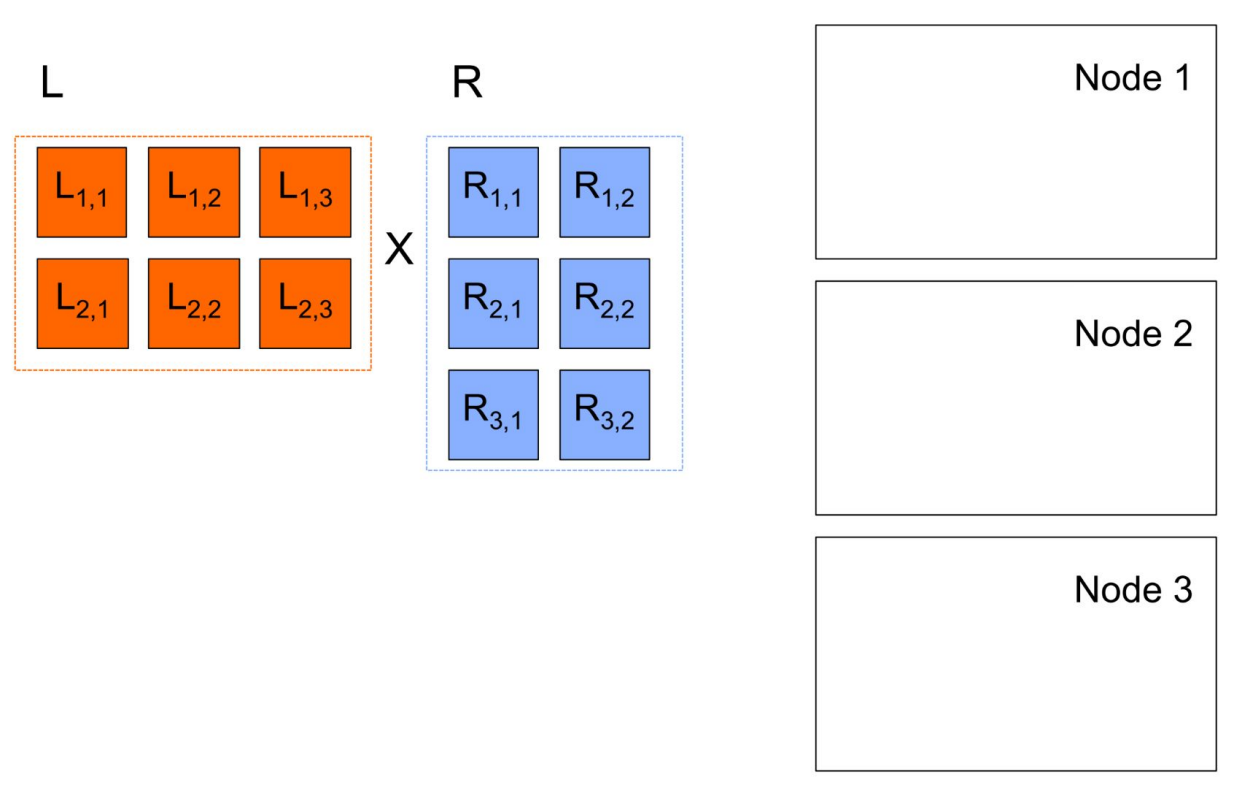
1. It would eliminate the extract-transform-reload workflow that is expensive to do and requires transfering the data from one system to another
2. People would not have to adopt yet another data processing system to perform linear algebra

What are the benefits?

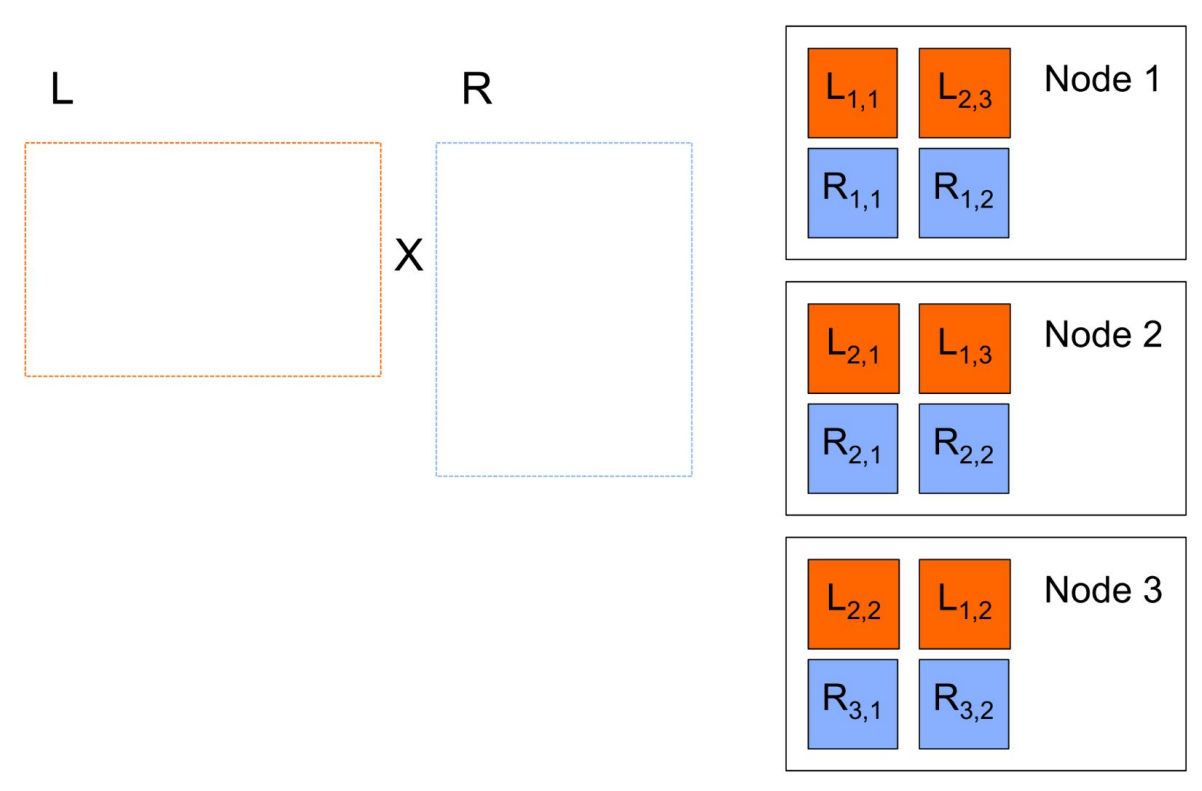
- Most of the worlds data sits in relational databases -> if the databases could do linear algebra -> the data would be sitting in linear algebra systems meaning:

1. It would eliminate the extract-transform-reload workflow that is expensive to do and requires transfering the data from one system to another
2. People would not have to adopt yet another data processing system to perform linear algebra
3. The design and implementation of high performance relational database systems is well understood, and if it is possible to adapt this system to perform linear algebra much of the past research is directly applicable

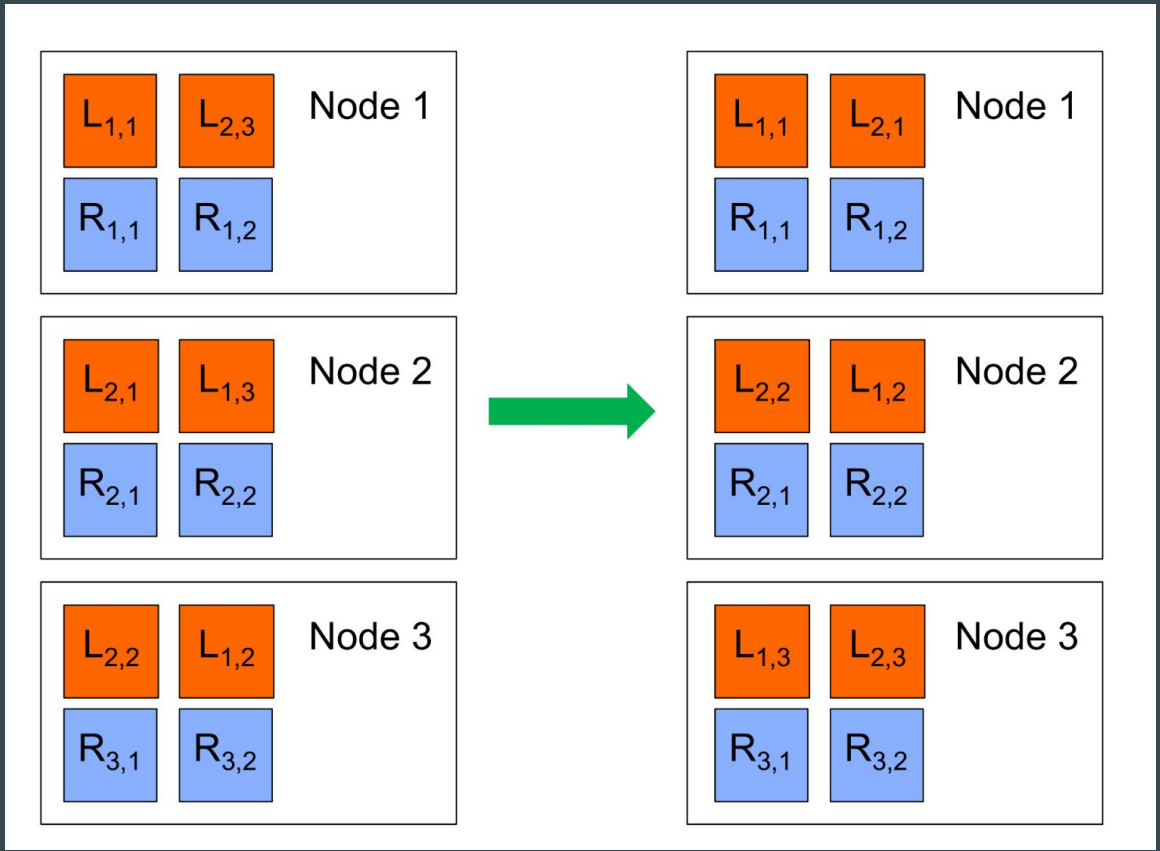
Typical Linear Algebra Computation - Divide Matrices



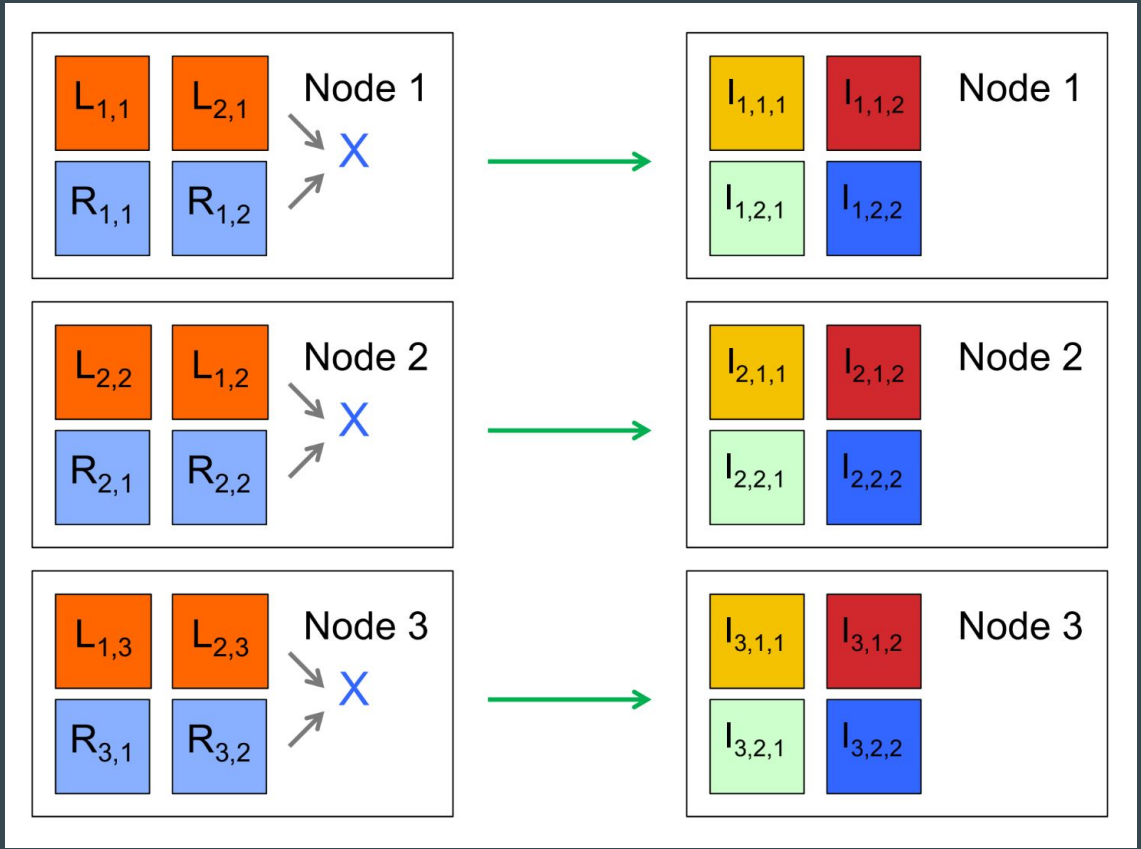
Typical Linear Algebra Computation - Partition Blocks



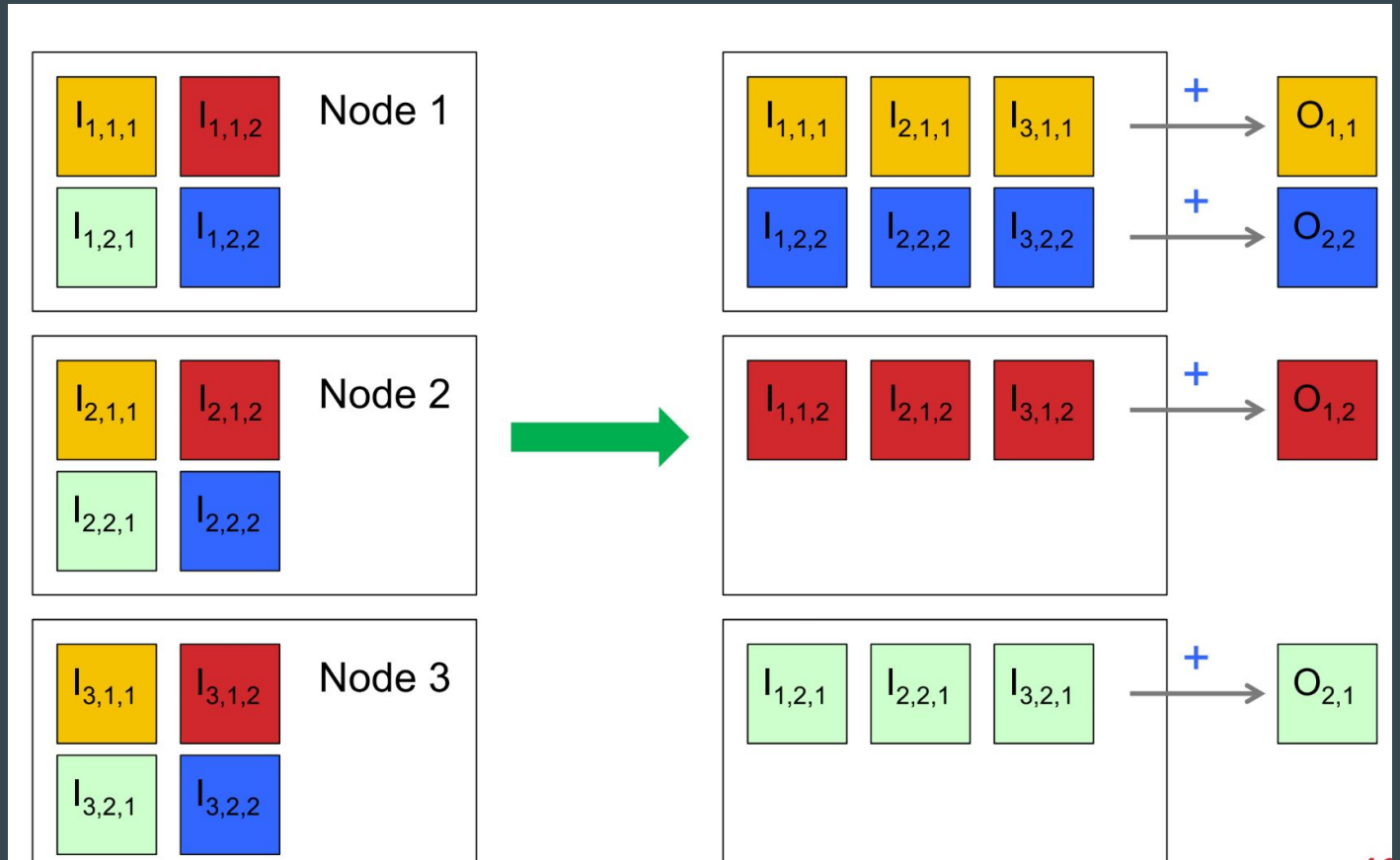
Typical Linear Algebra Computation - Shuffle Blocks



Typical Linear Algebra Computation - Local Multiply



Typical Linear Algebra Computation - Shuffle and Sum

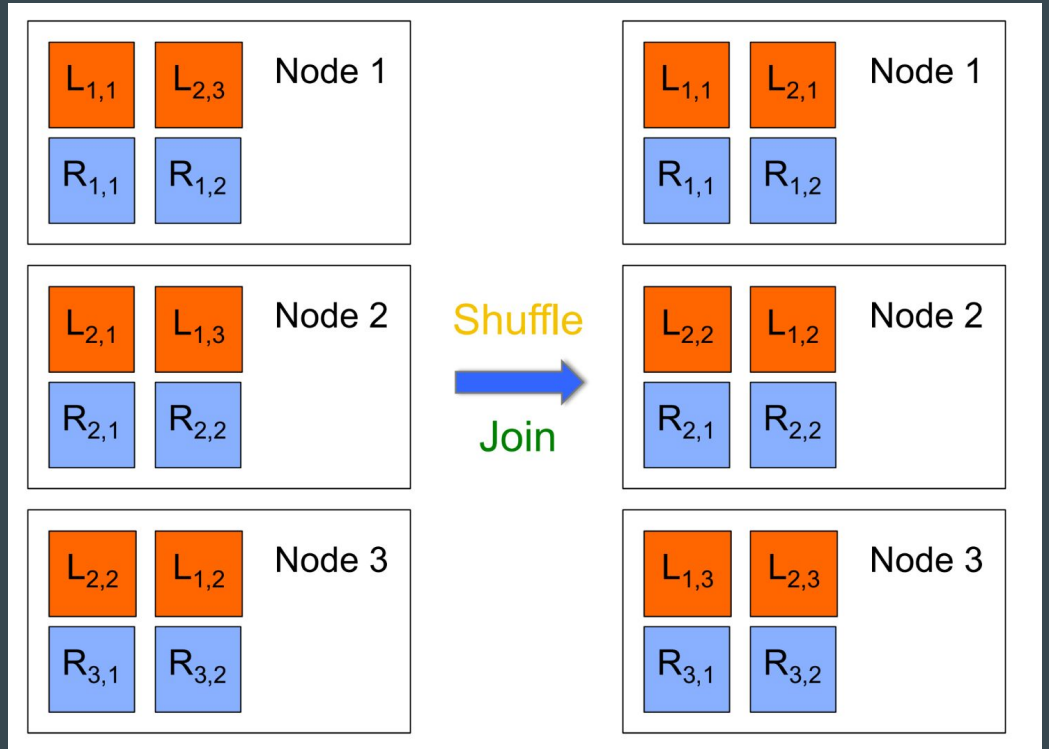


How does this relate to relational algebra?

- Key observation is that this is just a relational algebra computation over the two blocks L and R

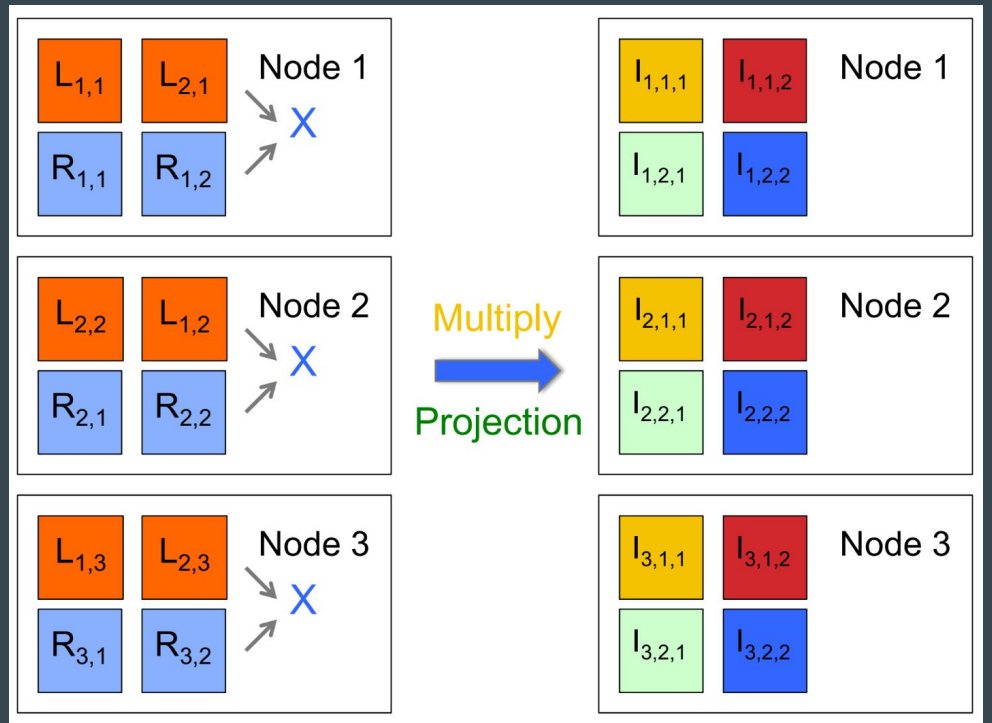
How does this relate to relational algebra?

- Shuffling the blocks is just a distributed join over the L(i,j) R(i,j) pairs!



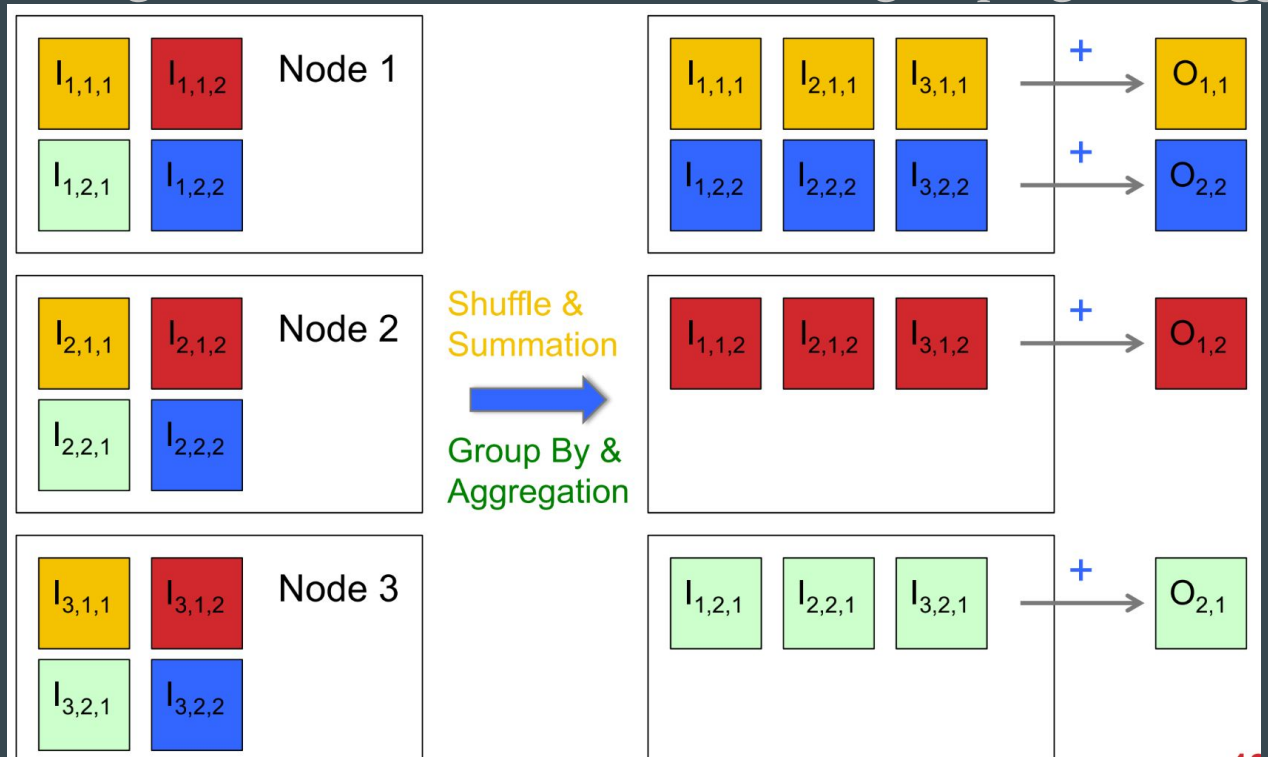
How does this relate to relational algebra?

- The matrix multiplication is just a projection!



How does this relate to relational algebra?

- The shuffling and summation is a distributed grouping with aggregation



How does this relate to relational algebra?

- Key observation is that this is just a relational algebra computation over the two blocks L and R
- This shows that distributed linear algebra computations are analogous to distributed relational algebra computations!

How does this relate to relational algebra?

- Key observation is that this is just a relational algebra computation over the two blocks L and R
- This shows that distributed linear algebra computations are analogous to distributed relational algebra computations!
- Using a relational database for linear algebra means we do not have to reinvent the wheel as we can benefit from decades of research into query optimization

Question:

Can performant distributed linear algebra be implemented on top of a relational database?

Question:

Can performant distributed linear algebra be implemented on top of a relational database?

Claim:

We can make a very small set of changes to a relational system to make it suitable for scalable in database linear algebra

Contributions

- SQL Extensions:
  a. New built in signed-scalar, vector and matrix data types
  b. New aggregate functions for constructing the above data types
  c. New linear algebra functions to operate on the data types - 22 functions like matrix multiplication, diagonal elements, dot product, etc.

Contributions

- SQL Extensions:
  a. New built in signed-scalar, vector and matrix data types
  b. New aggregate functions for constructing the above data types
  c. New linear algebra functions to operate on the data types - 22 functions like matrix multiplication, diagonal elements, dot product, etc.
- Made the semantics of linear algebra operations visible to the query optimizer

Contributions

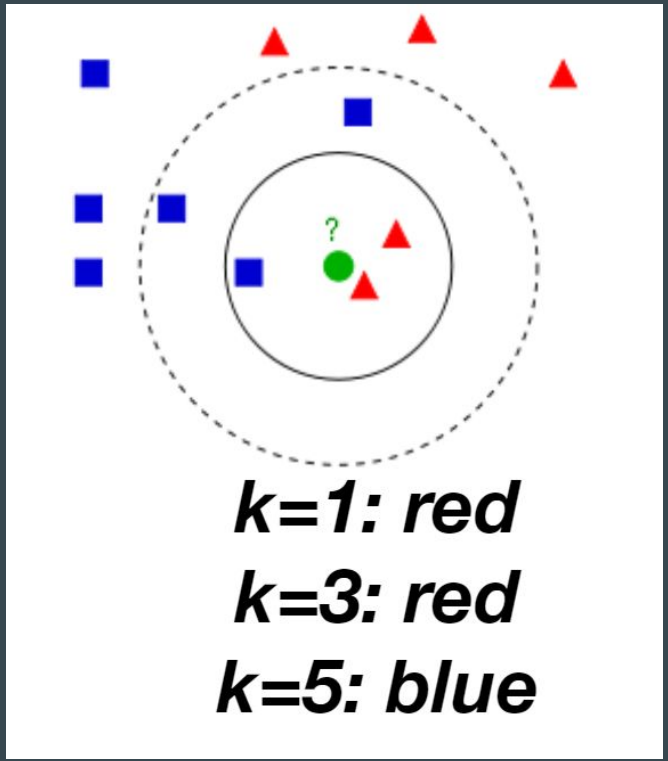
- SQL Extensions:
  a. New built in signed-scalar, vector and matrix data types
  b. New aggregate functions for constructing the above data types
  c. New linear algebra functions to operate on the data types - 22 functions like matrix multiplication, diagonal elements, dot product, etc.
- Made the semantics of linear algebra operations visible to the query optimizer
- Compared performance of this approach to standard linear algebra packages

Performance Comparison

- SimSQL
  - An distributed relational system with the authors ideas implemented on it
  - Three types of programs for tests: tuple based, vector based, and block based
- Apache SystemML
  - Scalable linear algebra system for machine learning
- SciDB
  - A scalable array database system
- Apache Spark with mllib.linalg library
  - A linear algebra library ran on top of a dataflow platform

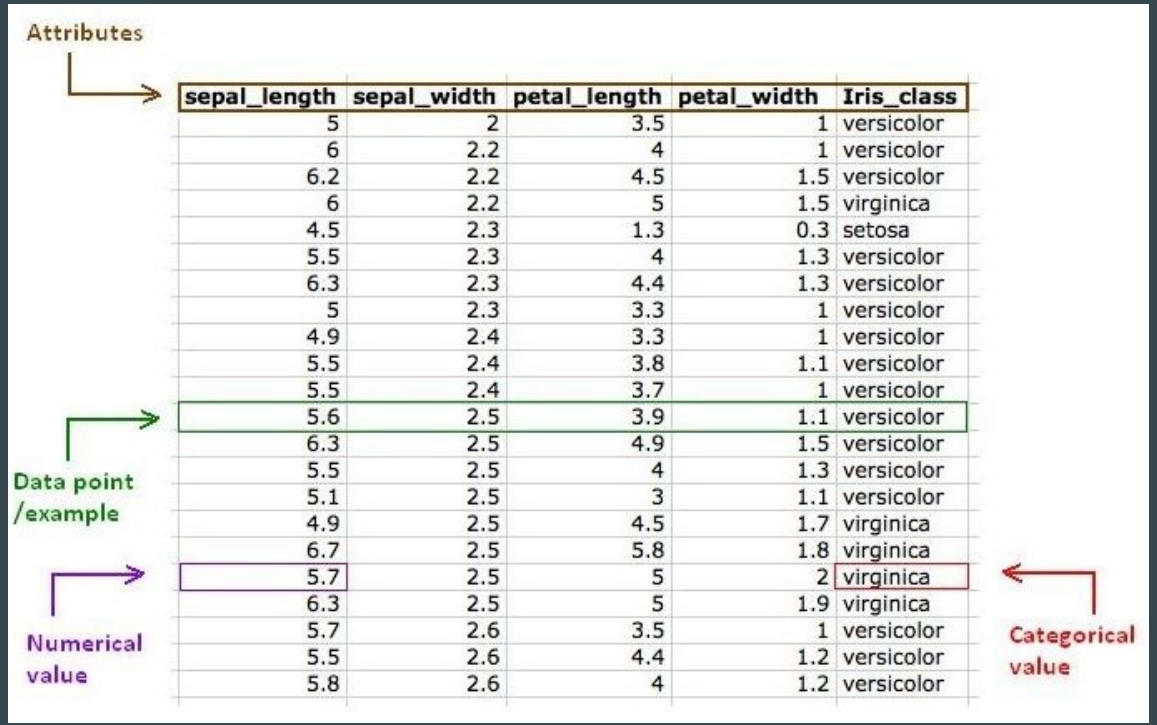
k Nearest Neighbor Classification - Review

- General idea is to find the k nearest neighbors using some distance metric
- Collect the predictions of the k nearest neighbors
- The class that has the largest number of predictions becomes the predicted class of the unknown instance



k Nearest Neighbor Classification - Review

- You could classify the type of an unknown flower using machine learning algorithms like k nearest neighbor
- If you had this data inside a relational database you would first have to take it out into this form



sepal_length	sepal_width	petal_length	petal_width	Iris_class
5	2	3.5	1	versicolor
6	2.2	4	1	versicolor
6.2	2.2	4.5	1.5	versicolor
6	2.2	5	1.5	virginica
4.5	2.3	1.3	0.3	setosa
5.5	2.3	4	1.3	versicolor
6.3	2.3	4.4	1.3	versicolor
5	2.3	3.3	1	versicolor
4.9	2.4	3.3	1	versicolor
5.5	2.4	3.8	1.1	versicolor
5.5	2.4	3.7	1	versicolor
5.6	2.5	3.9	1.1	versicolor
6.3	2.5	4.9	1.5	versicolor
5.5	2.5	4	1.3	versicolor
5.1	2.5	3	1.1	versicolor
4.9	2.5	4.5	1.7	virginica
6.7	2.5	5.8	1.8	virginica
5.7	2.5	5	2	virginica
6.3	2.5	5	1.9	virginica
5.7	2.6	3.5	1	versicolor
5.5	2.6	4.4	1.2	versicolor
5.8	2.6	4	1.2	versicolor

k Nearest Neighbor Classification - Review

Algorithm 3 KNN-PREDICT($\mathbf{D}$, K, $\hat{\boldsymbol{x}}$)

1: $S \leftarrow [\,]$
2: **for** $n = 1$ **to** N **do**
3: $\quad S \leftarrow S \oplus \langle \mathrm{d}(\boldsymbol{x}_n, \hat{\boldsymbol{x}}), n \rangle$ // store distance to training example n
4: **end for**
5: $S \leftarrow \text{SORT}(S)$ // put lowest-distance objects first
6: $\hat{y} \leftarrow 0$
7: **for** $k = 1$ **to** K **do**
8: $\quad \langle dist, n \rangle \leftarrow S_k$ // n this is the kth closest data point
9: $\quad \hat{y} \leftarrow \hat{y} + y_n$ // vote according to the label for the nth training point
10: **end for**
11: **return** $\text{SIGN}(\hat{y})$ // return $+1$ if $\hat{y} > 0$ and -1 if $\hat{y} < 0$

- L_2 norm: Euclidean distance:

$$L_2(x, y) = \sqrt{|x_1 - y_1|^2 + \cdots + |x_d - y_d|^2}$$

- L_1 norm: Manhattan distance:

$$L_1(x, y) = |x_1 - y_1| + \cdots + |x_d - y_d|$$

Example - k Nearest Neighbor Classification

```
CREATE VIEW xDiff (pointID, dimID, value) AS
 SELECT x2.pointID, x2.dimID, x1.value - x2.value
 FROM data AS x1, data AS x2
 WHERE x1.pointID = i and x1.dimID = x2.dimID

SELECT x.pointID, SUM (firstPart.value * x.value)
FROM (SELECT x.pointID AS pointID, a.colID AS
        colID, SUM (a.value * x.value) AS value
      FROM xDiff AS x, matrixA AS a
      WHERE x.dimID = a.rowID
      GROUP BY x.pointID, a.colID)
        AS firstPart, xDiff AS x
WHERE firstPart.colID = x.dimID
      AND firstPart.pointID = x.pointID
GROUP BY x.pointID
```

- Nested subquery and a view
- 4 joins and 2 groupings
- Difficult to understand the math

Example - k Nearest Neighbor Classification

```
CREATE VIEW xDiff (pointID, dimID, value) AS
 SELECT x2.pointID, x2.dimID, x1.value - x2.value
 FROM data AS x1, data AS x2
 WHERE x1.pointID = i and x1.dimID = x2.dimID

SELECT x.pointID, SUM (firstPart.value * x.value)
FROM (SELECT x.pointID AS pointID, a.colID AS
        colID, SUM (a.value * x.value) AS value
      FROM xDiff AS x, matrixA AS a
      WHERE x.dimID = a.rowID
      GROUP BY x.pointID, a.colID)
        AS firstPart, xDiff AS x
WHERE firstPart.colID = x.dimID
      AND firstPart.pointID = x.pointID
GROUP BY x.pointID
```

- Nested subquery and a view
- 4 joins and 2 groupings
- Difficult to understand the math

- If the data is dense and the dimensionality of the data is large (many dimID for each pointID)
- Execution of this query will move a huge number of small tuples
- In classical iteration based model there is a fixed cost per tuple
- If you have 10,000 points with 1,000 features each that is 10 million tuples
- Can we do better?

Example - k Nearest Neighbor Classification

```
SELECT x2.pointID,
   inner_product (
      matrix_vector_multiply (
         a.val, x1.val - x2.val),
         x1.val - x2.val) AS value
FROM data AS x1, data AS x2, matrixA AS a
WHERE x1.pointID = i
```

- Dramatically simpler
- Optimized for linear algebra
- Performance is improved significantly

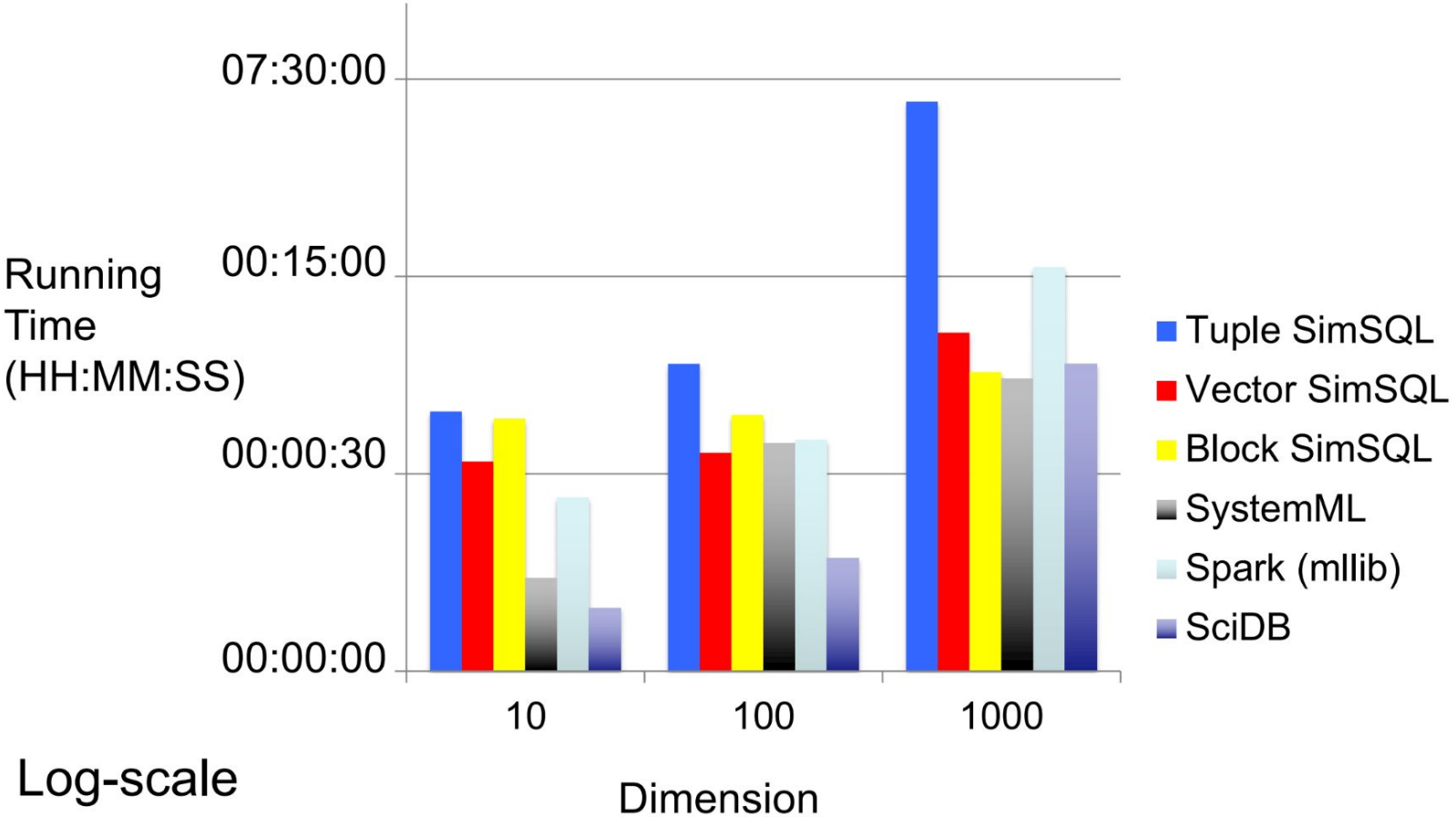
Example - k Nearest Neighbor Classification

```
SELECT x2.pointID,
   inner_product (
      matrix_vector_multiply (
         a.val, x1.val - x2.val),
         x1.val - x2.val) AS value
FROM data AS x1, data AS x2, matrixA AS a
WHERE x1.pointID = i
```

- Dramatically simpler
- Optimized for linear algebra
- Performance is improved significantly

- The vector and matrix representations can be manipulated as a single unit during query processing
- Significant performance increase!
- Simple addition of the vector and matrix data types allows for quick computation of what would be very complex queries

Gram Matrix Computation: $G = X^T X$



Gram Matrix Cost Comparison

- For the 1,000 dimensional data each tuple joins with the other 1,000 tuples for the same data point and then all of those need to be aggregated
- Since 5E5 data points are stored as 5E8 tuples, this results in 5E11 tuples that need to be aggregated
- Tuple based linear algebra -> even a tiny cost per tuple can result in very long run times

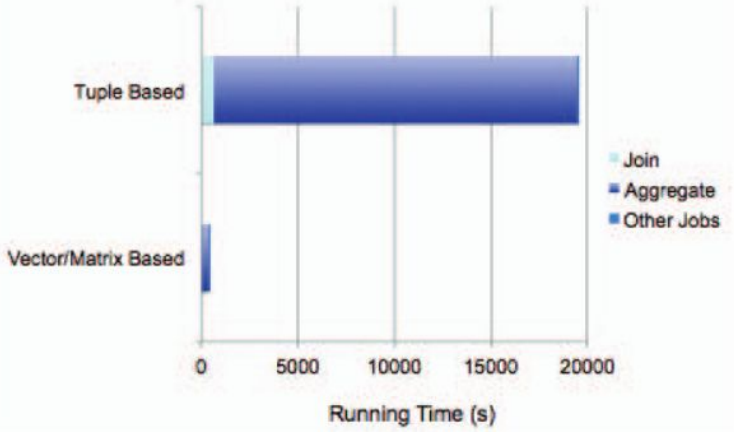


Fig. 7: Comparison of Gram matrix computation for tuple-based and vector-based SimSQL.

Linear Regression Review

- Predict a numerical value using the features X
- Assumes relationship between the features X and predicted quantity y is linear
- $\boldsymbol{\beta}$ is the coefficient vector that represents the relationship between each feature and target
- $\boldsymbol{\beta} = (X^T X)^{-1} X^T y$

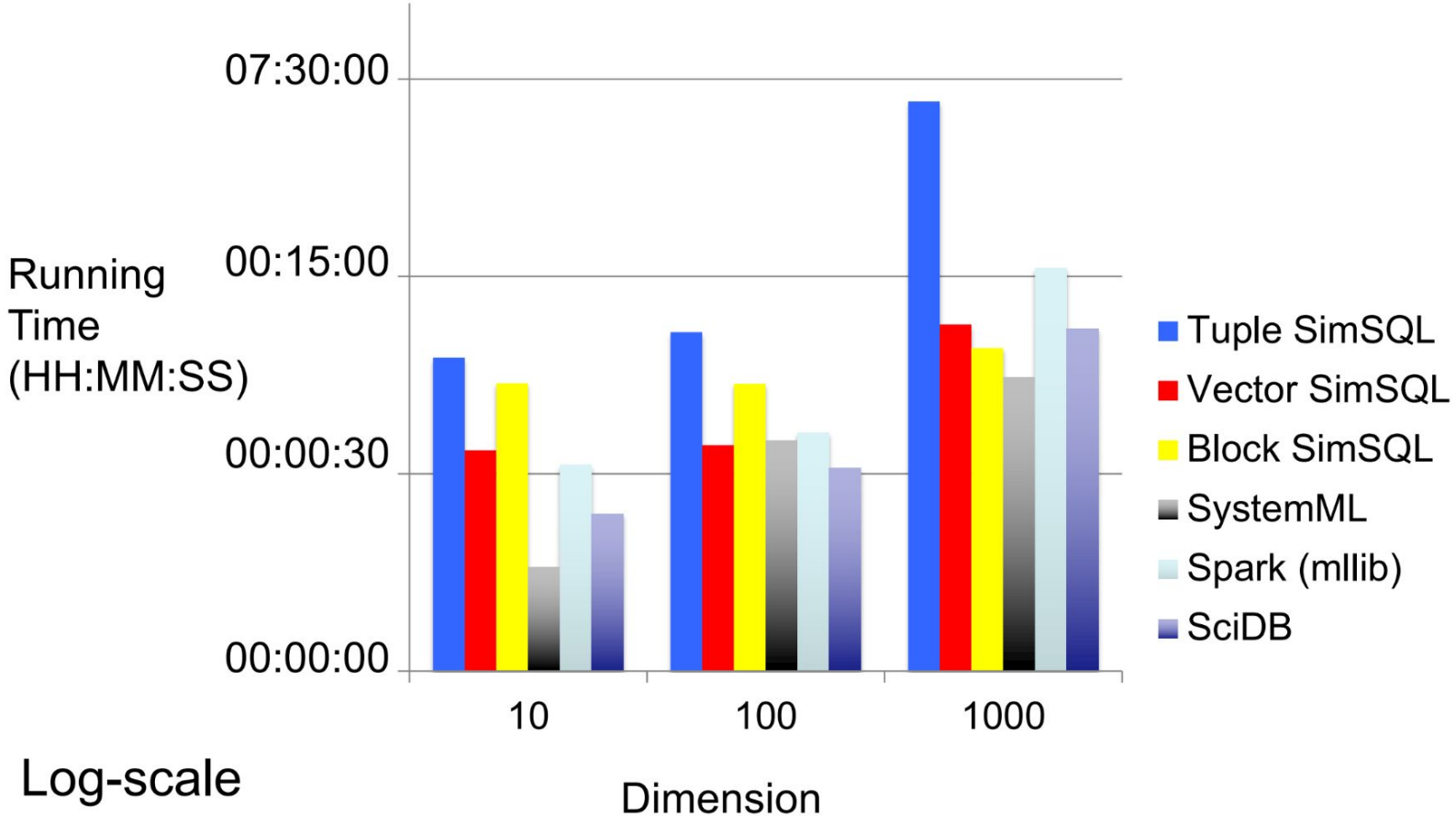
$$y_i = \beta_0 1 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^\mathsf{T} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

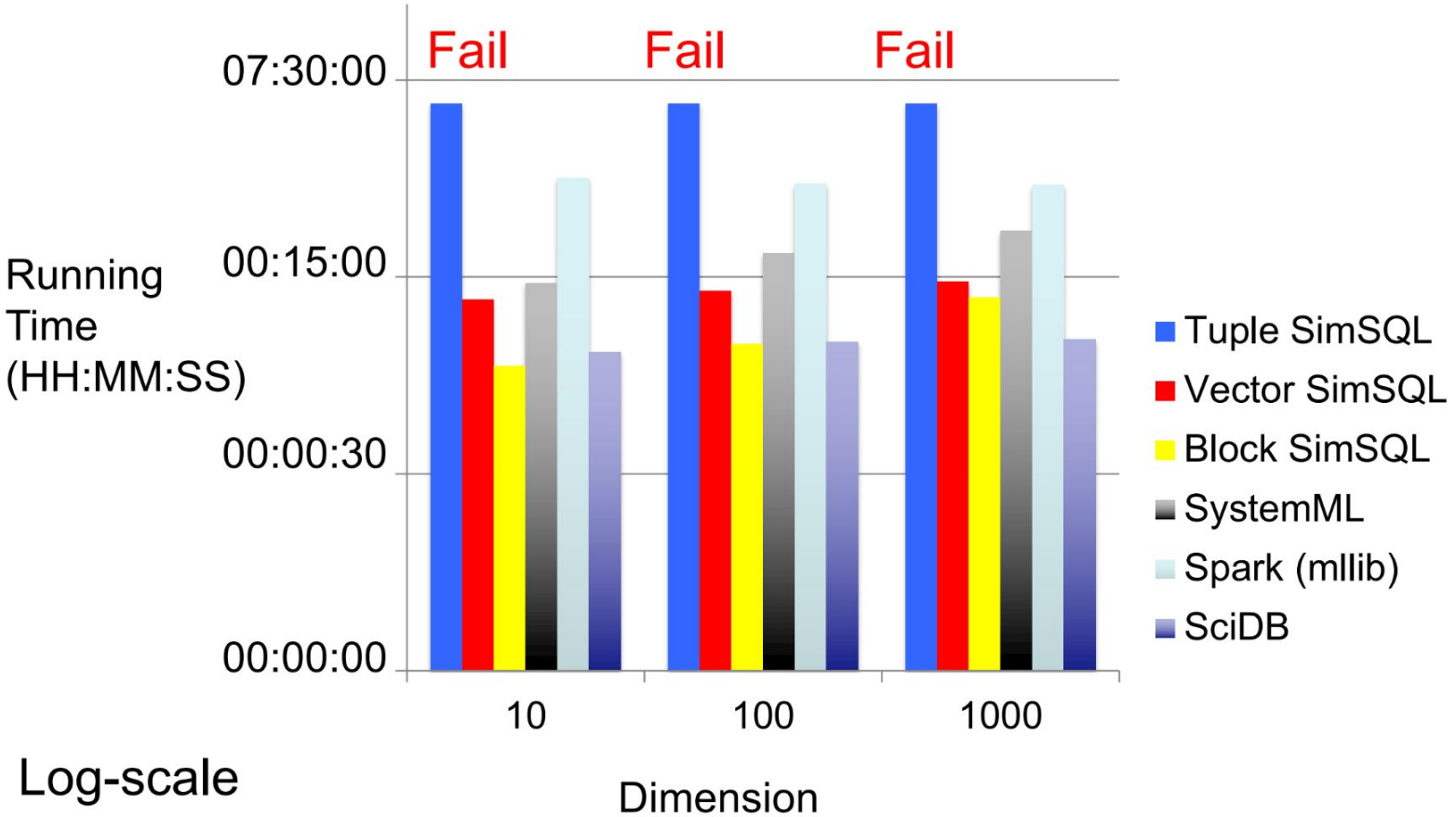
$$X = \begin{pmatrix} \mathbf{x}_1^\mathsf{T} \\ \mathbf{x}_2^\mathsf{T} \\ \vdots \\ \mathbf{x}_n^\mathsf{T} \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix},$$

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Linear Regression: $\hat{\beta} = (X^T X)^{-1} X^T y$



Outlier Detection: $\underset{i}{\text{argmax}}\left(\min_{x_j \neq x_i} (x_i - x_j)^T A (x_i - x_j)\right)$



Conclusion

- Introduced vector and matrix data types to a relational database
- Provided vector/matrix construction/deconstruction, linear algebra, and other vector/matric related functions
- Brought the semantics of linear algebra operations to the query optimizer
- Significantly increased the performance of a standard relational database by leveraging the additions

My Review

- Show their results in a clear, concise, and convincing manner
- Do their results support their hypothesis?
  - Yes they significantly improved the performance of performing linear algebra on a relational database
- Their work can be improved further as this is just a proof of concept study

Thank you!
Questions?