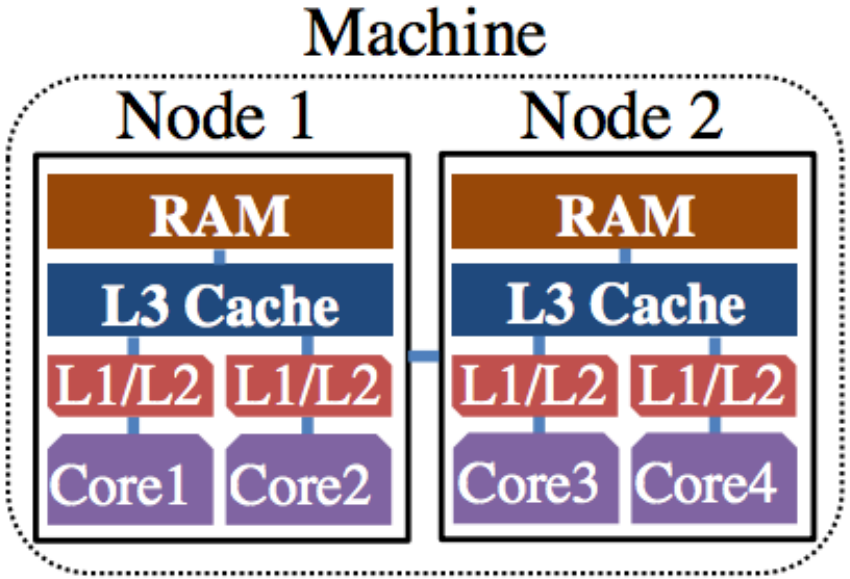
DIMMWITTED: A STUDY OF MAIN-MEMORY STATISTICAL ANALYTICS

Shivaram Venkataraman

MOTIVATION

How to best use main memory ?

Memory Bandwidth:
~60 GB/s r3.8xlarge on EC2

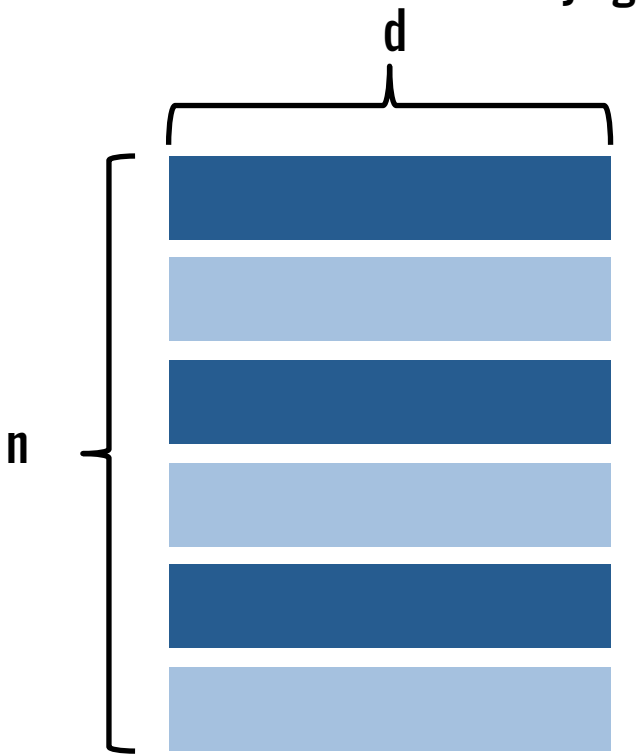


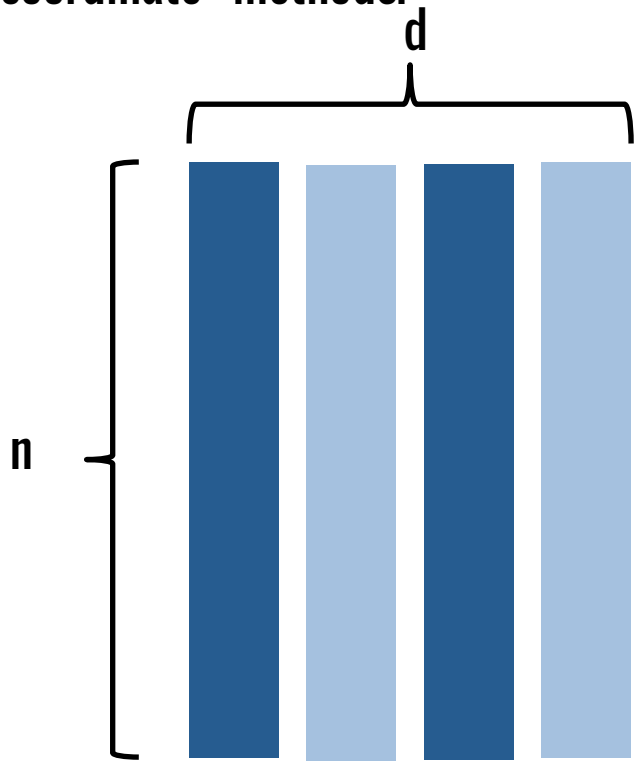
DESIGN SPACE

- Access method
  - Row vs. Column
  - Density
- Replication
  - Data
  - Model

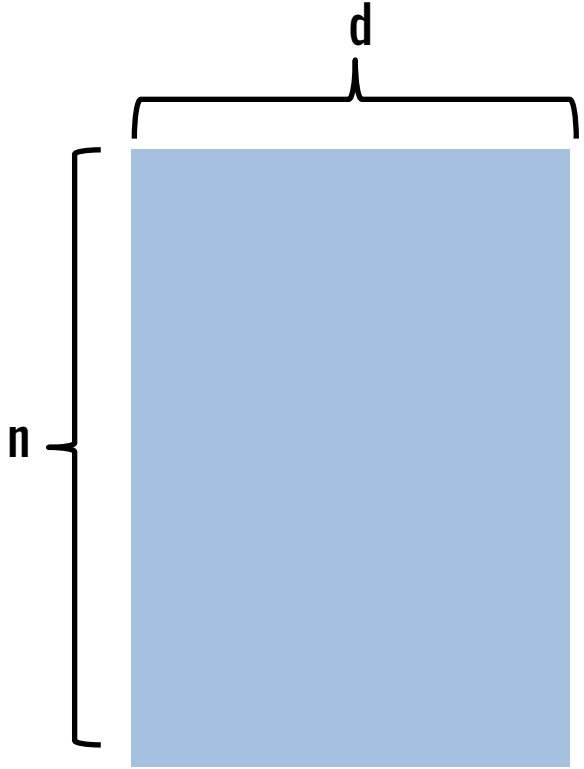
ITERATIVE ALGORITHMS: ACCESS METHOD

Sample rows vs. columns
Broadly “gradient” vs “coordinate” methods.





DATA DENSITY: DENSE VS. SPARSE



Dense Linear Algebra

- More FLOPs / CPU intensive
- e.g., Matrix vector multiply: O(n * d)

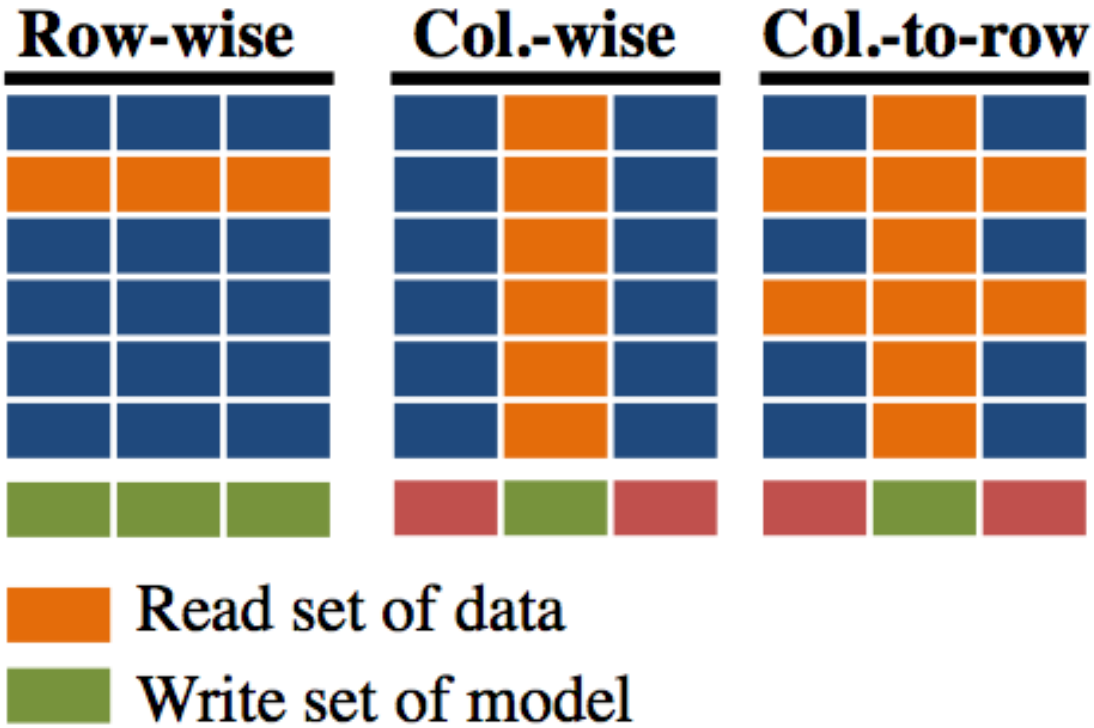
Sparse Linear Algebra

- Lesser FLOPs / communication intensive
- e.g., Matrix vector multiply: O(nnz * d)

DIMM WITTED: ACCESS METHODS

Data

Model



REPLICATION

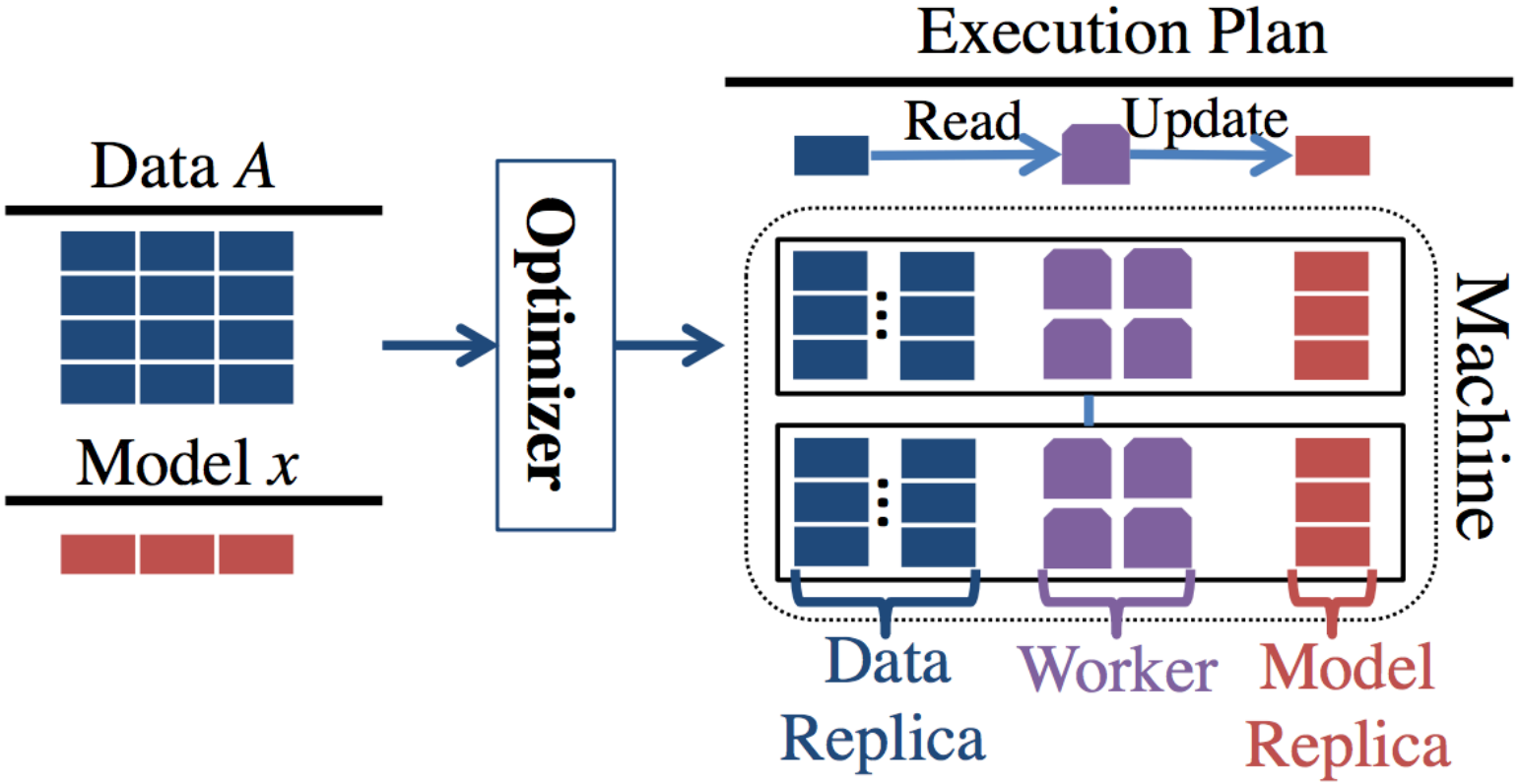
Model

- Replica per core ? Similar to Spark, shared nothing
- Replica per machine ? Shared memory
- Hybrid: Replica per NUMA node

Data

- Partition per core ? Similar to shared nothing
- Replicate data per NUMA node?

DIMM WITTED


Execution Plan
Data A
Model x
Optimizer
Read
Update
Machine
Data Replica
Worker
Model Replica

OPTIMIZER

Inputs

- $f_{row}, f_{col}, f_{ctr}$
- data $A \in R^{N \times d}$
- Initial model vector

Output

- Execution plan for each CPU
  - subset of data
  - model replica
  - access method to use

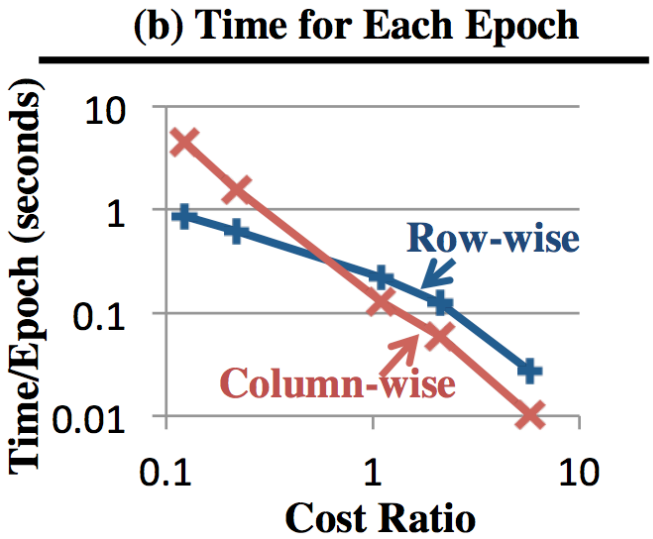
ACCESS METHOD

Algorithm	Read	Write (Dense)	Write (Sparse)
Row-wise	$\sum n_i$	dN	$\sum n_i$
Column-wise	$\sum n_i$	d	
Column-to-row	$\sum n_i^2$		

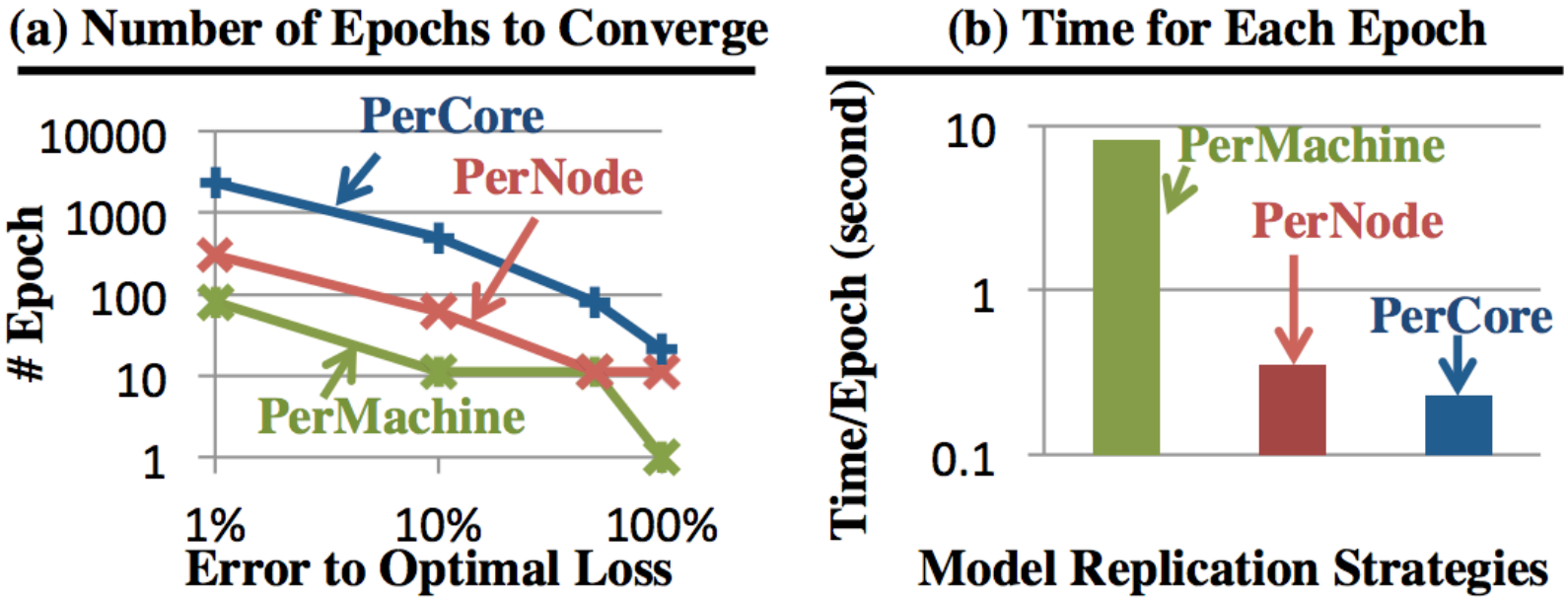
- Cost Ratio: how much more expensive writes are

- Row-wise is more efficient when writes are cheap

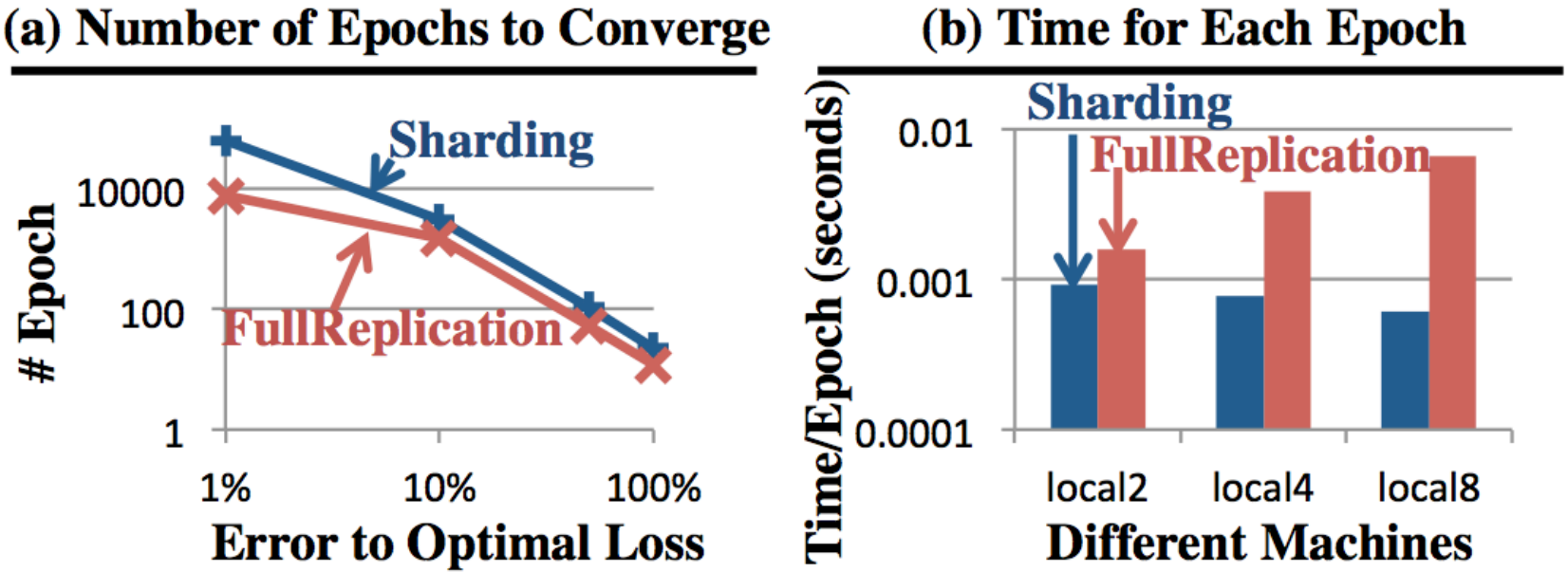
- Column-to-row becomes more efficient at some point



MODEL REPLICATION



DATA REPLICATION



TAKEAWAYS

- Data access patterns matters but changes based on problem
- Model / data replication design space
- “Optimizer” for ML

QUESTIONS / DISCUSSION ?