High-Performance Distributed Machine Learning in Heterogeneous Compute Environments

Celestine Dünner

Martin Jaggi (EPFL)
Thomas Parnell, Kubilay Atasu, Manolis Sifalakis, Dimitrios Sarigiannis, Haris Pozidis (IBM Research)

Motivation of Our Work

Distributed Training of Large-Scale Linear Models

- ✓ fast training
- ✓ interpretable models
- ✓ training on large-scale datasets

IBM

Motivation of Our Work

in practice

Distributed Training of Large-Scale Linear Models

Choose an Algorithm

Choose an Implementation

Choose an Infrastructure

? How does the infrastructure and the implementation impact the performance of the algorithm?

IBM

Motivation of Our Work

in practice

Distributed Training of Large-Scale Linear Models

Choose an Algorithm

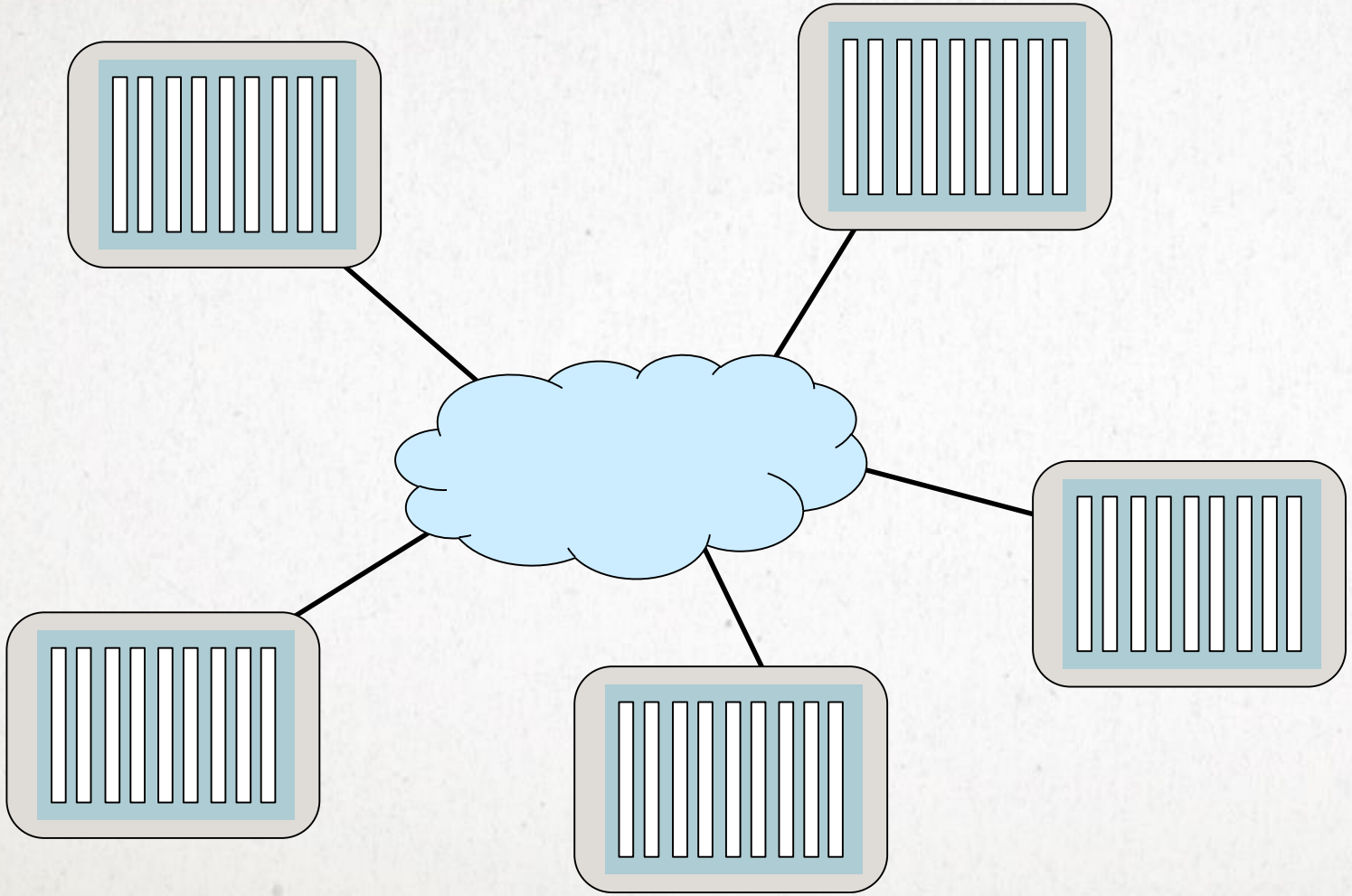
Choose an Implementation

Choose an Infrastructure

? How can algorithms be optimized and implemented to achieve optimal performance on a given system?

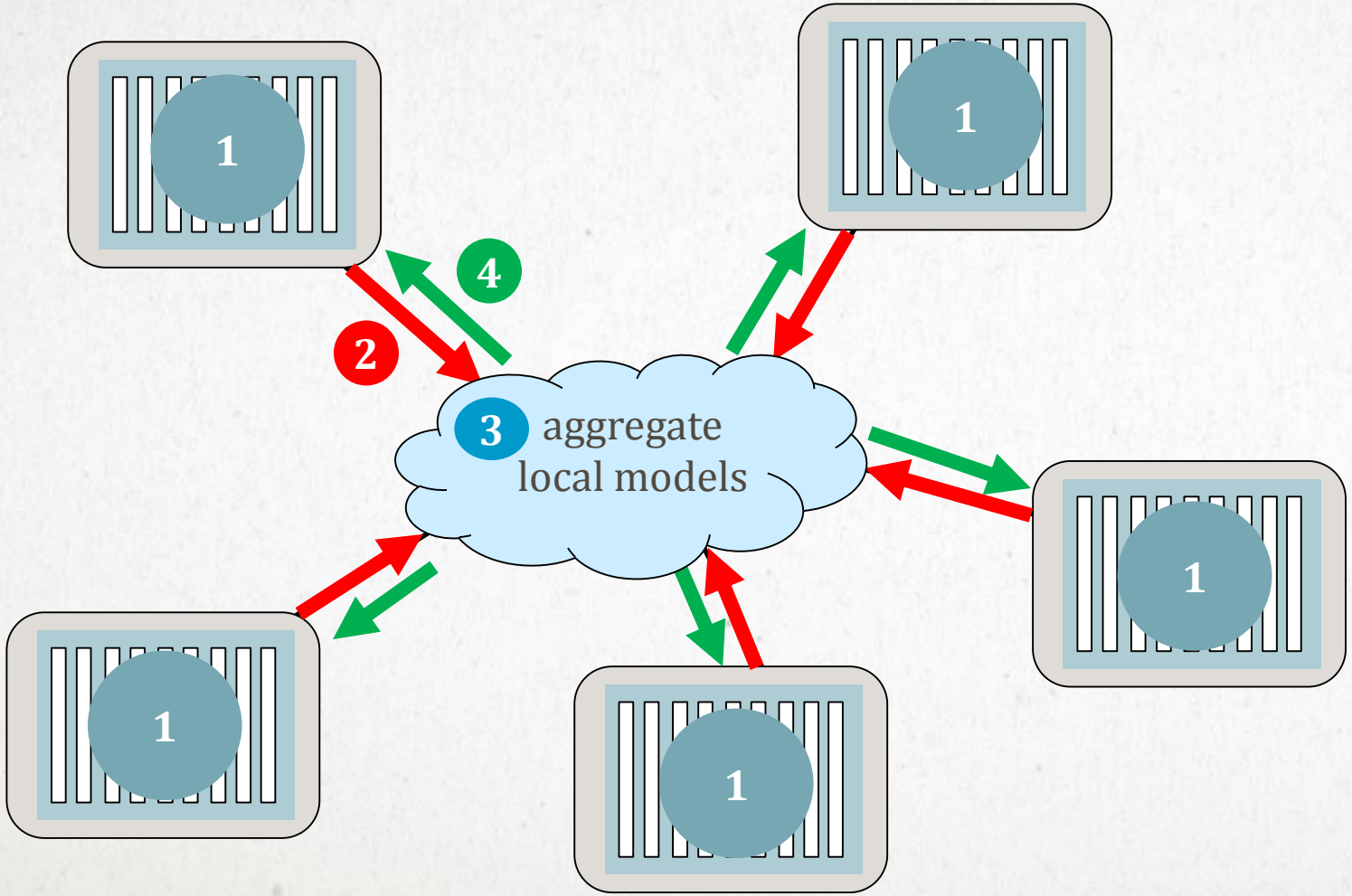
IBM

Algorithmic Challenge of Distributed Learning



$$\min_{\mathbf{w}} \quad f(A^\top \boldsymbol{w}) + g(\boldsymbol{w})$$

Algorithmic Challenge of Distributed Learning



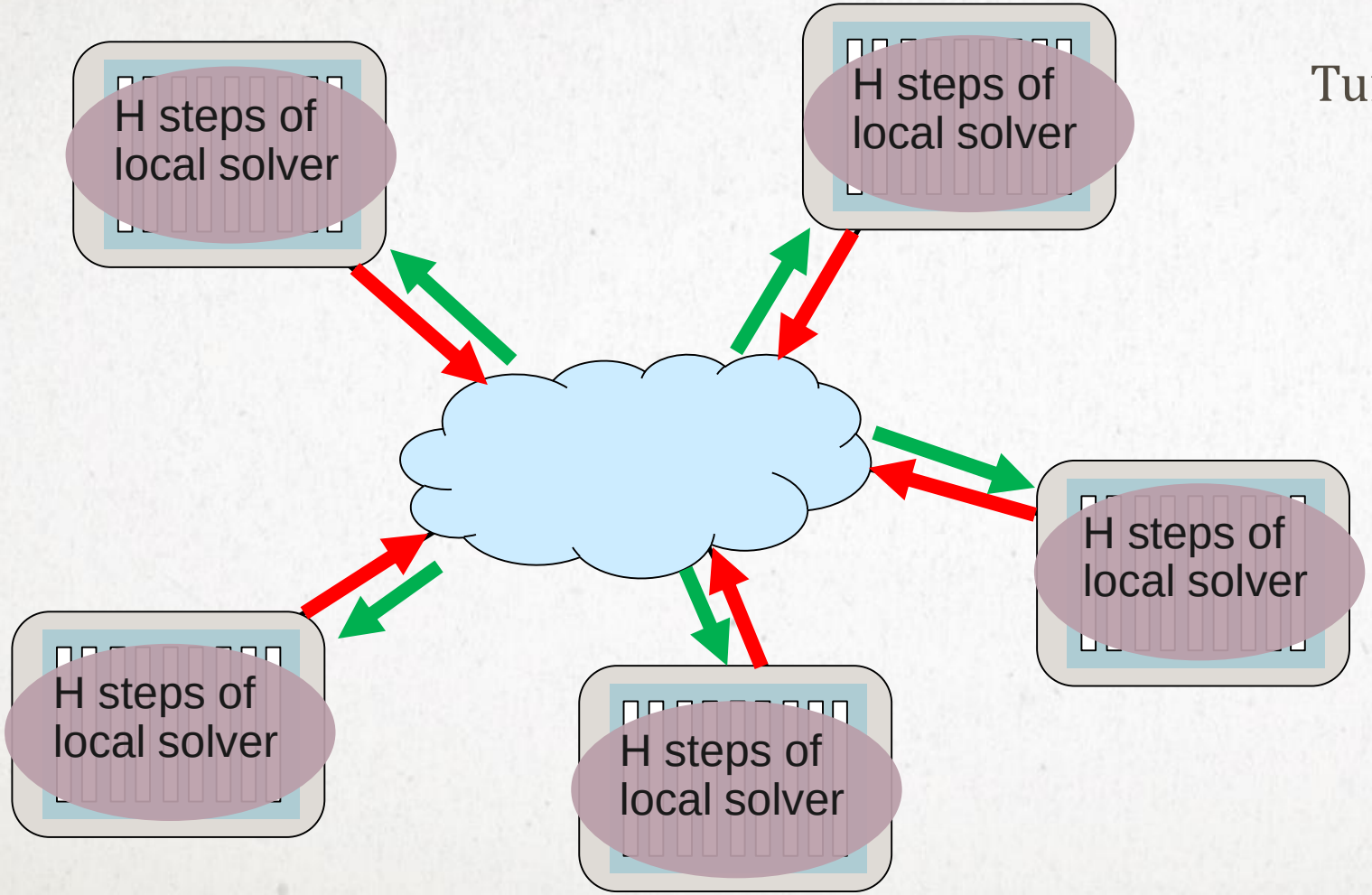
- The more frequently you exchange information the faster your model converges

Trade-off

- Communication over the network can be very expensive

$$\min_{\mathbf{w}} \quad f(A^\top \boldsymbol{w}) + g(\boldsymbol{w})$$

CoCoA Framework [*SMITH, JAGGI, TAKAC, MA, FORTE, HOFMANN, JORDAN*, 2013-2015]



Tunable hyper-parameter **H**

H* depends on:

- system
- Implementation/ framework

$$\min_{\mathbf{w}} \quad f(A^\top \boldsymbol{w}) + g(\boldsymbol{w})$$

Implementation: Frameworks for Distributed Computing

* http://spark.apache.org/

Open Source Cloud Computing Framework

- Easy-to-use
- Powerful APIs : *Python, Scala, Java, R*
- Poorly understood overheads

MPI

High-Performance Computing Framework

- Requires advanced system knowledge
- C++
- Good performance

Designed for different purposes ⇨ different characteristics

Different implementations of CoCoA

webspam dataset
8 Spark workers

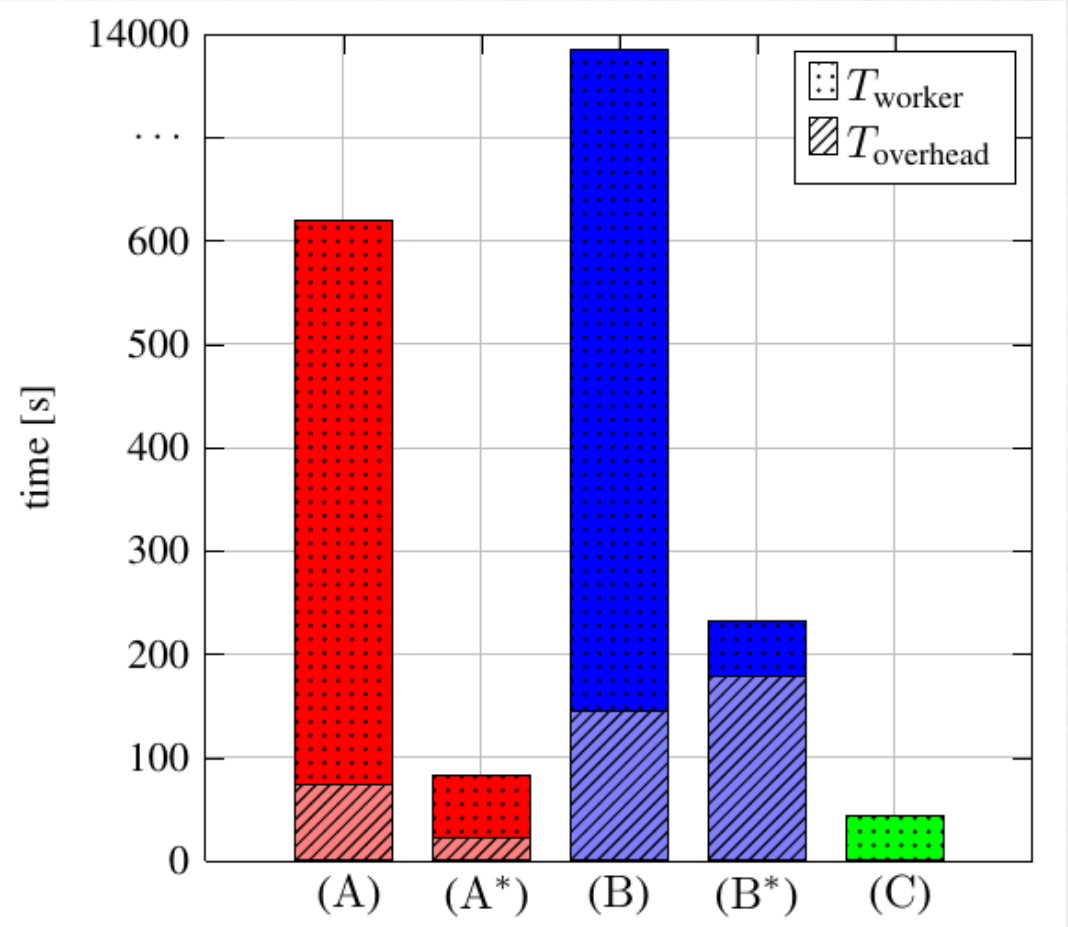
- (A) Spark Reference Implementation*
- (B) pySpark Implementation
- (C) MPI Implementation

Offload local solver to C++

- (A*) Spark+C
- (B*) pySpark+C

➡ (A*),(B*),(C) execute identical C++ code

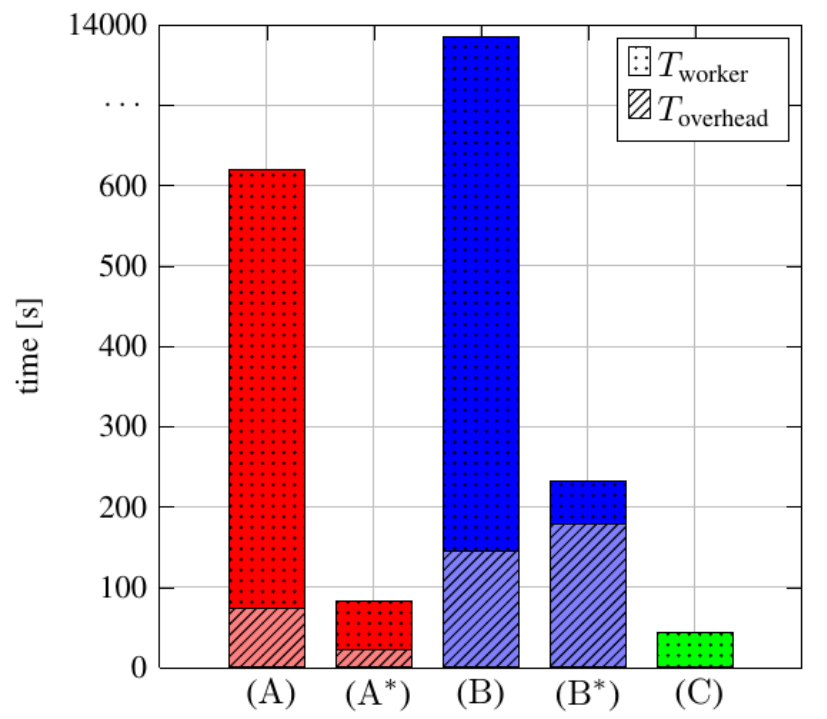
*https://github.com/gingsmith/cocoa

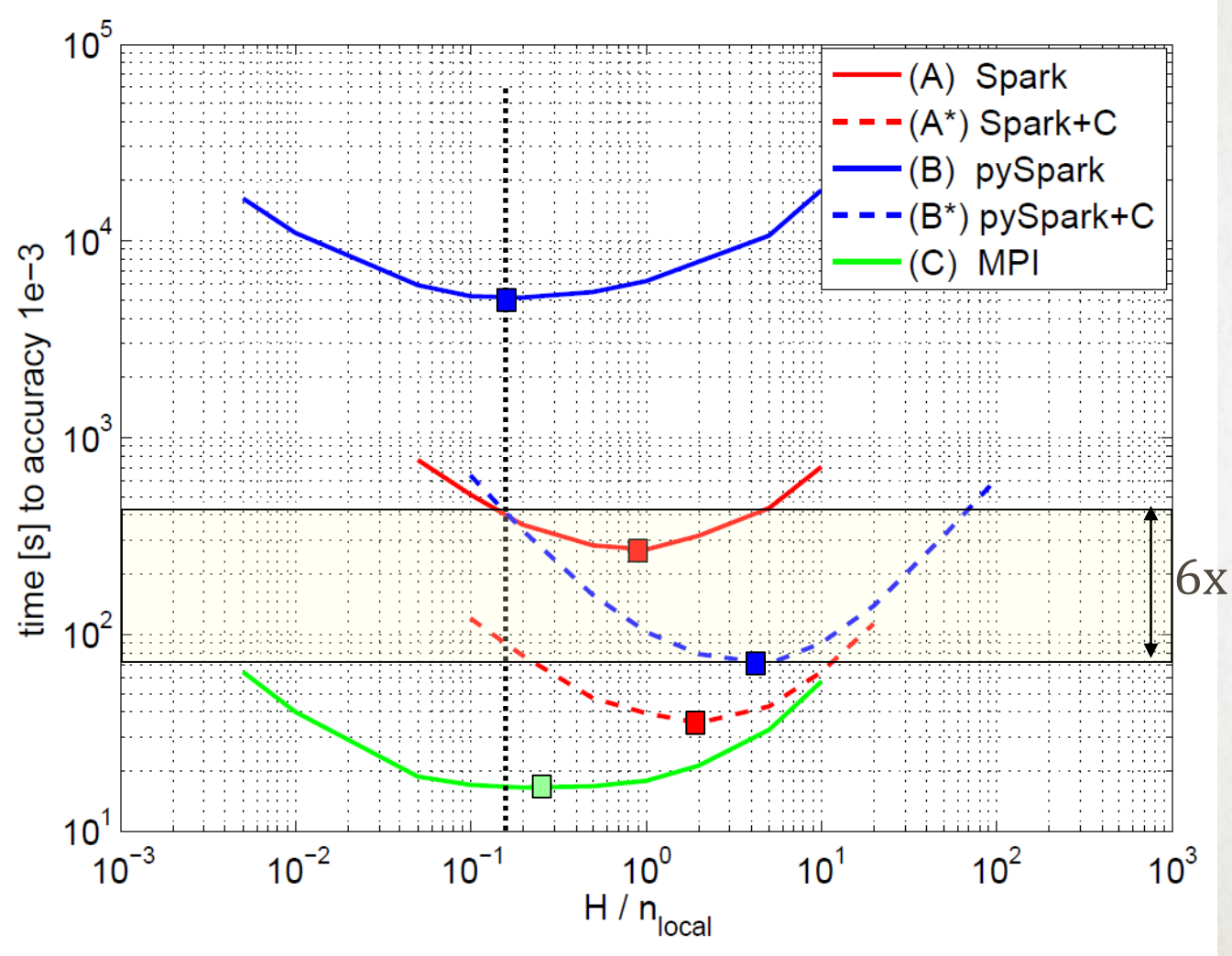


100 iterations of CoCoA for H fixed

Communication-Computation Tradeoff

webspam dataset
8 Spark workers





Understanding characteristics of the framework and correctly adapting the algorithm can decide upon orders of magnitude in performance!

to the designer

strive to design flexible algorithms that can be adapted to system characteristics

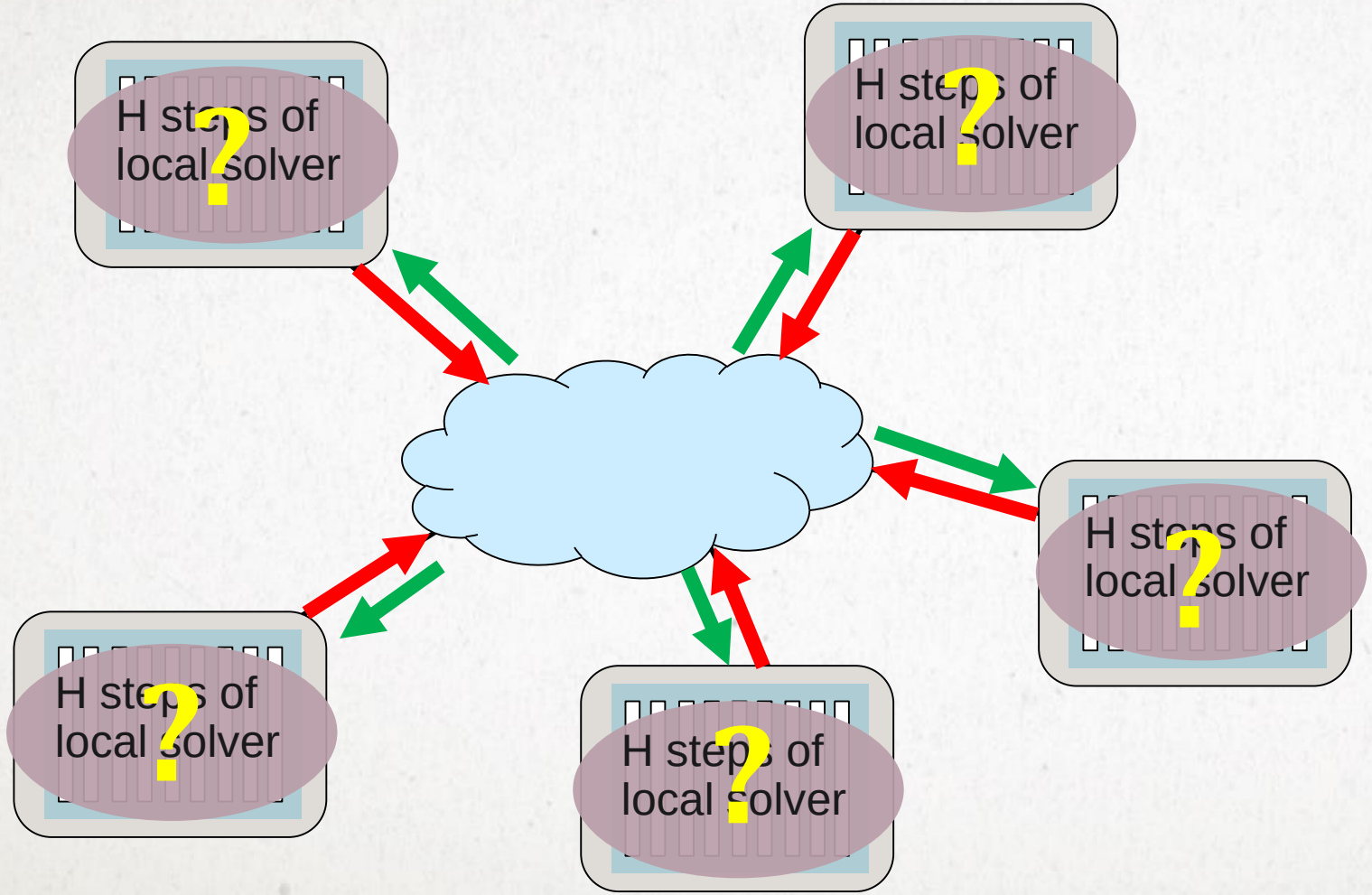
to the user

be aware that machine learning algorithms need to be tuned to achieve good performance

C. Dünner, T. Parnell, K. Atasu, M. Sifalakis, H. Pozidis,
"*Understanding and Optimizing the Performance of Distributed Machine Learning Applications on Apache Spark*",
IEEE International Conference on Big Data, Boston, 2017

IBM

Which local solver should we use?



Stochastic Primal-Dual Coordinate Descent Methods

Algorithm 1 Primal-Dual Coordinate Descent

1: Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
2: Initialize $\mathbf{v}^{(0)} = \mathbf{0} \quad (\mathbf{v} := A^\top \mathbf{w})$
3: **for** $t = 0, 1, 2, \ldots\ldots$ **do**
4: randomly select a coordinate $j \in [n]$
5: find Δw_j to minimize $f(\mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top) + g_j(w_j + \Delta w_j)$
6: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta w_j \mathbf{e}_j$
7: $\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top$
8: **end for**

✔ good convergence properties

✘ sequential

problem: cannot leverage full power of modern CPUs or GPUs

Asynchronous implementations

f: smooth

$$\min_{\boldsymbol{w}} f(A^\top \boldsymbol{w}) + \sum_i g_i(w_i)$$

Ridge Regression, Lasso, Logistic Regression….

$$\min_{\boldsymbol{w}} \frac{1}{2n}\|A^\top \boldsymbol{w} - \boldsymbol{y}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

$$\min_{\boldsymbol{w}} \frac{1}{2n}\|A^\top \boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_1$$

f^*: strongly-convex

$$\min_{\boldsymbol{\alpha}} f^*(\boldsymbol{\alpha}) + \sum_i g_i^*(-A_{:i}^\top \boldsymbol{w})$$

L_2-regularized SVM, Ridge Regression, L_2-regularized Logistic Regression,…

Asynchronous Stochastic Algorithms

$$\min_{\boldsymbol{w}} f(A^\top \boldsymbol{w}) + \sum_i g_i(w_i)$$

Algorithm 1 Primal-Dual Coordinate Descent

Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
Initialize $\mathbf{v}^{(0)} = \mathbf{0}$ $\quad (\mathbf{v} := A^\top \mathbf{w})$
for $t = 0, 1, 2, \ldots\ldots$ **do**
  randomly select a coordinate $j \in [n]$
  find Δw_j to minimize $f(\mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top) + g_j(w_j + \Delta w_j)$
  $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta w_j \mathbf{e}_j$
  $\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top$
end for

Parallelized over cores:
Every core updates a dedicated subset of coordinates

core ℓ

randomly select $j \in \mathcal{I}_\ell$
read $\hat{\mathbf{v}} = \mathbf{v}$
read w_j
find Δw_j
write $(w_j, w_j + \Delta w_j)$
write$(\mathbf{v}, \hat{\mathbf{v}} + \Delta w_j \mathbf{x}_j^\top)$

Write collision on shared vector

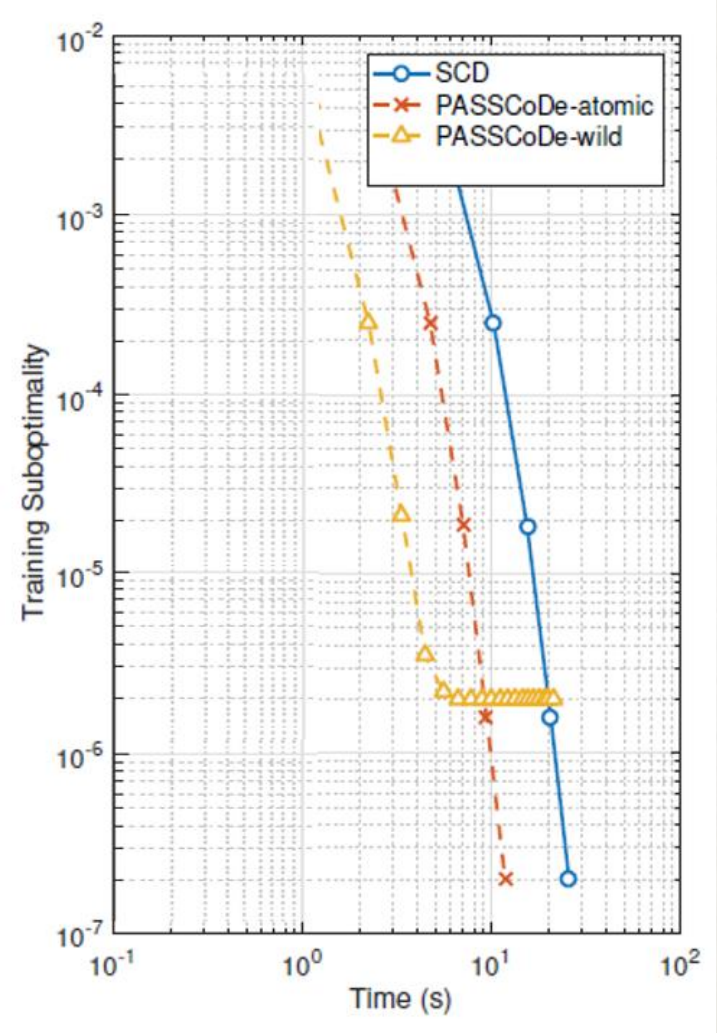
- Recompute shared vector *Liu et al., "AsySCD" (JMLR'15)*
- Memory-locking *K. Tran et al., "Scaling up SDCA, (SIGKDD'15)*
- Live with undefined behavior C.-J. Hsieh et al. "*PASSCoDe:*" (ICML'15)

Asynchronous Stochastic Algorithms

$$\min_{\boldsymbol{w}} f(A^\top \boldsymbol{w}) + \sum_i g_i(w_i)$$

Algorithm 1 Primal-Dual Coordinate Descent

Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
Initialize $\mathbf{v}^{(0)} = \mathbf{0}$ $\quad (\mathbf{v} := A^\top \mathbf{w})$
for $t = 0, 1, 2, \ldots\ldots$ **do**
  randomly select a coordinate $j \in [n]$
  find Δw_j to minimize $f(\mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top) + g_j(w_j + \Delta w_j)$
  $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta w_j \mathbf{e}_j$
  $\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top$
end for

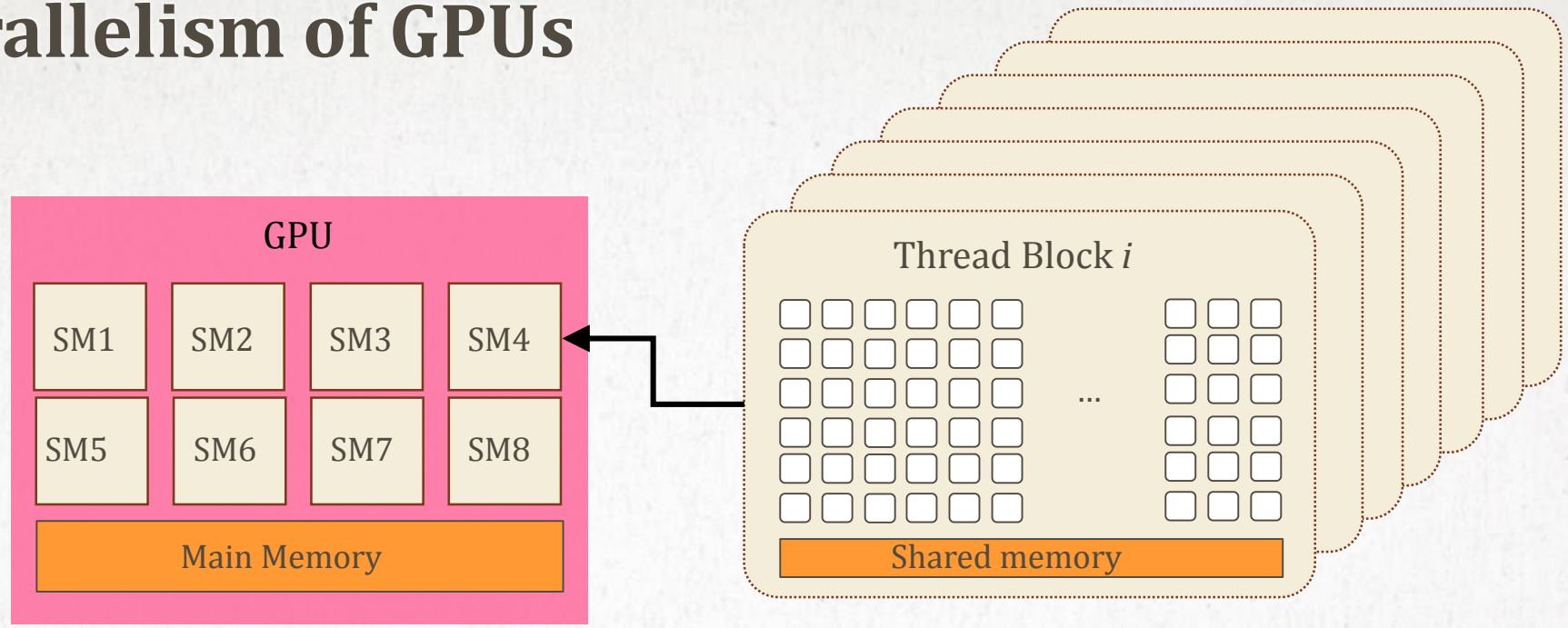


Ridge Regression on webspam

Write collision on shared vector

- Recompute shared vector *Liu et al., "AsySCD" (JMLR'15)*
- Memory-locking *K. Tran et al., "Scaling up SDCA, (SIGKDD'15)*
- Live with undefined behavior C.-J. Hsieh et al. "*PASSCoDe:*" (ICML'15)

2-level Parallelism of GPUs



1st level of parallelism:

- A GPU consists of streaming multiprocessors
- Thread blocks get assigned to multi-processors and are executed asynchronously

2nd level of parallelism

- Each thread block consists of up to 1024 threads
- Threads are grouped into warps (32 threads) which are executed as SMDI operations

GPU Acceleration

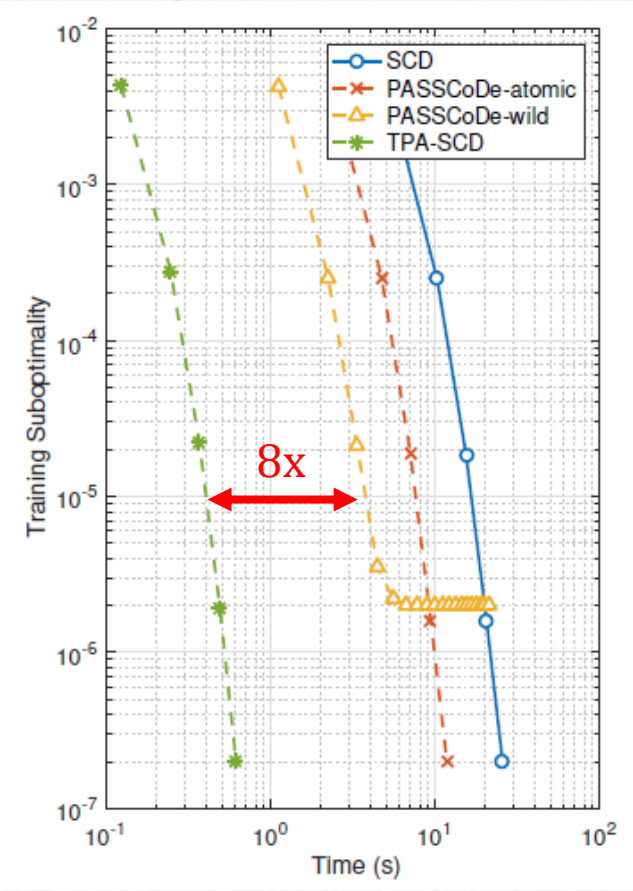
webspam dataset
GPU: GeForce GTX 1080Ti
CPU: 8-core Intel Xeon E5

A **T**wice **P**arallel **A**synchronous **S**tochastic **C**oordinate **D**escent (TPA-SCD) Algorithm

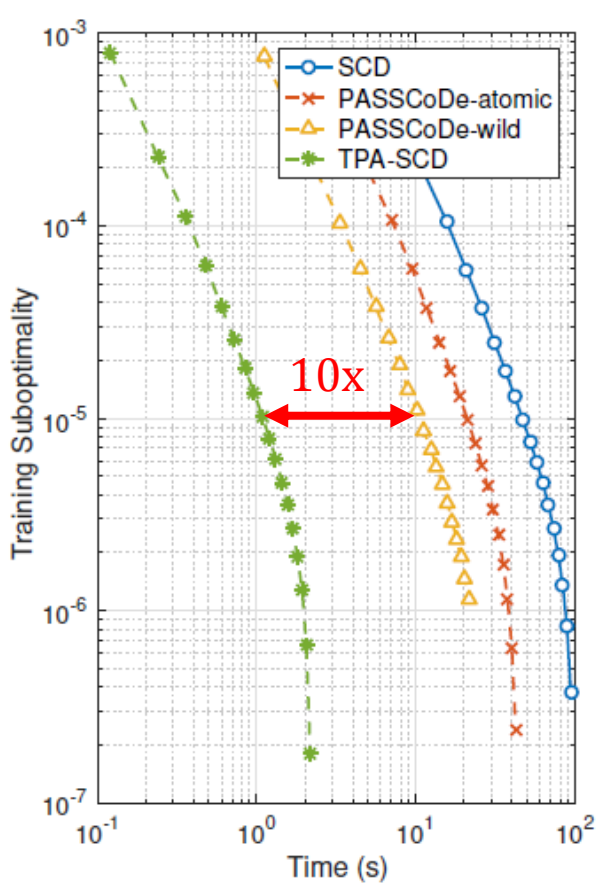
1. thread-blocks are executed in parallel, *asynchronously* updating one coordinate each.

2. Update computation within a thread block is interleaved to ensure memory locality within warp and local memory is used to accumulate partial sums

3. *Atomic add* functionality of modern GPUs is used to update shared vector



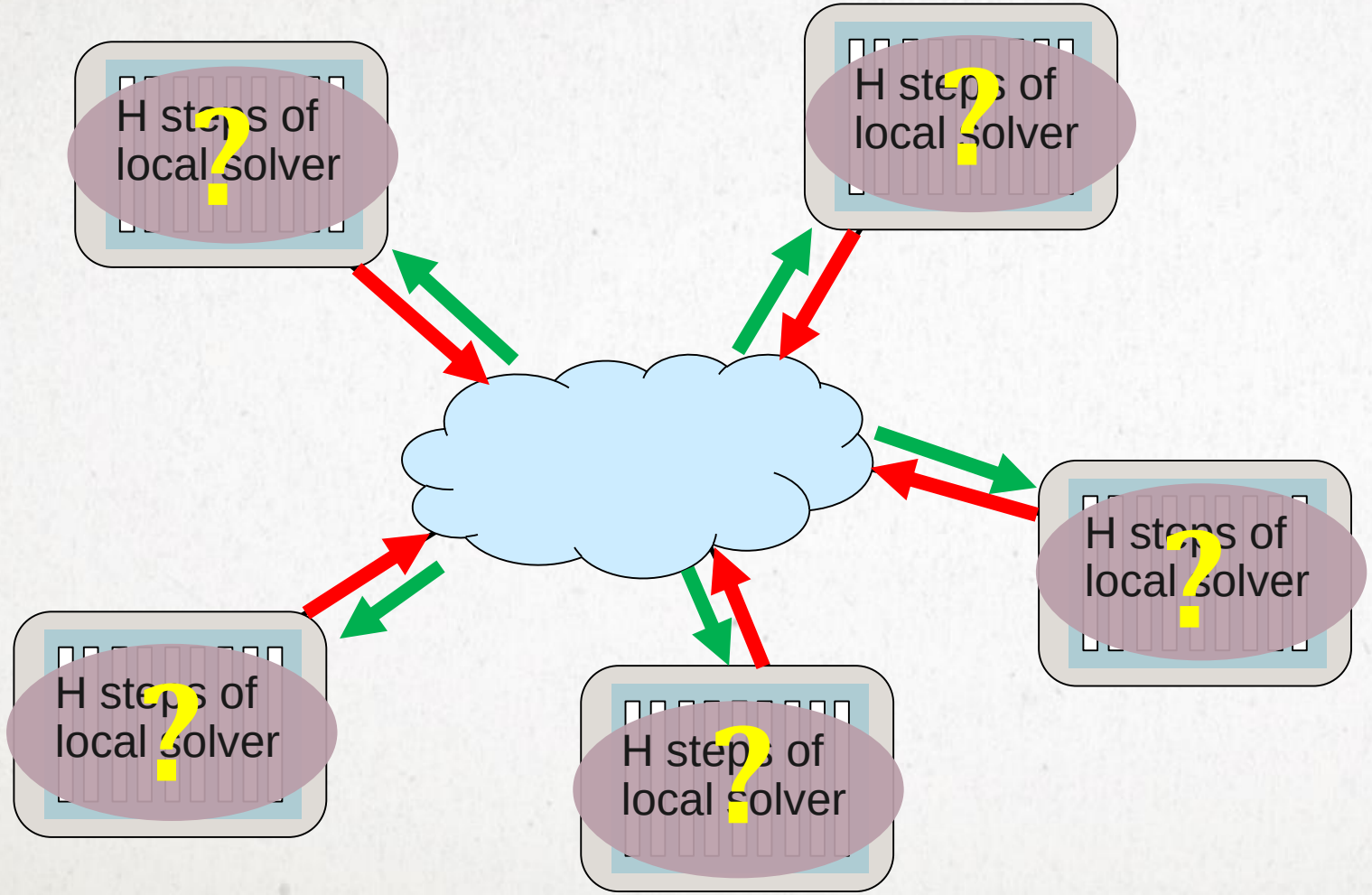
Ridge



SVM

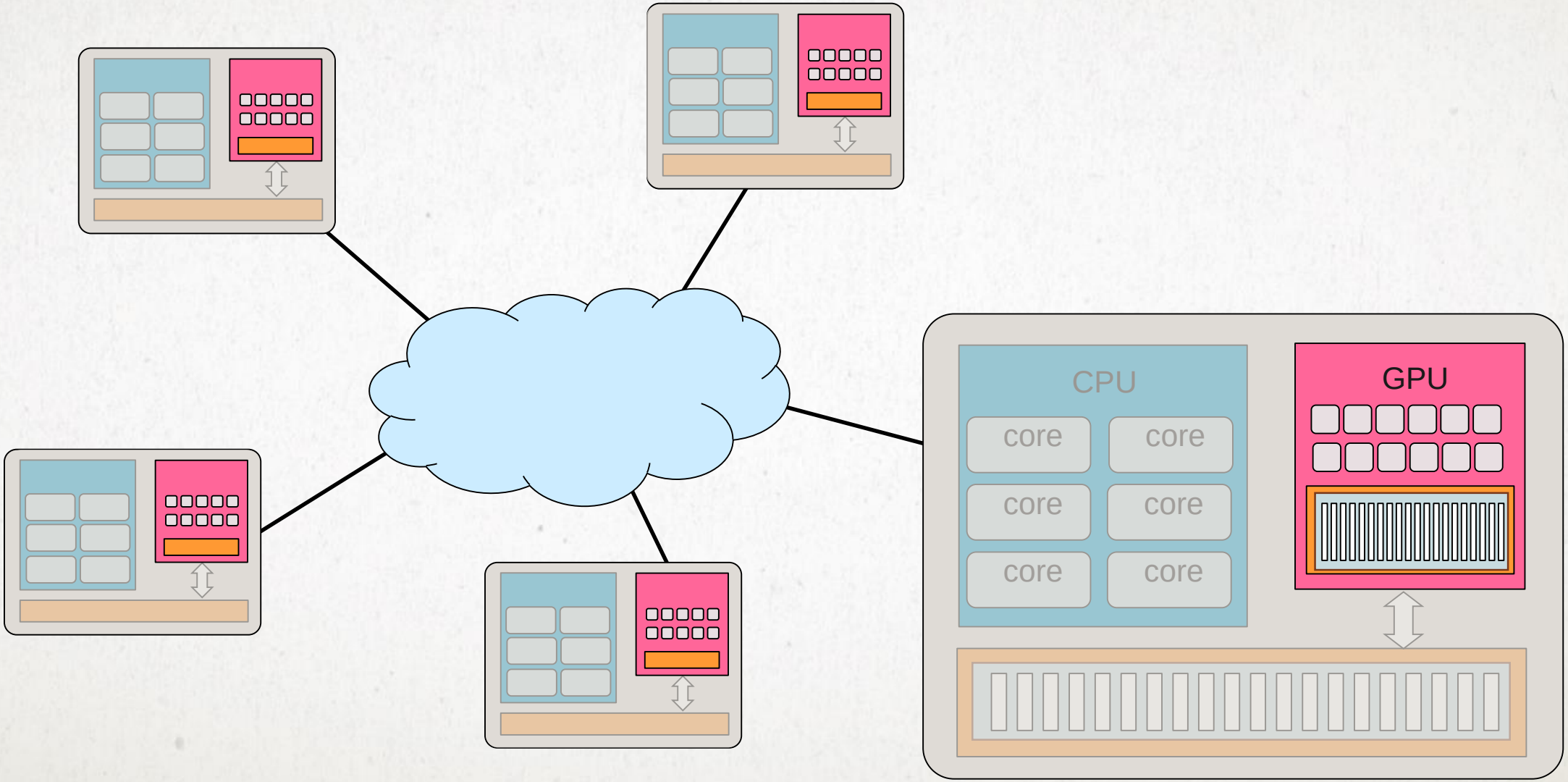
T. Parnell, C. Dünner, K. Atasu, M. Sifalakis, H. Pozidis, „*Large-Scale Stochastic Learning Using GPUs*“, IEEE International Parallel and Distributed Processing Symposium (IPDPS), Lake Buena Vista, FL, 2017

Which local solver should we use?

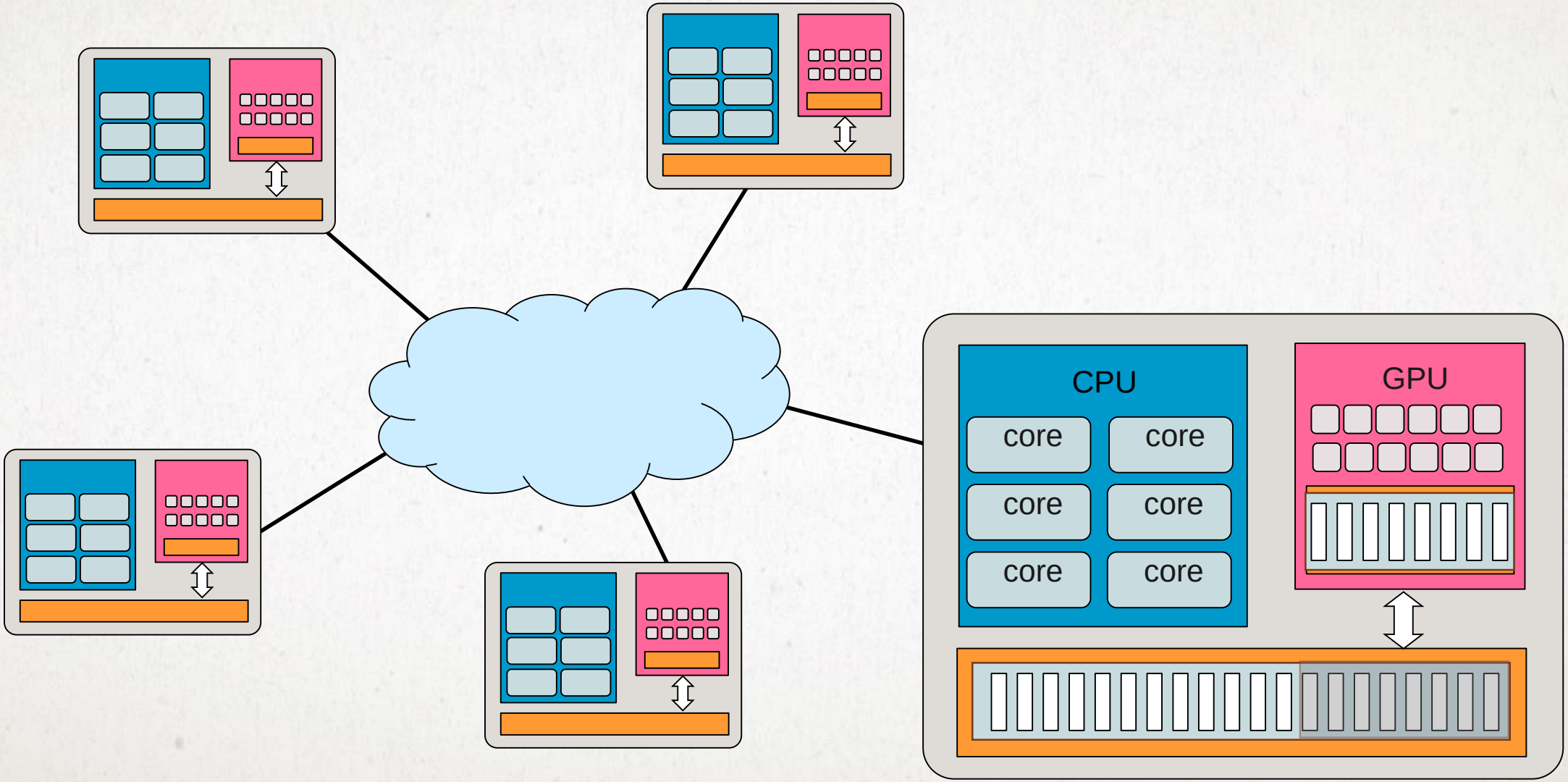


It depends on the available hardware

Heterogeneous System

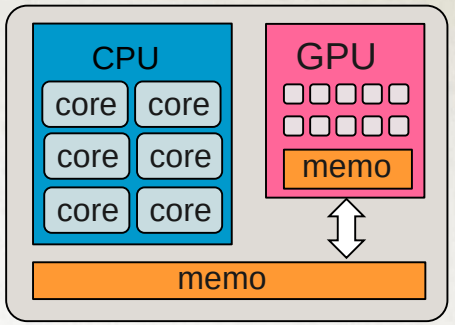


Heterogeneous System



Dual Heterogeneous Learning [NIPS'17]

A scheme to efficiently use Limited-Memory Accelerators for Linear Learning

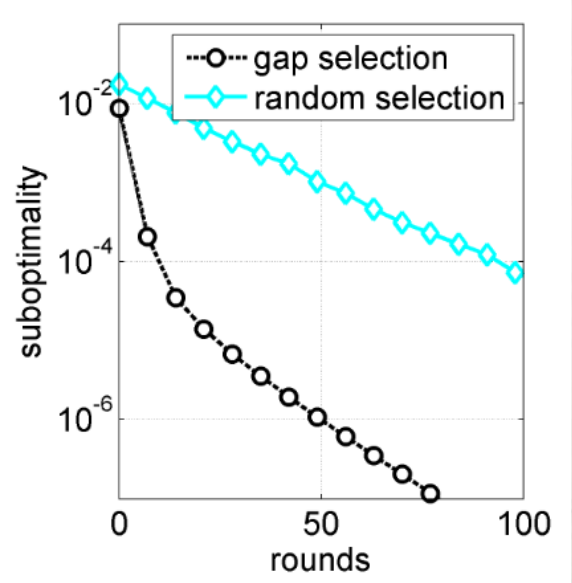


Idea: The GPU should work on the part of the data it can learn most from.

➡ Contribution of individual data columns to the duality gap is indicative for their potential to improving the model

Algorithm 1 Primal-Dual Coordinate Descent

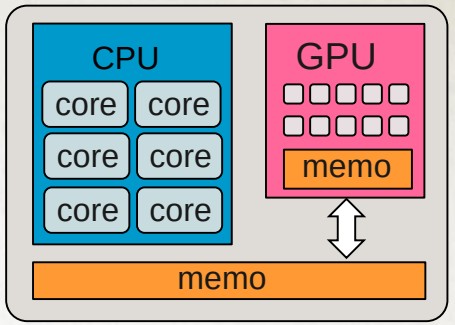
1: Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
2: Initialize $\mathbf{v}^{(0)} = \mathbf{0} \quad (\mathbf{v} := A^\top \mathbf{w})$
3: **for** $t = 0, 1, 2, \ldots..$ **do**
4: ~~randomly select a coordinate $j \in [n]$~~ Select coordinate j with largest duality gap
5: find Δw_j to minimize $f(\mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top) + g_j(w_j + \Delta w_j)$
6: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta w_j \mathbf{e}_j$
7: $\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top$
8: **end for**



Lasso
epsilon dataset

Dual Heterogeneous Learning [NIPS'17]

A scheme to efficiently use Limited-Memory Accelerators for Linear Learning



Idea: The GPU should work on the part of the data it can learn most from.

➡ Contribution of individual data columns to the duality gap is indicative for their potential to improving the model

Duality Gap Computation is expensive!

$$Gap(\boldsymbol{w}) = \sum_j w_j \langle \boldsymbol{x}_j, \boldsymbol{v} \rangle + g_j(w_j) + g_j^*(\boldsymbol{x}_j^\top \boldsymbol{v})$$

- we introduce a ***gap-memory***
- Parallelization of workload between CPU and GPU
  GPU runs algorithm on subset of the data
  CPU computes importance values

DuHL Algorithm

Algorithm 3 DuHL

Initialize $\mathbf{w}^{(0)} = \mathbf{0}, \mathbf{z} = \mathbf{0}$
for $t = 0, 1, 2, \ldots..$
  determine $\mathcal{P} := \arg\max_{\mathcal{P} \subset [n]: |\mathcal{P}| = m} \sum_{j \in \mathcal{P}} z_j$
  refresh GPU memory to contain $A_{[\mathcal{P}]}$.
  on GPU do:
    find $\Delta \mathbf{w}_{[\mathcal{P}]} \leftarrow$ approx. solution to local subproblem
  in parallel on CPU do:
    while GPU not finished
      sample $j \in [n]$
      update $z_j = \mathrm{Gap}_j(w_j^{(t)})$
  $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta \mathbf{w}_{[\mathcal{P}]}$

C. Dünner, T. Parnell, M. Jaggi, *„Efficient Use of Limited-Memory Accelerators for Linear Learning on Heterogeneous Systems "*, NIPS, Long Beach, CA, 2017

DuHL: Performance Results

ImageNet dataset 30GB

GPU: NVIDIA Quadra M4000 (8GB)
CPU: 8-core Intel Xeon X86 (64GB)

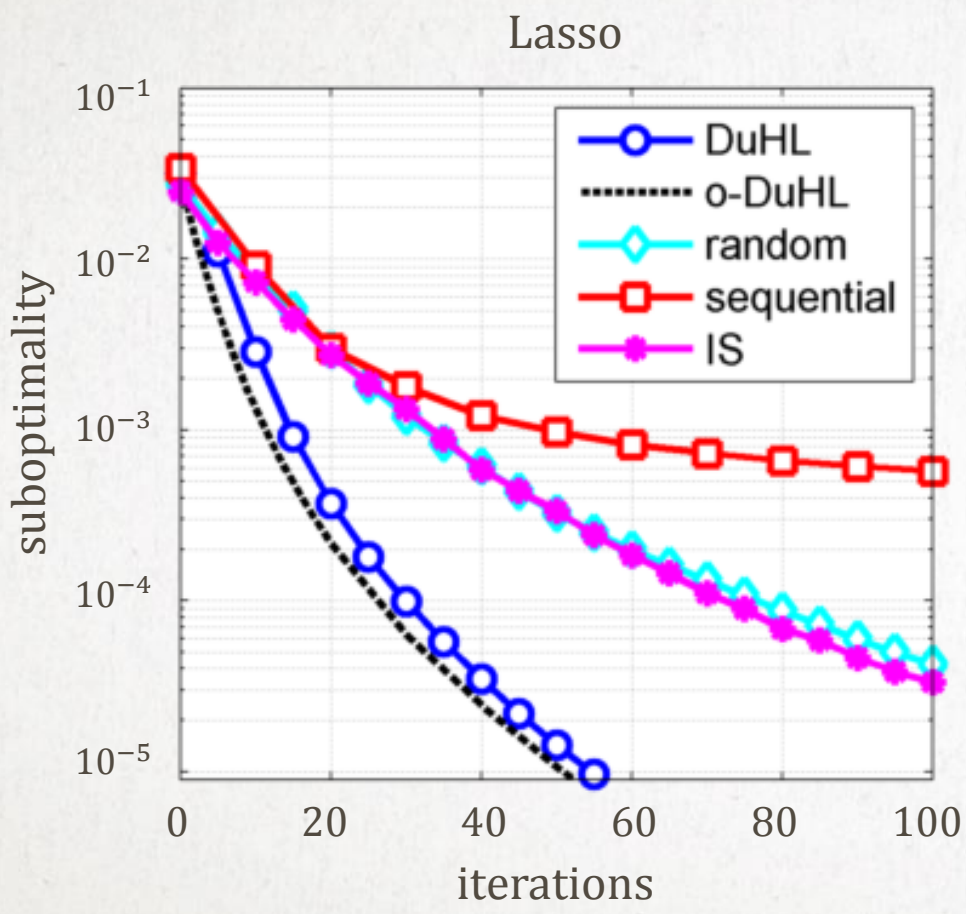


Fig 1: Superior convergence properties of DuHL over existing schemes

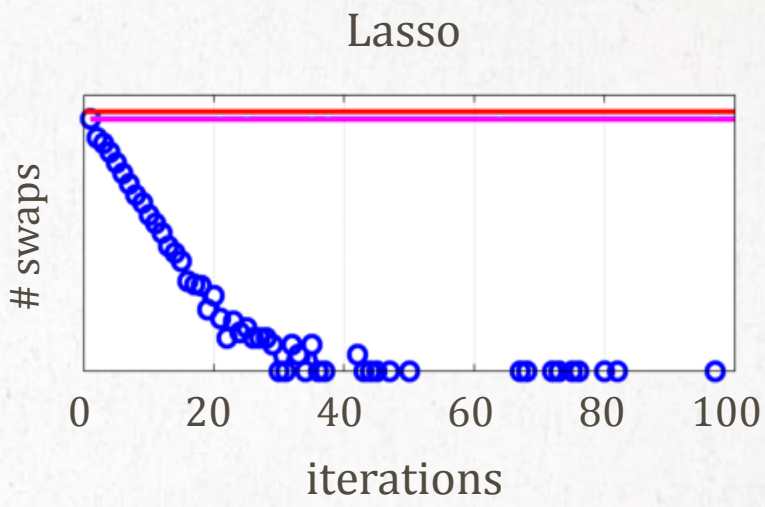


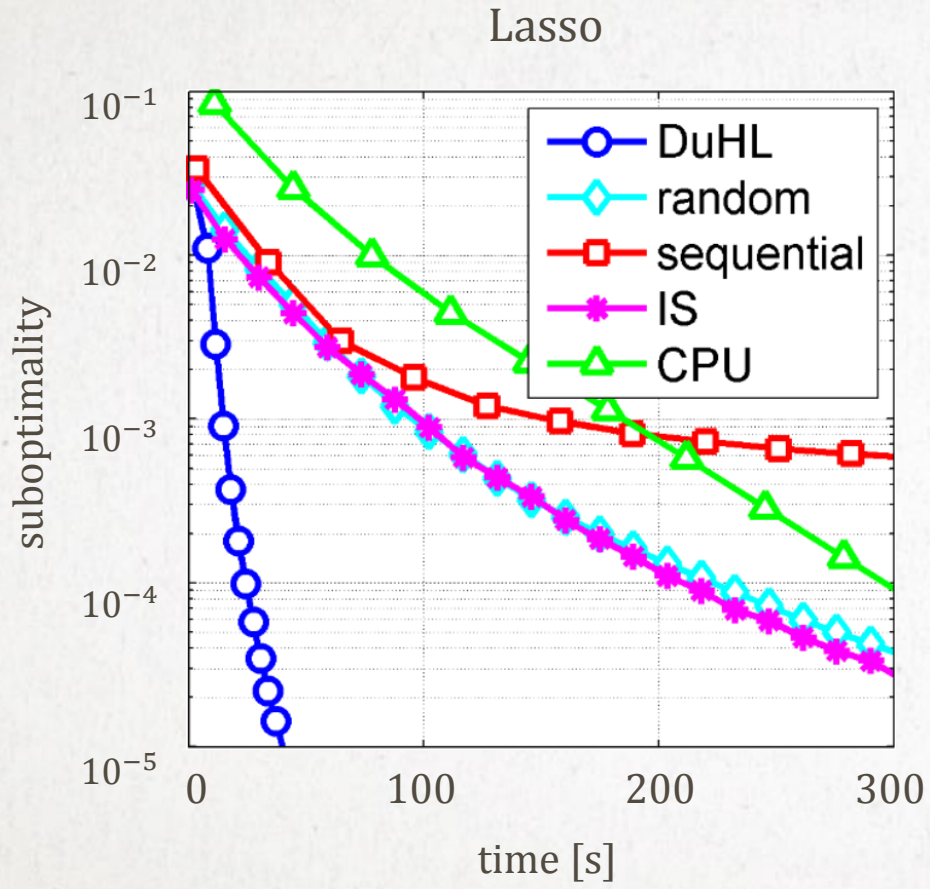
Fig 2: I/O efficiency of DuHL

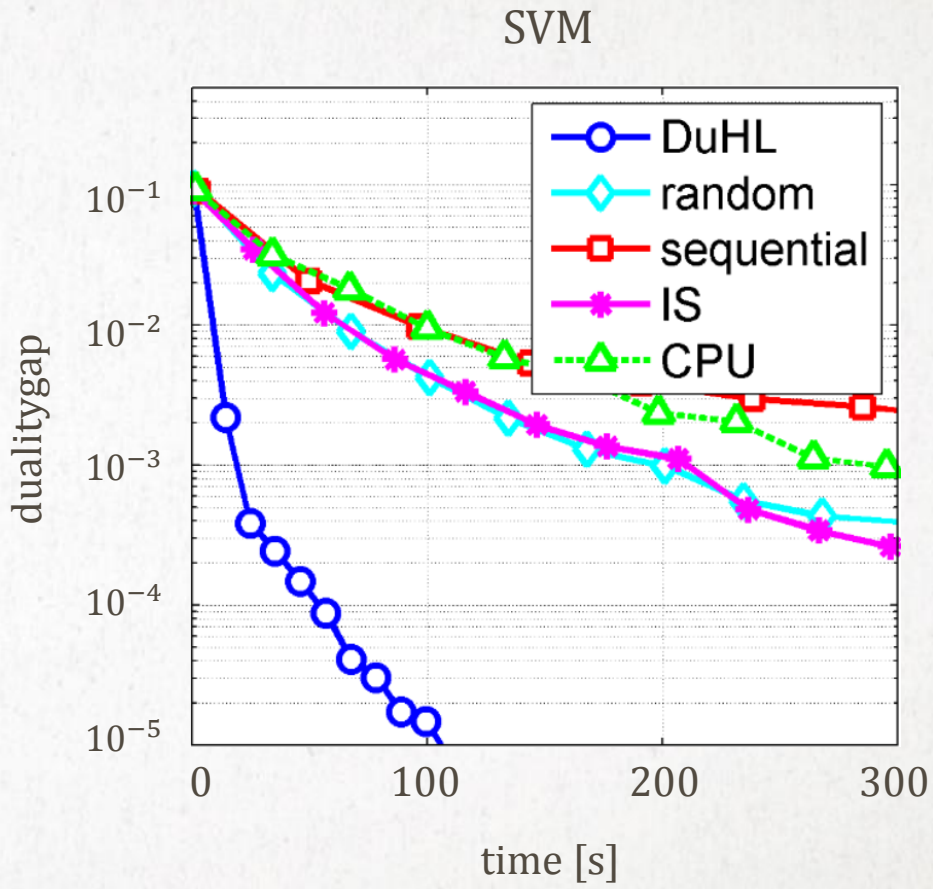
Reduced I/O cost and faster convergence accumulate to **10x speedup**

DuHL: Performance Results

ImageNet dataset 30GB

GPU: NVIDIA Quadra M4000 (8GB)
CPU: 8-core Intel Xeon X86 (64GB)





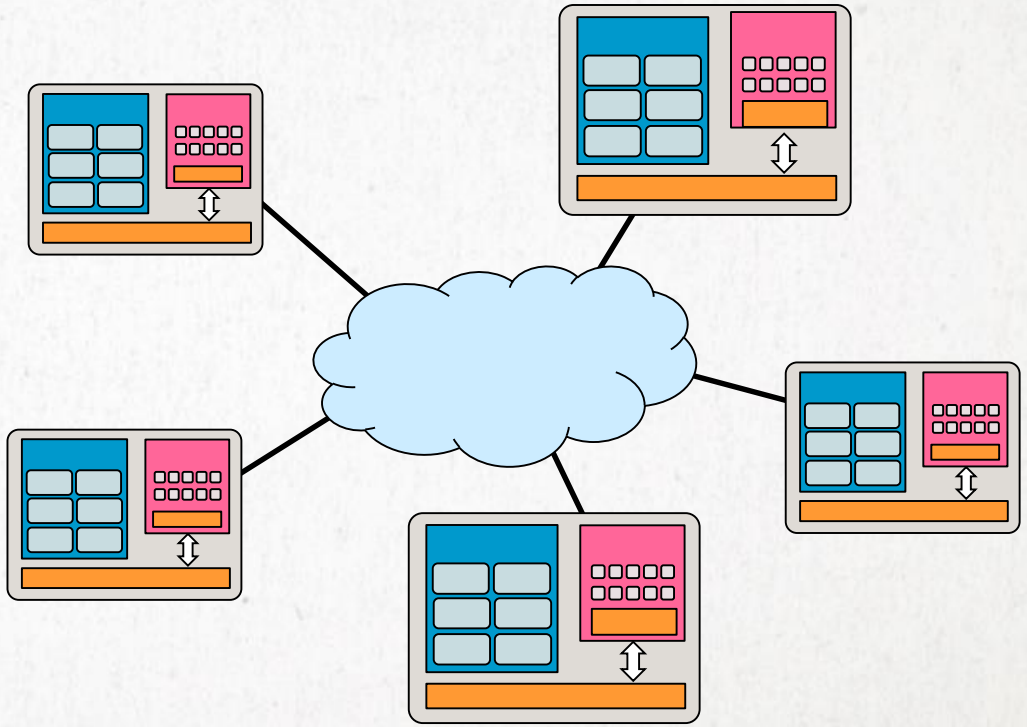
Reduced I/O cost and faster convergence accumulate to **10x speedup**

Combining it all

A library for ultra-fast machine learning

Goal: remove training as a bottleneck

- Enable seamless retraining of models
- Enable agile development
- Enable training on large-scale datasets
- Enable high quality insights



- ✔ Exploit Primal-Dual Structure of ML problems to minimize communication
- ✔ Offer GPU acceleration
- ✔ Implement DuHL for efficient utilization of limited-memory accelerators
- ✔ Improved memory management of Spark

Combining it all

A library for ultra-fast machine learning

Goal: remove training as a bottleneck

- Enable seamless retraining of models
- Enable agile development
- Enable training on large-scale datasets
- Enable high quality insights

- ✔ Exploit Primal-Dual Structure of ML problems to minimize communication
- ✔ Offer GPU acceleration
- ✔ Implement DuHL for efficient utilization of limited-memory accelerators
- ✔ Improved memory management of Spark

Tera-Scale Advertising Application

Predict whether a user will click on a given advert based on an anonymized set of features.

training: 1 billion example
testing: 100 million unseen examples

Linear Regression (8 executors)

Library	Time (s)	MSE	Speed-up
spark.ml	20,000	6.1%	n/a
our library	**7**	6.1%	2800x

Power8 (Minksy) infrastructure
8 x P100 GPU

Summary

- Distributed Algorithm that offer tunable hyper-parameters are of particular practical interest
- A user may expect orders of magnitude improvement by optimizing such parameters
- GPUs can accelerate machine learning workloads by an order of magnitude if algorithms are carefully designed
- DuHL enables GPU acceleration even if the data exceeds the capacity of the GPU memory
- Combining all this knowledge we can remove training time as a bottleneck for ML applications

IBM

Questions?