Scalable Linear Algebra on a Relational Database System

University at Buffalo The State University of New York

Why Scalable Linear Algebra System ?

- ❑ Data Analytics
- ❑ Machine Learning
- ❑ Large Scale Statistical Processing

Also, since Data Analytics has become an important application for Modern Data Management Systems!!

Note: All these important application domains require linear algebra for its computations.

Efforts Towards Building Complete Data Management Systems

- ❑ Support for Vectors, Matrices and Standard Operations on them
- ❑ Storage and retrieval of Data to/from disk
- ❑ Buffering / Caching of Data
- ❑ automatic logical/physical optimization of computations
- ❑ Recovery
- ❑ Special purpose domain specific language

Note: Did you notice that most of these features are already supported in a Relational Database System? Is this new system really necessary ?

Proposal

- ❑ A parallel or distributed system is an excellent platform upon which a scalable linear algebra system can be built.
- ❑ Most Relational Systems have Cost Based Optimization which can be leveraged for scaling linear algebra computations.
- ❑ If Scalable linear algebra is to be added to a modern dataflow platform such as Spark, they should be added on top of the system's more structured relational data abstractions.

Note: Also because data analytics has become an important application for modern Data Management Systems.

Obvious Benefits

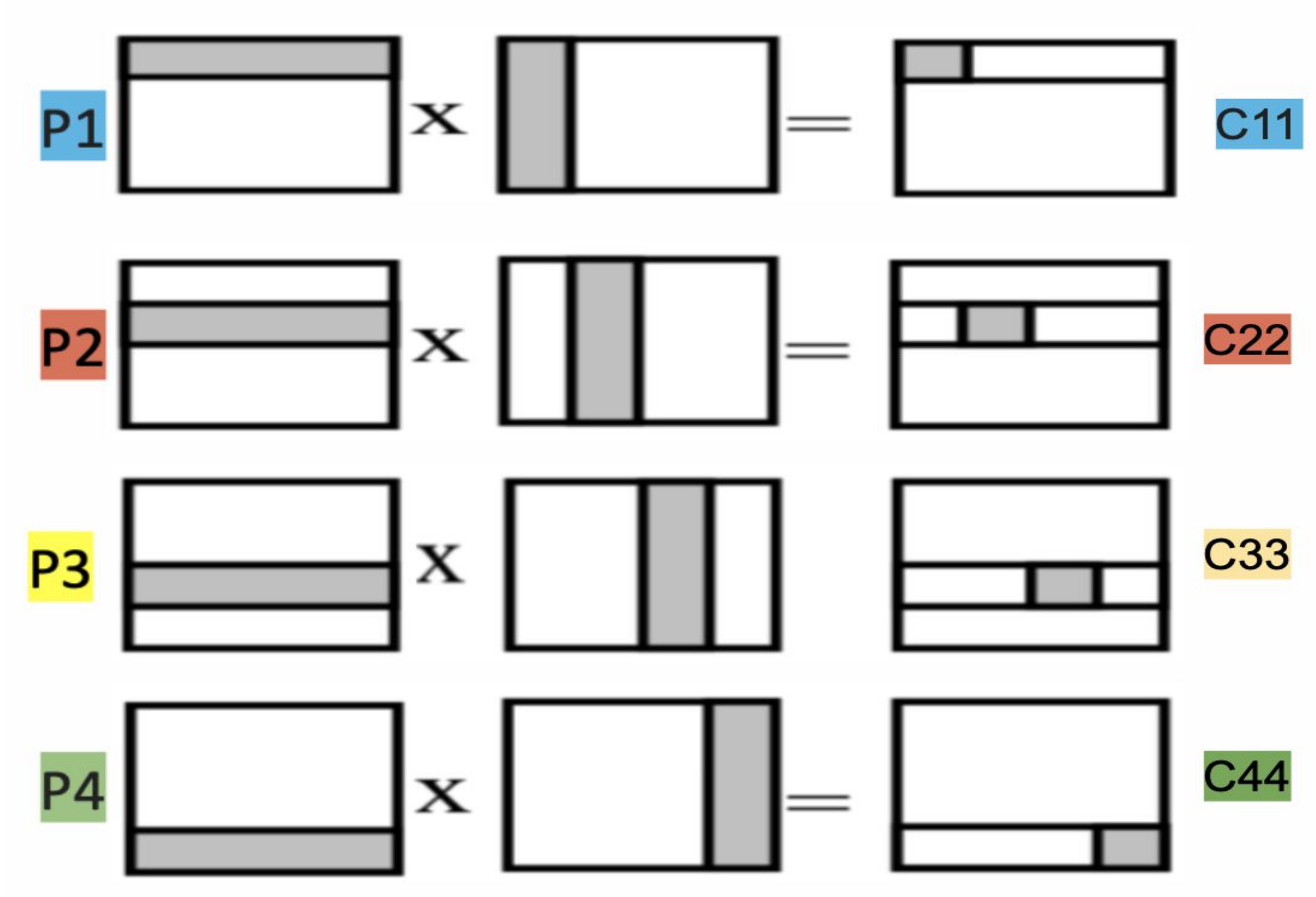
- Eliminate "extract-transform-reload nightmare".
- Eliminate the need to adopt yet another type of data processing system.
- Most or all of the decades worth of research aimed at distributed relational system, is directly applicable.

Proposed Changes

- Adding LABELED_SCALAR, VECTOR and MATRIX data types to SQL-based relational system.
- Make Relational Query optimizer "linear algebra aware"
- Changes to SQL that makes it easy to specify complicated computations over vectors and matrices.
- Language Mechanisms to support moving between relational data, vectors and matrices.

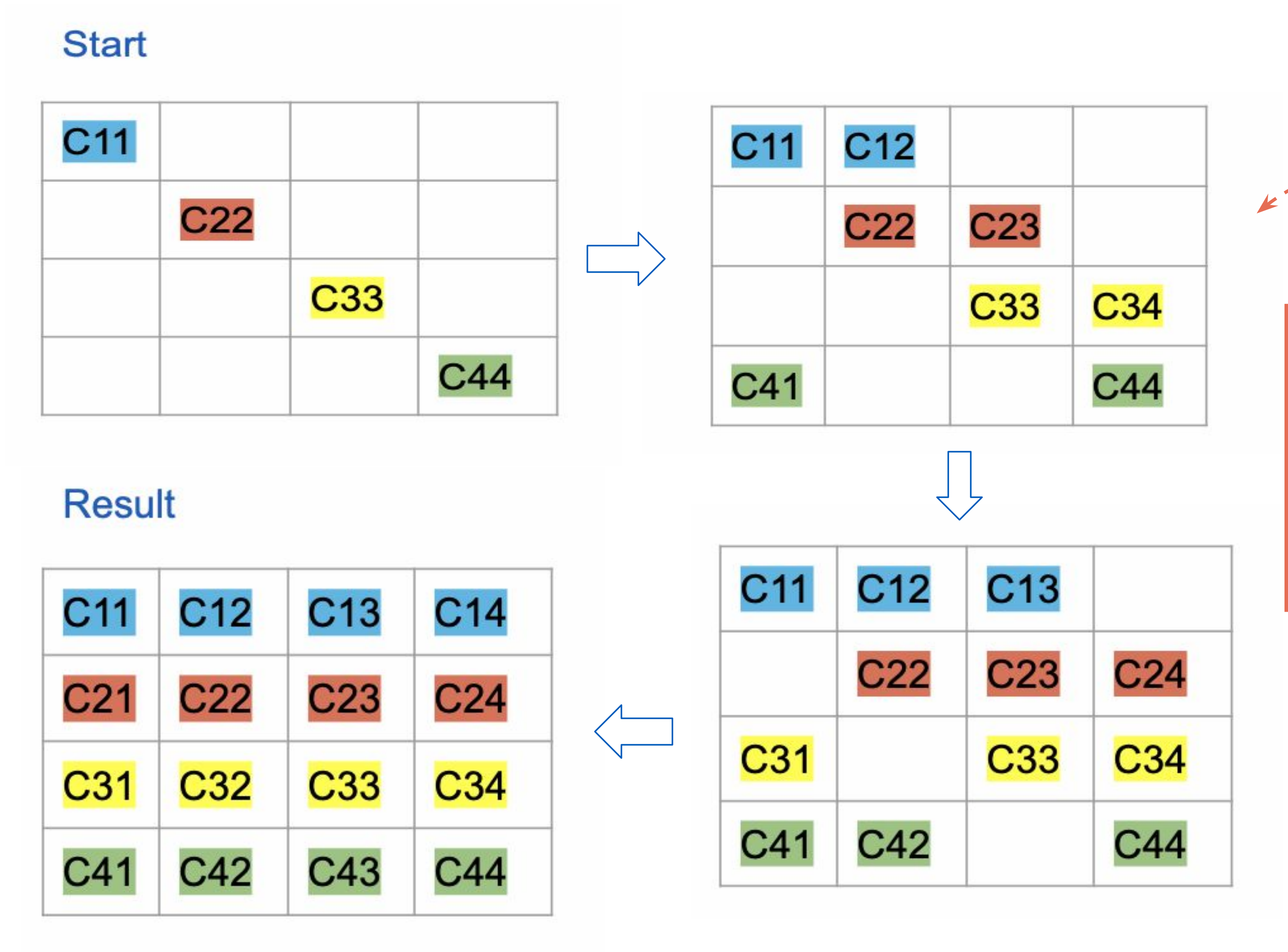
Distributed Multiplication of two Large Dense Matrices

First Iteration



- ❑ Matrix L is row partitioned
- ❑ Matrix R is Column partitioned

At Every step, each of the four processors compute the next block of C in their row in a cyclic fashion .To produce C, as depicted in the following slide.



Start

C11			
	C22		
		C33	
			C44

C11	C12		
	C22	C23	
		C33	C34
C41			C44

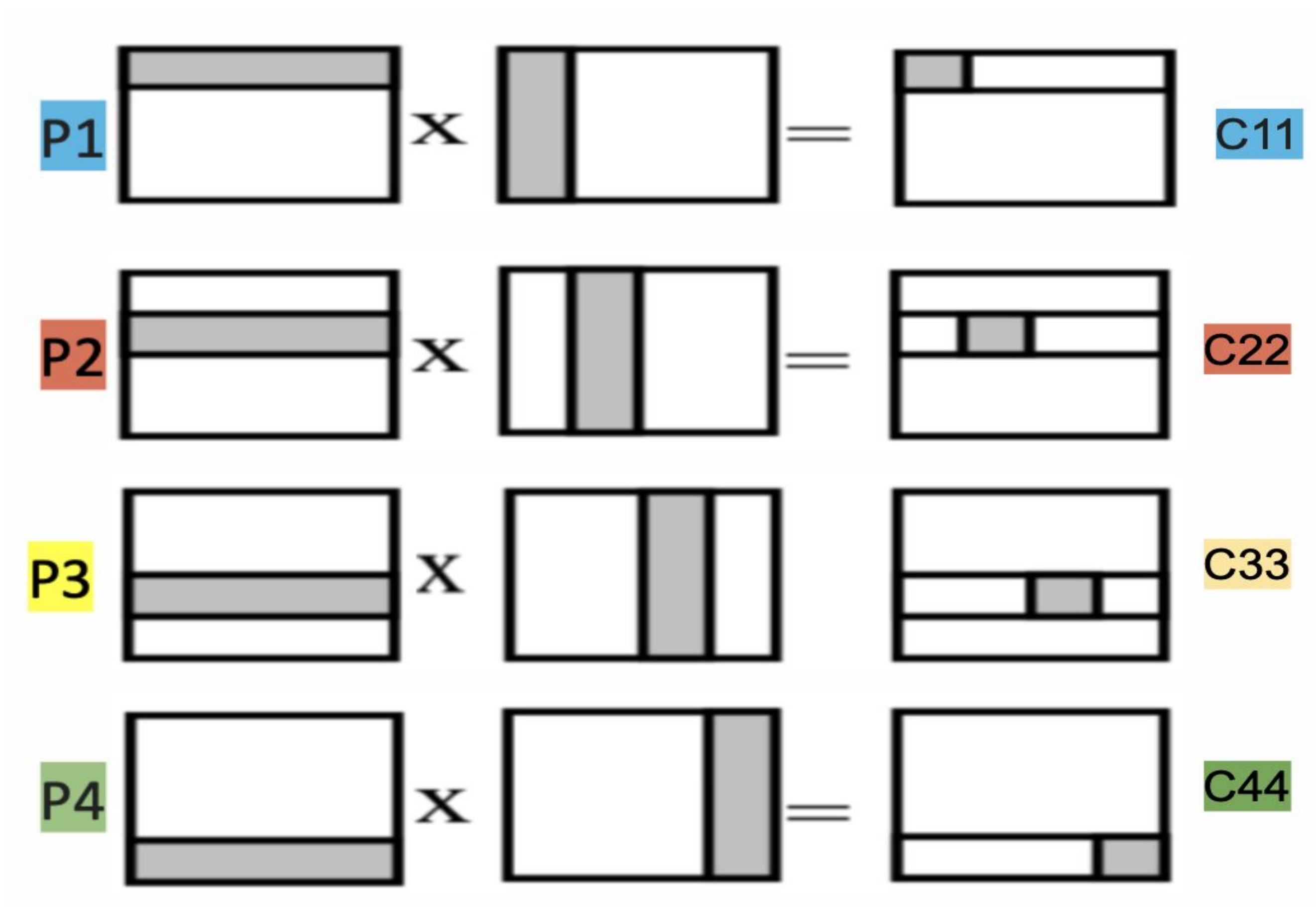
C11	C12	C13	
	C22	C23	C24
C31		C33	C34
C41	C42		C44

Result

C11	C12	C13	C14
C21	C22	C23	C24
C31	C32	C33	C34
C41	C42	C43	C44

At Every step, each of the four processors compute the next block of C in their row in a round-robin fashion.

Matrix Multiplication in Relational Algebra Terms



- A local join, in this case a cross product is performed by iterating through every row in block Lji to be combined with every column in Rik to compute every element in Cjk, through aggregation.
- This is just a Relational algebra computation over blocks making up Matrix L and Matrix R.
- Benefits of Query optimization are directly available.

Why are the proposed changes Necessary?

- Complexity of Writing Linear Algebra on top of SQL

- Performance: When expressing Linear Algebra through SQL performance will be detoriated as they require several joins and aggregate operations.

- In a classical Iterator-based execution model there is a fixed cost per tuple, which will translate to very high cost.

Solution

- ❑ Adding LABELED_SCALAR, VECTOR and MATRIX data types to SQL-based relational system.
- ❑ Extensions in SQL language for manipulating these types and moving between them.

Introducing these non-normalized data types cause the contents of vectors and matrices to be manipulated as a single unit during query processing, and will bring in significant performance.

Introducing New Types

At the very highest level, we propose adding VECTOR, MATRIX and LABELED_SCALAR column types to SQL and the relational model.

<u>example:</u>

create table m (mat MATRIX[10][10], vec VECTOR[100]);

Built in Operation

In addition to standard arithmetic operations, linear algebra operations are also defined over MATRIX and VECTOR types.

<u>example:</u>

```
SELECT matrix_multiply(mat,mat) from m;
SELECT mat *mat from m;
```

While the first produces Matrix product, the second produces Hadamard product of Matrix with itself.

Moving Between Types

Matrix can be represented in different forms and moving between them should be supported.

<u>example:</u>

mat(row INTEGER, col INTEGER, value DOUBLE) (or)

row_mat(row INTEGER, vec_value VECTOR[ ]) (or)

col_mat(col INTEGER, vec_value VECTOR[ ]) (or)

mat (value MATRIX [ ] [ ])

University at Buffalo The State University of New York

Denormalizing Vector Types

```
CREATE TABLE y (i Integer, Y_i Double);

SELECT VECTORIZE (label_scalar (Y_i, i)) FROM y
```

label_scalar function associates Y_i with label i.

VECTORIZE aggregates label_scalar into vector

Denormalizing Matrices

```
mat(row INTEGER, col INTEGER, value DOUBLE)


CREATE VIEW Vecs SELECT VECTORIZE(label_scalar (val,col))

AS vec, row from mat GROUP BY row;


SELECT ROWMATRIX (label_vector(vec, row)) FROM vecs;
```

ROWMATRIX function aggregates a bunch of vectors as rows to form a matrix

Implementation

- Implemented in java on top of SimSQL
- Incremental not Revolutionary
- A small set of changes

Normalizing

CREATE TABLE vecs (vec VECTOR[ ]);

SELECT label.id, get_scalar(vecs.vec ,

label.id) FROM vecs, label

Distributed Matrices

- Should Individual matrices stored in RDBMS be allowed to be large enough to exceed the size of RAM available on one machine.
- Vectors/Matrices are stored as attributes in tuples.
- What if one has a matrix that is too large to fit in RAM of an individual machine?
- A large dense matrix with 100,000 rows and 100,000 cols and requiring nearly a terabyte of data can be stored as 100 tuples in the table

```
bigMatrix (tileRow INTEGER, tileCol INTEGER,
     mat MATRIX[10000][10000])
```

Algebraic Operations

- Basic operations are implemented directly in java on top of their in memory representation
- Basic operations include extracting the diagonal of a matrix, scalar/Matrix multiplication
- For complex operations like Matrix/Matrix Multiplication and Matrix Inverse the data is transformed into C++ objects and BLAS implementations are used.

Balancing Distributed Computations

- Common way data is partitioned across machines is Hash partitioning
- Hash based partitioning is implicitly relies on the assumption that the number of data items is large
- The number of objects is ideally not large here. Therefore Hashing is ineffective.
- Why should number of objects be small in the first place?

Balancing Distributed Computations

- Consider multiplying two 10^5 * 10^5 matrices. Partitioning the matrices into 1000 * 1000 blocks results in 10^4 different blocks.
- This join will output 10^4 * 10^2 output blocks or 10^6 * 8 MB = 8 TB of data has to be shuffled.

- If the block size is 10^4 * 10^4 then this would result in only 10^2 * 10 output blocks.
- This is less than 1TB of data to shuffle.

Optimization in SimSQL

Consider the following Query

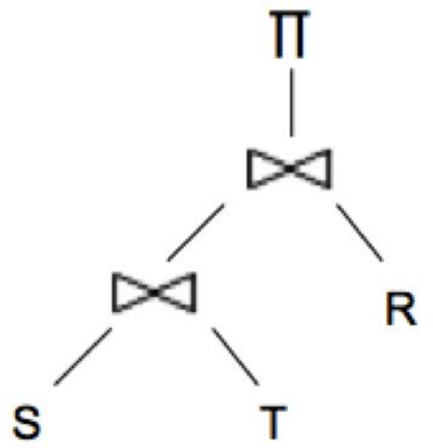
SELECT matrix_multiply (r_matrix, s_matrix) **FROM** R, S, T

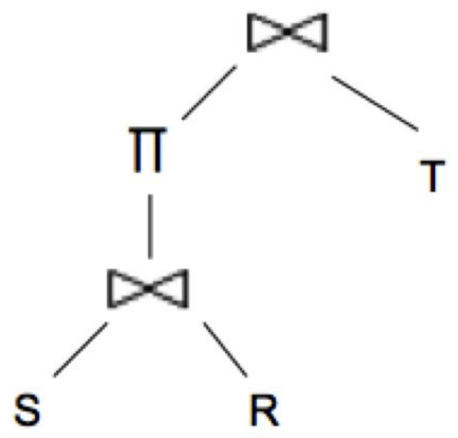
WHERE r_rid = t_rid **AND** s_sid = t_sid

R (r_rid **INTEGER**, r_matrix **MATRIX**[10][100000])

S(s_sid **INTEGER**, s_matrix **MATRIX**[100000][100])

T (t_rid **INTEGER**, t_sid **INTEGER**)





TypeSignatures

- Type signature for any function, that includes vectors and matrices is *templated.*
  - **diag**(**MATRIX**[a][a]) -> **VECTOR**[a]
  - **matrix_multiply**(**MATRIX**[a][b], **MATRIX**[b][c]) -> **MATRIX**[a][c]
- For the query "**SELECT matrix_multiply**(u_matrix, v_matrix) **FROM** U, V" and schema " U (u_matrix **MATRIX**[1000][100]), V (v_matrix **MATRIX**[100][10000])", size will be estimated as

  1000 * 10000 * 8 bytes ~ 80 MB

- When dimension of a matrix is unknown, SimSQL estimates the dimension using the stats collected when materialized views are created and data is loaded.

References

- https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Varsha-Ganesh-Spring-2019.pdf
- https://ieeexplore.ieee.org/abstract/document/8340060

Thank You!