Tensor Relational Algebra for Distributed Machine Learning System Design

Binhang Yuan, Dimitrije Jankov, Jia Zou,
Yuxin Tang, Daniel Bourgeois, Chris Jermaine

Current ML System

Systems like TensorFlow and PyTorch include:

- A front-end:
  - An API for specifying the model;
  - An automatic differentiation engine.
- A back-end:
  - A compute engine responsibly for (parallel/distributed) execution of SGD optimization;
  - Common parallel strategies include: *data parallelism*; *pipeline parallelism* and *operation partitioning*.

What is the Problem?

- Current ML runtime is optimized for particular device(s) and computation paradigms, without careful consideration for distribution.
- Database systems are very successful for declarative parallel/distributed processing.
- However, database is built on set theory which is not appropriate for ML computation.
- Our goal is to bridge this gap!

Run it in a Relational Style

- Encode each computation to a group of relational operations, e.g., join, selection, and aggregation;
- Encode the whole computation flow graph into a query plan;
- Let the system automate the optimization in a distributed runtime.

Matrix Multiplication Example

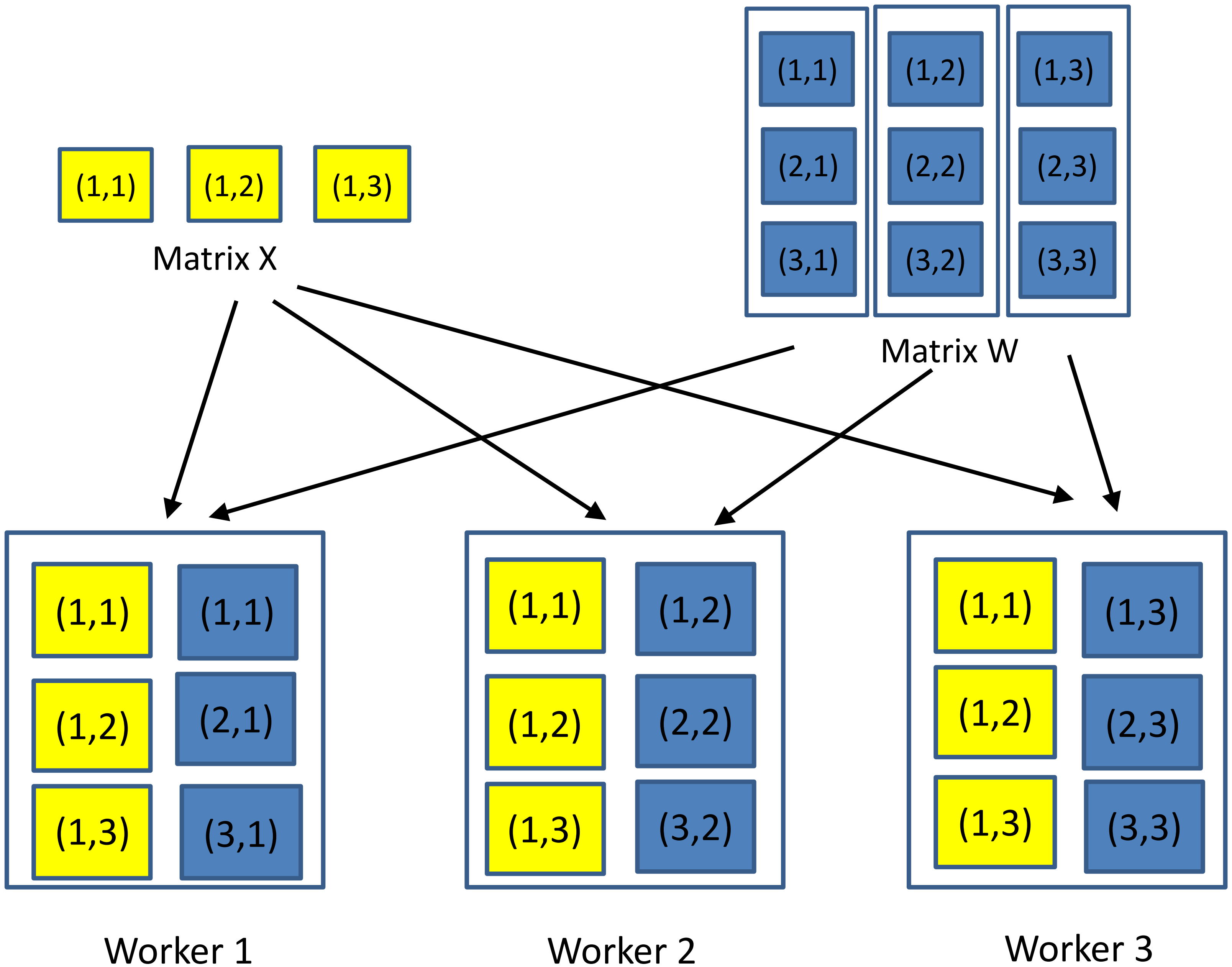
Let us take matrix multiply in FC as an example:

Assume two matrices are stored as blocks in two relations;

How the System Executes the plan?

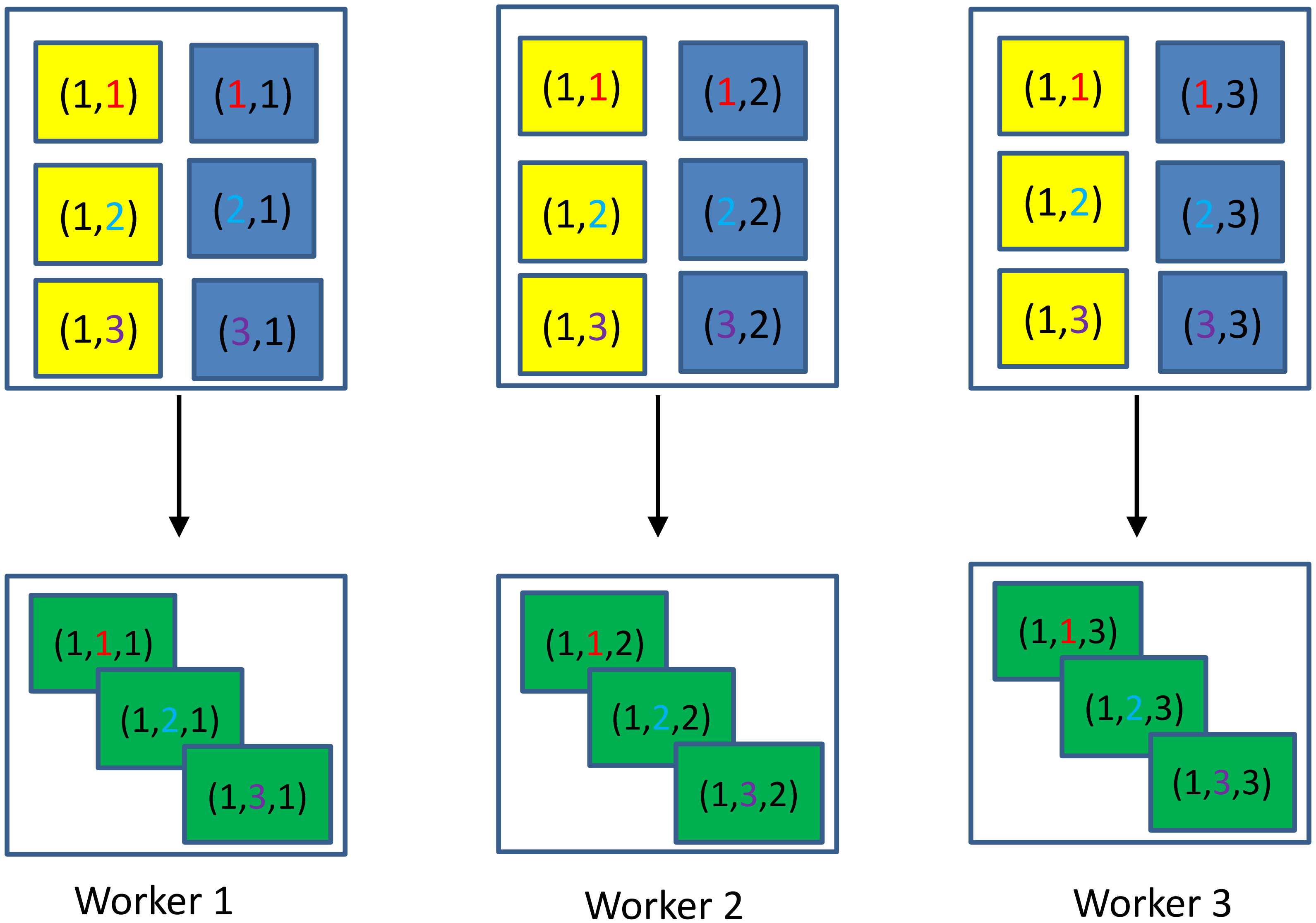
Given 3 worker nodes;

The system can determine to broadcast **X** and partition **W**:



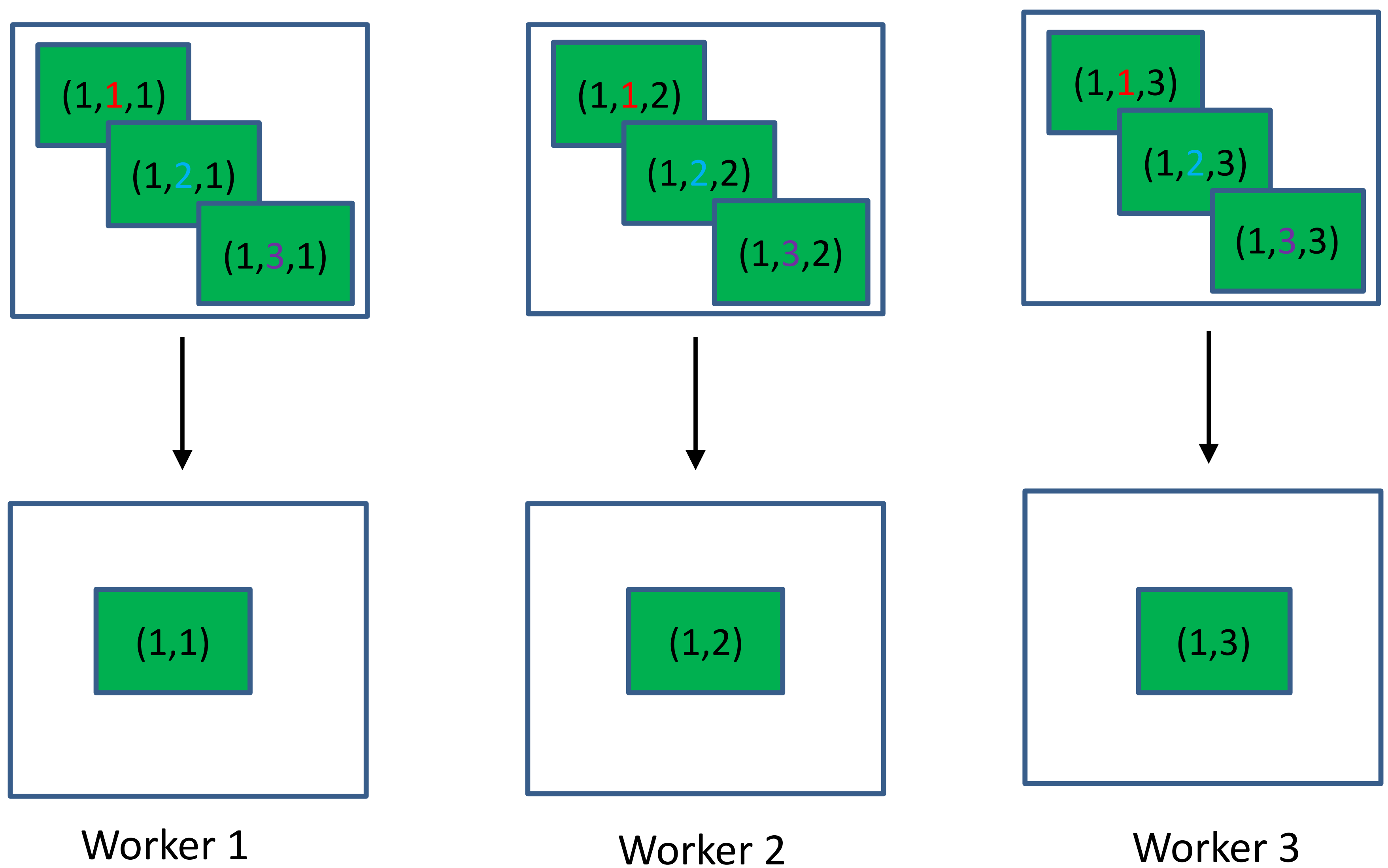
How the System Executes the plan?

The system will do local join and apply multiplication on blocks:



How the System Executes the plan?

The system will do aggregation: local aggregation and apply addition on blocks:



What Should Be the Right Abstractions?

- Answer three fundamental questions:
  - *What should be the right formalization for express distributed ML computations?*

  ➡ **Tensor Relational Algebra (TRA).**

  - What is the right abstraction to implement such formalization, especially for a distributed runtime?

  ➡ **Implementation Abstraction (IA).**

  - What are the unique optimizations we can adopt for this implementation abstraction?

  ➡ **A rule based relational optimizer.**

Tensor Relational Algebra (TRA)

- A **tensor relation** includes tuples of <key, array>.
- Support UDF-style operations:
  Aggregate Σ; *Join* $\bowtie$; *ReKey*; *Filter* σ; *Transform* λ; *Tile; Concat*.
- This abstraction is expressive enough to represent any tensor manipulation allowed by Einstein Notation.

TRA — Operations

- **Join**: $\bowtie$: $\left((\mathbb{Z}^*)^g \times (\mathbb{Z}^*)^g \times \left(T^{(r_l,\mathrm{b}_l)} \times T^{(r_r,\mathrm{b}_r)} \to T^{(r_o,\mathrm{b}_o)}\right)\right) \to \left(R^{(k,r,\mathrm{b})} \to R^{(g,r,\mathrm{b})}\right)$
- **Aggregation**: $\Sigma : \left((\mathbb{Z}^*)^g \times \left(T^{(r,\mathrm{b})} \times T^{(r,\mathrm{b})} \to T^{(r,\mathrm{b})}\right)\right) \to \left(R^{(k,r,\mathrm{b})} \to R^{(g,r,\mathrm{b})}\right)$
- **Rekey**: $\text{ReKey} : \left((\mathbb{Z}^*)^{k_i} \to (\mathbb{Z}^*)^{k_o}\right) \to \left(R^{(k_i,r,\mathrm{b})} \to R^{(k_o,r,\mathrm{b})}\right)$
- **Transform**: $\lambda : \left(T^{(r_i,\mathrm{b}_i)} \to T^{(r_o,\mathrm{b}_o)}\right) \to \left(R^{(k,r_i,\mathrm{b}_i)} \to R^{(k,r_o,\mathrm{b}_o)}\right)$
- **Filter**: $\sigma : \left((\mathbb{Z}^*)^k \to \{\text{true}, \text{false}\}\right) \to \left(R^{(k,r,\mathrm{b})} \to R^{(k,r,\mathrm{b})}\right)$
- **Tile**: $\text{Tile} : (\mathbb{Z}^* \times \mathbb{Z}^*) \to \left(R^{(k,r,\mathrm{b})} \to R^{(k+1,r,\mathrm{b}')}\right)$
- **Concat**: $\text{Concat} : (\mathbb{Z}^* \times \mathbb{Z}^*) \to \left(R^{(k,r,\mathrm{b})} \to R^{(k-1,r,\mathrm{b}')}\right)$

Implementation Abstraction (IA)

- TRA can formalize ML operations, but is not sufficient to specify different distributed implementations.
- IA is to bridge this gap.
- A physical tensor relation includes tuples of $\langle$key, array, site$\rangle$:
- Extend the TRA to operations preferred in Distributed runtimes.

IA Operations

- Assign tuple to site:
  - **Broadcast**: BROADCAST : $\mathscr{R}^{(k,r,\mathrm{b},s)} \to \mathscr{R}^{(k,r,\mathrm{b},s)}$, after this we will have $\mathrm{ALL}\,(R) = \mathrm{true}$
  - **Shuffle**: SHUFFLE : $2^{\{1\ldots k\}} \to \left(\mathscr{R}^{(k,r,\mathrm{b},s)} \to \mathscr{R}^{(k,r,\mathrm{b},s)}\right)$, after this we will have $\mathrm{PART}_D\,(R) = \mathrm{true}$
- local operations:
  - **Local Join**: $\bowtie^L$: $\left((\mathbb{Z}*)^g \times (\mathbb{Z}*)^g \times \left(T^{(r_l,\mathrm{b}_l)} \times T^{(r_r,\mathrm{b}_r)} \to T^{(r_o,\mathrm{b}_o)}\right)\right) \to \left(\mathscr{R}^{(k_l,r_l,\mathrm{b}_l,s)} \times \mathscr{R}^{(k_r,r_r,\mathrm{b}_r,s)} \to \mathscr{R}^{(k_l+k_r-g,r_o,\mathrm{b}_o,s)}\right)$
  - **Local Aggregate**: Σ^L : $\left((\mathbb{Z}*)^k \times \left(T^{(r,\mathrm{b})} \times T^{(r,\mathrm{b})} \to T^{(r,\mathrm{b})}\right)\right) \to \left(\mathscr{R}^{(k_l,r_l,\mathrm{b}_l,s)} \times \mathscr{R}^{(k_r,r_r,\mathrm{b}_r,s)} \to \mathscr{R}^{(k_l+k_r-g,r_o,\mathrm{b}_o,s)}\right)$
  - **Local filter**: σ^L : $\left((\mathbb{Z}*)^g \to \{\mathrm{true}, \mathrm{false}\}\right) \to \left(\mathscr{R}^{(k,r,\mathrm{b},s)} \to \mathscr{R}^{(k,r,\mathrm{b},s)}\right)$
  - **Local map**: λ^L : $\left(\left((\mathbb{Z}*)^{k_i} \to \left((\mathbb{Z}*)^{k_o}\right)^m\right) \times \left(T^{(r_i,\mathrm{b}_i)} \to \left(T^{(r_o,\mathrm{b}_o)}\right)^m\right)\right) \to \left(\mathscr{R}^{(k_i,r_i,\mathrm{b}_i,s)} \to \mathscr{R}^{(k_o,r_o,\mathrm{b}_o,s)}\right)$

Optimization — Equivalence

- Equivalence of IA expressions: Equivalent input physical tensor relations leads to equivalent output physical tensor relations.
- Simple equivalence rules:
  - Kernel function composition;
  - Equivalent partitioning.
- Domain specific equivalence rules:
  - E.g., different IA representations for distributed matrix multiplication.

Optimization — Equivalence

- Domain specific equivalence rules for matrix multiplication.
- Target TRA: $\Sigma_{\left(\langle 0,2\rangle,\text{matAdd}\right)}\left(\bowtie_{\left(\langle 1\rangle,\langle 0\rangle,\text{matMul}\right)}(\mathrm{R}_X, \mathrm{R}_Y)\right)$
- IA can provide implementations for:
  - Broadcast based matrix multiplication (BMM).
  - Cross product based matrix multiplication (CPMM).
  - Replication based matrix multiplication (RMM).

Optimization — Cost Model

- A cost model to consider network traffic.
- In contrast to the classic query optimization (where estimating selectivity is usually difficult) — the continuity constraints make this free.
- Give input tensor relation(s), the output tensor relation's frontier (for the key) and bound (for the array) can be estimated.
- For a tensor relation R, suppose f is the estimated floating numbers in R, s is the number of sites, then:
  - Broadcast has a cost of $f \times s$;
  - Shuffle has a cost of s.

Experiments

- **Aims:**
  - *Can the proposed optimization generate efficient plan(s) for a particular task over a particular dataset?*
  - *Can the end-to-end performance be competitive to the STOA HPC or ML systems?*
- Benchmarks:
  - Distributed matrix multiplication (MM).
  - Nearest Neighbor Search (NNS) in Riemannian metric space.
  - **Feed Forward Neural Network SGD iteration (FFNN).**

Experimental Task — FFNN

- 2-layer FFNN for classification:
  - MiniBatch $\left(\mathbf{X} \in \mathbb{R}^{N \times D}, \quad \mathbf{Y} \in \mathbb{R}^{N \times L}\right)$,
  - Weight matrix of layer 1 $\mathbf{W}_1 \in \mathbb{R}^{D \times H}$,
  - Weight matrix of layer 2 $\mathbf{W}_2 \in \mathbb{R}^{H \times L}$;
- Settings:
  - Google speech recognition:
    $N = 10^4, \quad D = 1600, \quad L = 10, \quad H = 10^5,\ 1.5 \times 10^5,\ 2 \times 10^5;$
  - Amazon 14k XML:
    $N = 10^3, \quad D = 597540, \quad L = 14588, \quad H = 10^3,\ 3 \times 10^3,\ 5 \times 10^3;$
- IA implementation:
  - TRA-DP, TRA-MP;
- Compared with:
  - TensorFlow(TF)/PyTorch data parallel implementation;
  - A careful HPC implementation based on ScaLapack;
  - Dask — a popular distributed python analytic package.

Experimental Results — FFNN1

2-layer FFNN for Google Speech

Cluster	CPU			GPU		
Nodes	2	5	10	2	5	10
100k Neurons						
PyTorch-DP	11.16	6.15	4.75	0.99	1.19	1.27
TF-DP	11.93	7.32	5.51	0.87	1.13	1.17
ScaLAPACK	8.32	4.97	2.79	NA	NA	NA
Dask	62.57	56.57	49.63	NA	NA	NA
TRA-DP	11.62	6.51	5.20	1.49	1.59	1.63
TRA-MP	26.56	28.71	29.09	7.01	11.56	Fail
200k Neurons						
PyTorch-DP	17.25	11.94	9.30	Fail	2.09	2.42
TF-DP	21.36	13.21	11.21	1.52	2.12	2.46
ScaLAPACK	17.18	10.05	5.06	NA	NA	NA
Dask	136.66	112.72	104.01	NA	NA	NA
TRA-DP	17.89	12.51	9.69	1.49	1.59	1.63
TRA-MP	37.82	54.23	59.84	Fail	Fail	Fail

Predicted Cost

	TRA-DP	TRA-MP
100k Neurons	**9.7×10^{8}**	1.0×10^{10}
200k Neurons	**1.9×10^{9}**	2.0×10^{10}

Experimental Results — FFNN2

2-layer FFNN for Amazon-14k XML (seconds)

Cluster	CPU			GPU		
Nodes	2	5	10	2	5	10
1k Neurons						
PyTorch-DP	9.74	10.29	10.34	2.67	3.76	4.20
TF-DP	Fail	Fail	Fail	Fail	Fail	Fail
ScaLAPCK	8.16	6.65	2.47	NA	NA	NA
Dask	45.40	42.15	29.34	NA	NA	NA
TRA-DP	12.50	14.29	15.68	4.67	4.69	4.73
TRA-MP	**3.86**	**2.79**	**1.7**	**0.4**	**0.37**	**0.35**
5k Neurons						
PyTorch-DP	34.05	46.53	50.17	Fail	Fail	Fail
TF-DP	Fail	Fail	Fail	Fail	Fail	Fail
ScaLAPACK	23.21	11.65	8.33	NA	NA	NA
Dask	246.56	143.86	127.26	NA	NA	NA
TRA-DP	44.12	68.54	75.15	Fail	Fail	Fail
TRA-MP	**18.59**	**8.07**	**5.57**	Fail	**0.59**	**0.48**

Predicted Cost

	TRA-DP	TRA-MP
1k Neurons	3.7×10^{9}	$\mathbf{1.0\times10^{7}}$
5k Neurons	1.8×10^{10}	$\mathbf{5.0\times10^{7}}$

Thank you!