Numerically-intensive Machine Learning at Scale

Michael W. Mahoney

(RISELab, ICSI, and Department of Statistics, UC Berkeley)

November 2017

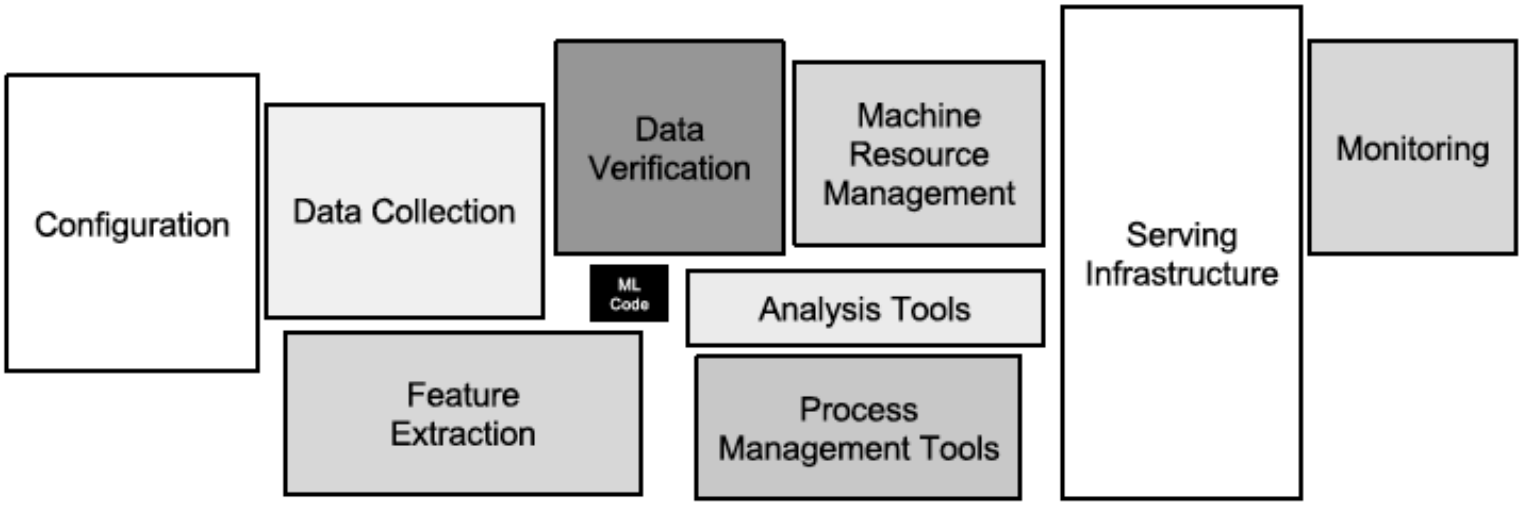


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Hidden Technical Debt in Machine Learning Systems, D. Sculley, et al.

Overview

Linear Algebra in Spark for science problems

- CX and SVD/PCA implementations and performance
- Applications of the CX and PCA matrix decompositions
- To mass spec imaging, climate science, etc.

The Next Step: Alchemist

- Combining Spark and MPI

Communication-avoiding LA/ML

- Going beyond CA-LA to CA-ML

Overview

Linear Algebra in Spark for science problems

- CX and SVD/PCA implementations and performance
- Applications of the CX and PCA matrix decompositions
- To mass spec imaging, climate science, etc.

The Next Step: Alchemist

- Combining Spark and MPI

Communication-avoiding LA/ML

- Going beyond CA-LA to CA-ML

Where do you run your linear algebra?

Single machine

- Think about RAM, call LAPACK, etc.
- Someone else thought about numerical issues, memory hierarchies, etc.
- This is the 99%

Supercomputer

- High end, compute-intensive.
- Big emphasis on HPC (High Performance Computing)
- C+MPI, etc.

Distributed data center

- High end, data-intensive
- BIG emphasis on HPC (High Productivity Computing)
- Databases, MapReduce/Hadoop, Spark, etc.

Two related issues with eigen-analysis

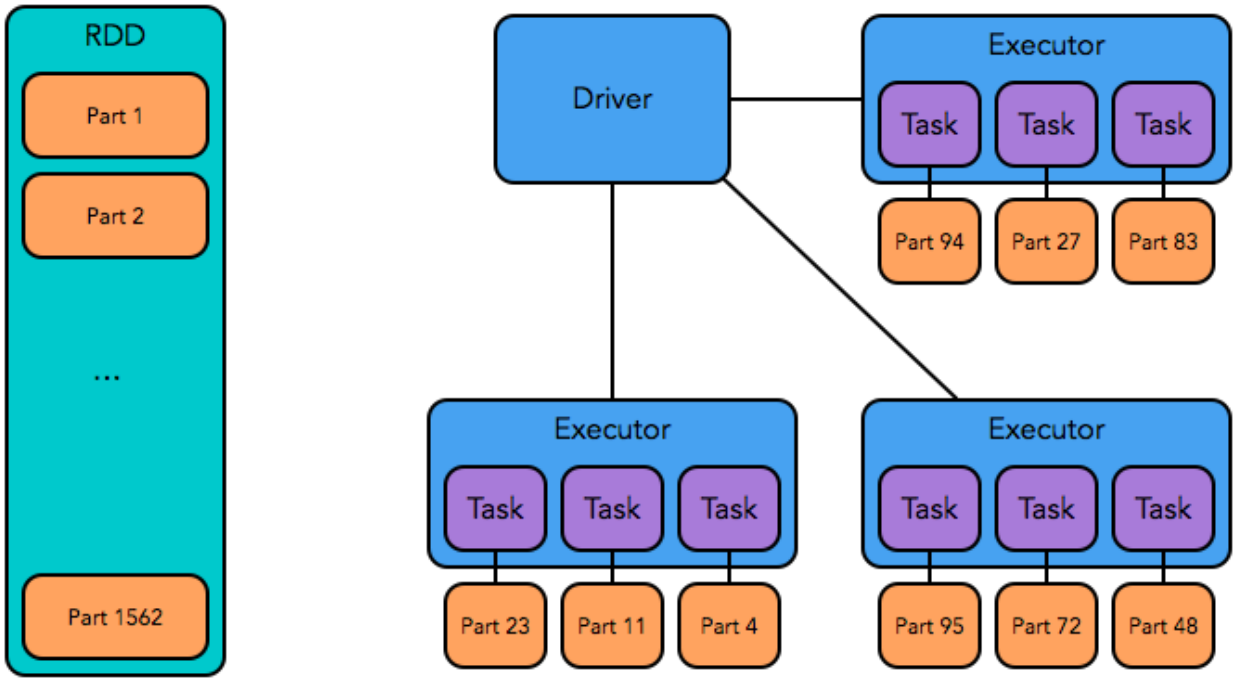
Computing large SVDs: computational time

- In commodity hardware (e.g., a 4GB RAM, dual-core laptop), using MatLab 7.0 (R14), the computation of the SVD of the dense 2,240-by-447,143 matrix A takes ca 20 minutes.
- Computing this SVD is not a one-liner, since we can not load the whole matrix in RAM (runs out-of-memory in MatLab).
- Instead, compute the SVD of AA^T.
- In a similar experiment, compute **1,200 SVDs** on matrices of dimensions (approx.) 1,200-by-450,000 (roughly, a full leave-one-out cross-validation experiment) (DLP2010)

Selecting *actual columns* that "capture the structure" of the top PCs

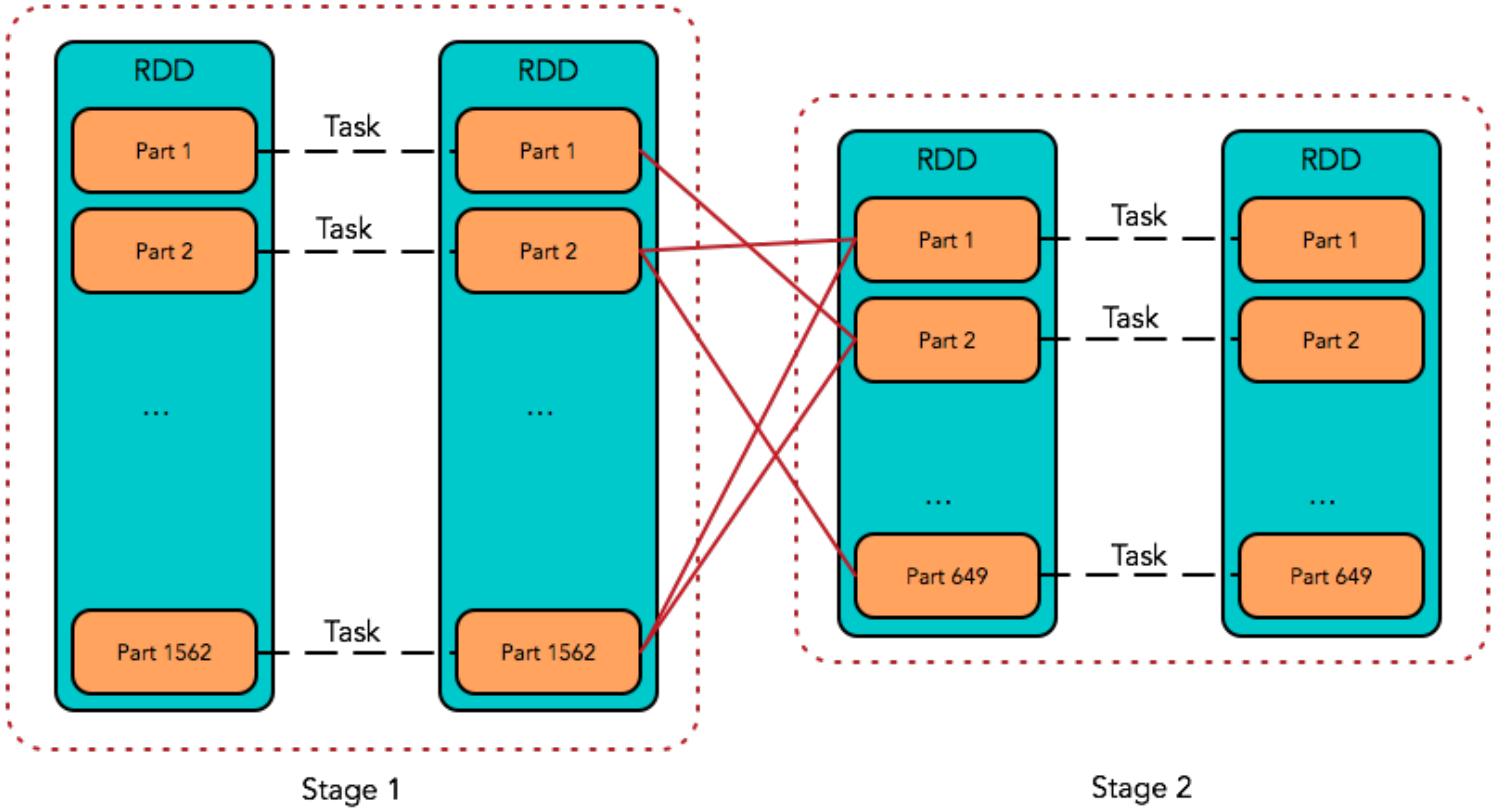
- Combinatorial optimization problem; hard even for small matrices.
- Often called the Column Subset Selection Problem (CSSP).
- Not clear that such "good" columns even exist.
- Avoid "reification" problem of "interpreting" singular vectors!
- (Solvable in "random projection time" with CX/CUR decompositions! (PNAS, MD09))

Spark Architecture



- Data parallel programming model
- Resilient distributed datasets (RDDs) (think: distributed array type)
- RDDs can optionally be cached in memory b/w iterations
- Driver forms DAG, schedules tasks on executors

Spark Communication



- Computation operate on one RDD to produce another RDD
- Each overall job (DAG) broken into stages
- Stages broken into parallel, independent tasks
- Communication happens only between stages

Why do linear algebra in Spark?

Pros:

- *Widely used*
- *Easier to use for non-experts*
- An entire ecosystem that can be used before and after the NLA computations
- Spark can take advantage of available single-machine linear algebra codes (e.g. through netlib-java)
- Automatic fault-tolerance
- Transparent support for out of memory calculations

Cons:

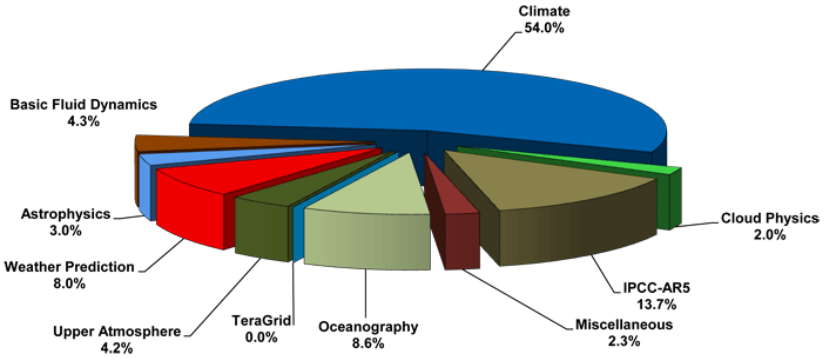
- Classical MPI-based linear algebra algorithms are faster and more efficient
- No way, currently, to leverage legacy parallel linear algebra codes
- JVM matrix size restrictions, and RDD rigidity

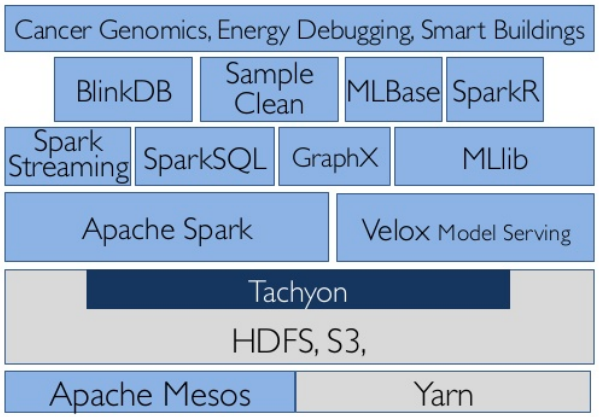
Our Goals

- ***Provide implementations*** of low-rank factorizations (PCA, NMF, and randomized CX) in Spark
- Apply low-rank matrix factorization methods ***to TB-scale scientific datasets*** in Spark
- Understand Spark performance on ***commodity clusters vs HPC platforms***
- ***Quantify the scalability gaps*** between highly-tuned C/MPI and current Spark-based implementations
- ***Provide a general-purpose interface*** for matrix-based algorithms between Spark and traditional MPI codes

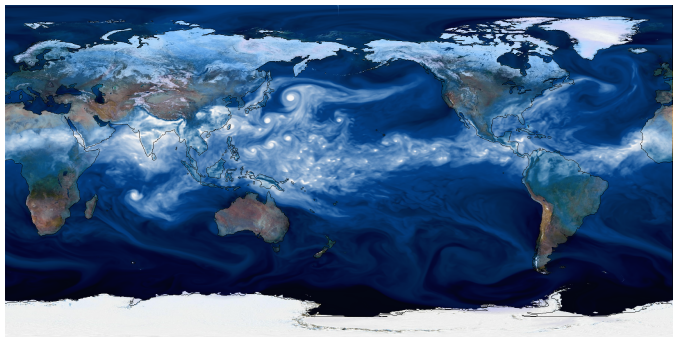
Motivation

- **NERSC**: Spark for data-centric workloads and scientific analytics
- **AMPLab**: characterization of linear algebra in Spark (MLlib, MLMatrix)
- **Cray**: customers demand for Spark; understand performance concerns



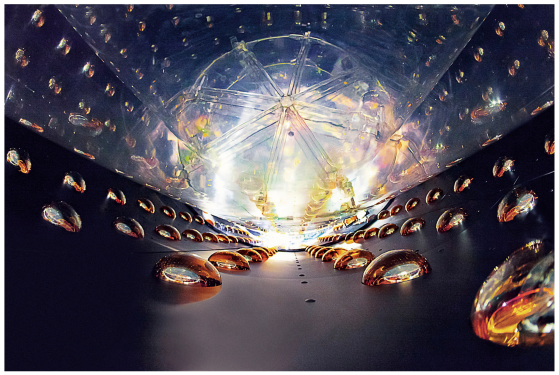


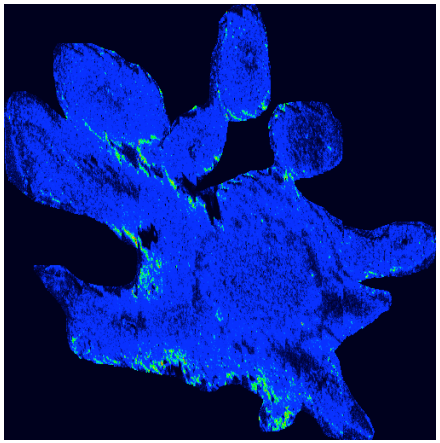
Three Science Drivers



Climate Science:
extract trends in variations of oceanic and atmospheric variables (**PCA**)

Nuclear Physics:
learn useful patterns for classification of subatomic particles (**NMF**)





Mass Spectrometry:
location of chemically important ions (**CX**)

Datasets

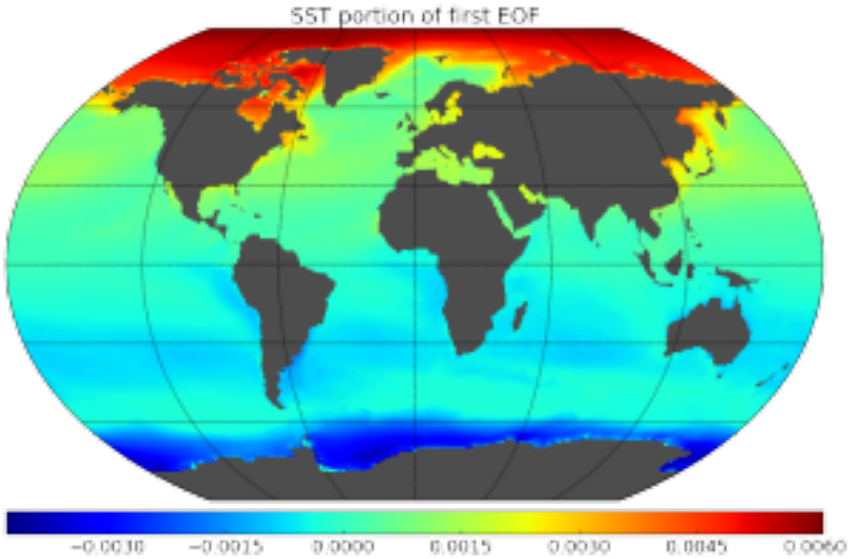
Science Area	Format/Files	Dimensions	Size
MSI	Parquet/2880	$8,258,911 \times 131,048$	1.1TB
Daya Bay	HDF5/1	$1,099,413,914 \times 192$	1.6TB
Ocean	HDF5/1	$6,349,676 \times 46,715$	2.2TB
Atmosphere	HDF5/1	$26,542,080 \times 81,600$	16TB

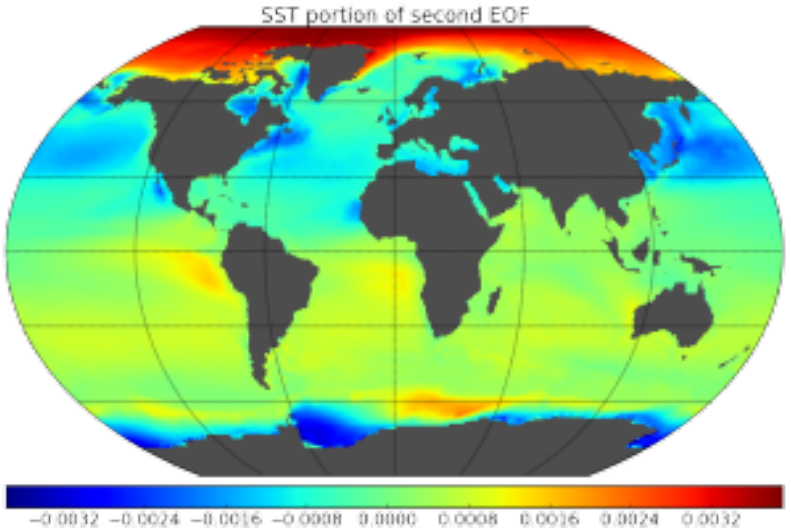
MSI — a sparse matrix from measurements of drift times and mass charge ratios at each pixel of a sample of *Peltatum*; used for CX decomposition

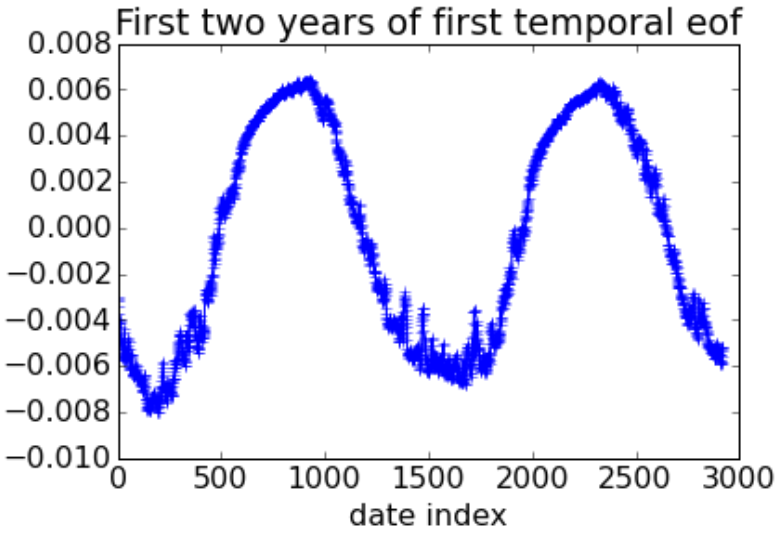
Daya Bay — neutrino sensor array measurements; used for NMF

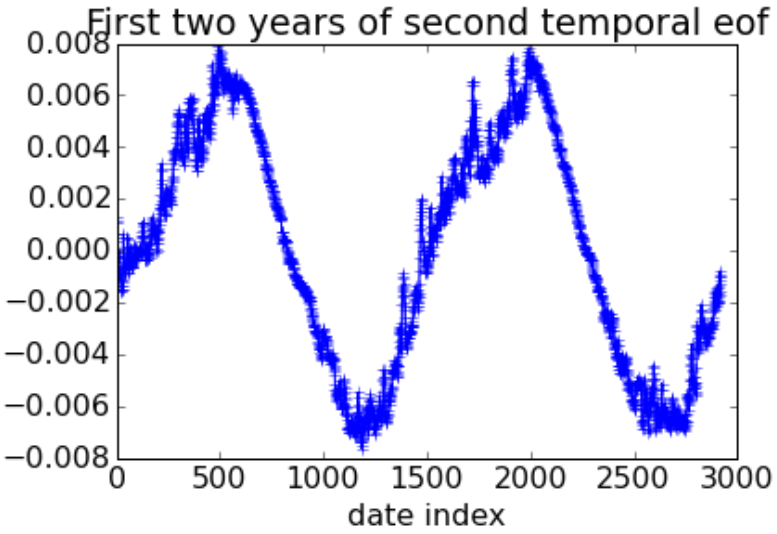
Ocean and Atmosphere — climate variables (ocean temperature, atmospheric humidity) measured on a 3D grid at 3 or 6 hour intervals over about 30 years; used for PCA

CFSR Ocean Temperature Dataset (II)

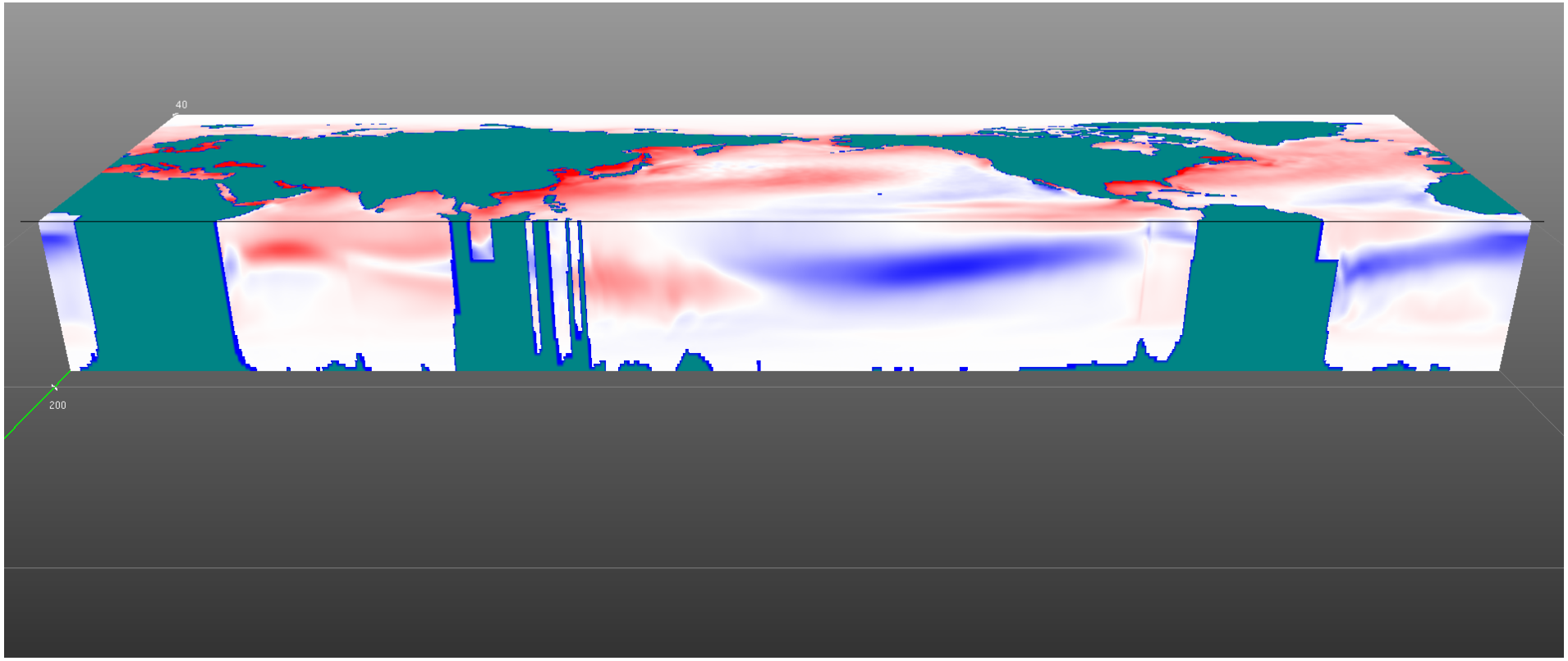








Climate Science Results on Ocean (CFSRO) dataset



- First principal component of temperature field at 180 degree latitude.
- Clear that there is a significant vertical component to the PCs which are lost when you do the traditional surface-only analyses

Running times for NMF and PCA

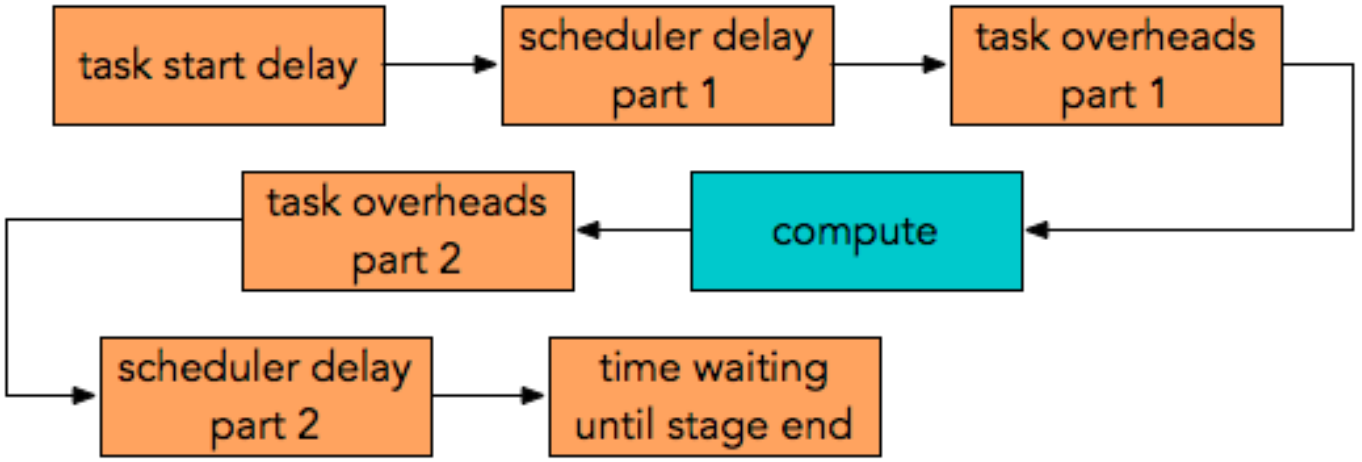
Cori's specs:

- 1630 compute nodes,
- 128 GB/node,
- 32 2.3GHz Haswell cores/node

	Nodes / cores	MPI Time	Spark Time	Gap
NMF	50 / 1,600	1 min 6 s	4 min 38 s	4.2x
	100 / 3,200	45 s	3 min 27 s	4.6x
	300 / 9,600	30 s	70 s	2.3x
PCA (2.2TB)	100 / 3,200	1 min 34 s	15 min 34 s	9.9x
	300 / 9,600	1 min	13 min 47 s	13.8x
	500 / 16,000	56 s	19 min 20 s	20.7x
PCA (16TB)	MPI: 1,600 / 51,200 Spark: 1,522 / 48,704	2 min 40 s	69 min 35 s	26x

- Anti-scaling!
- And it worsens both with concurrency and data size.

Spark Overheads: the view of one task



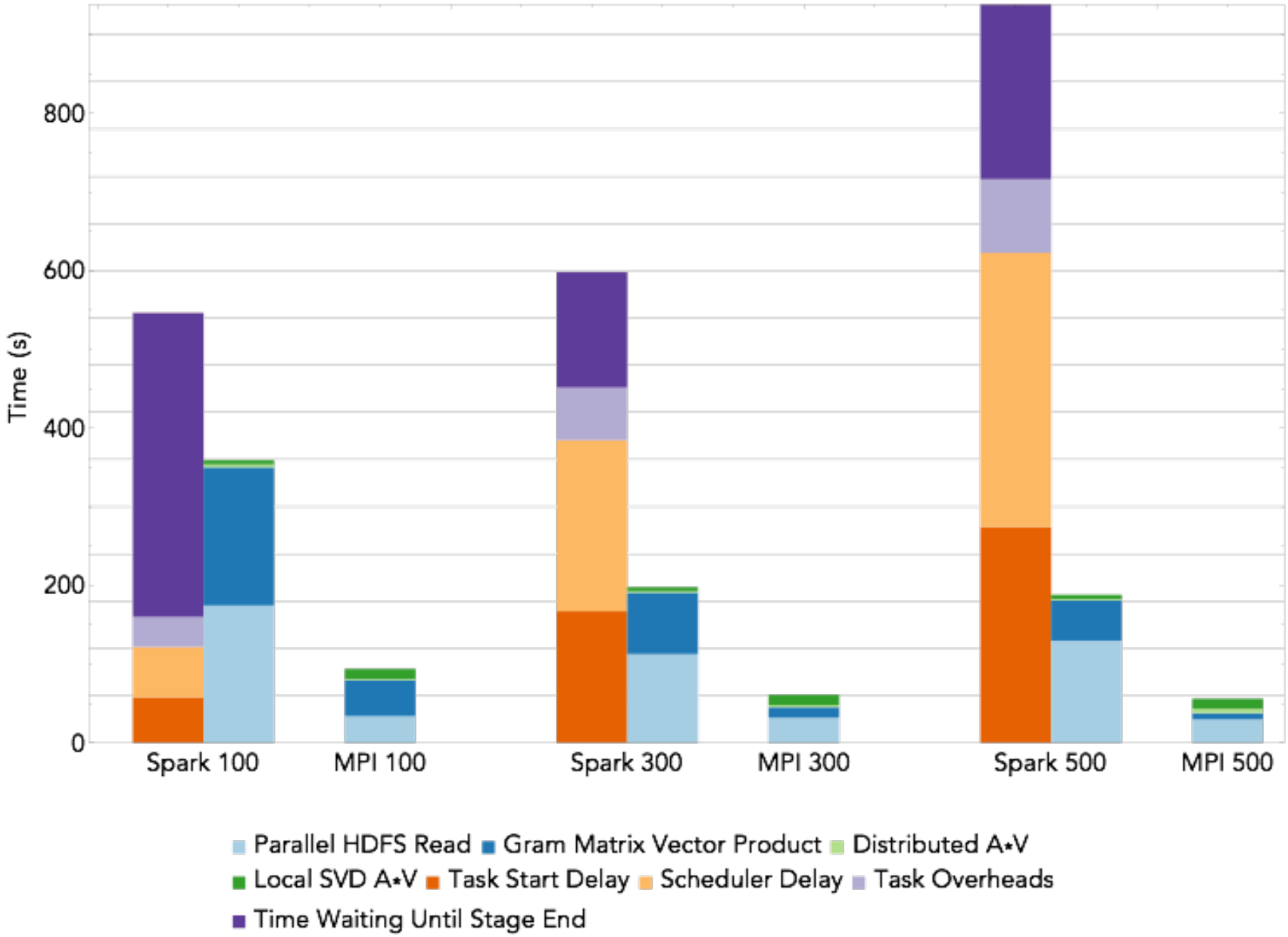
task start delay = (time between stage start and when driver sends task to executor)

scheduler delay = (time between task being sent and time starts deserializing)+ (time between task result serialization and driver receiving task's completion message)

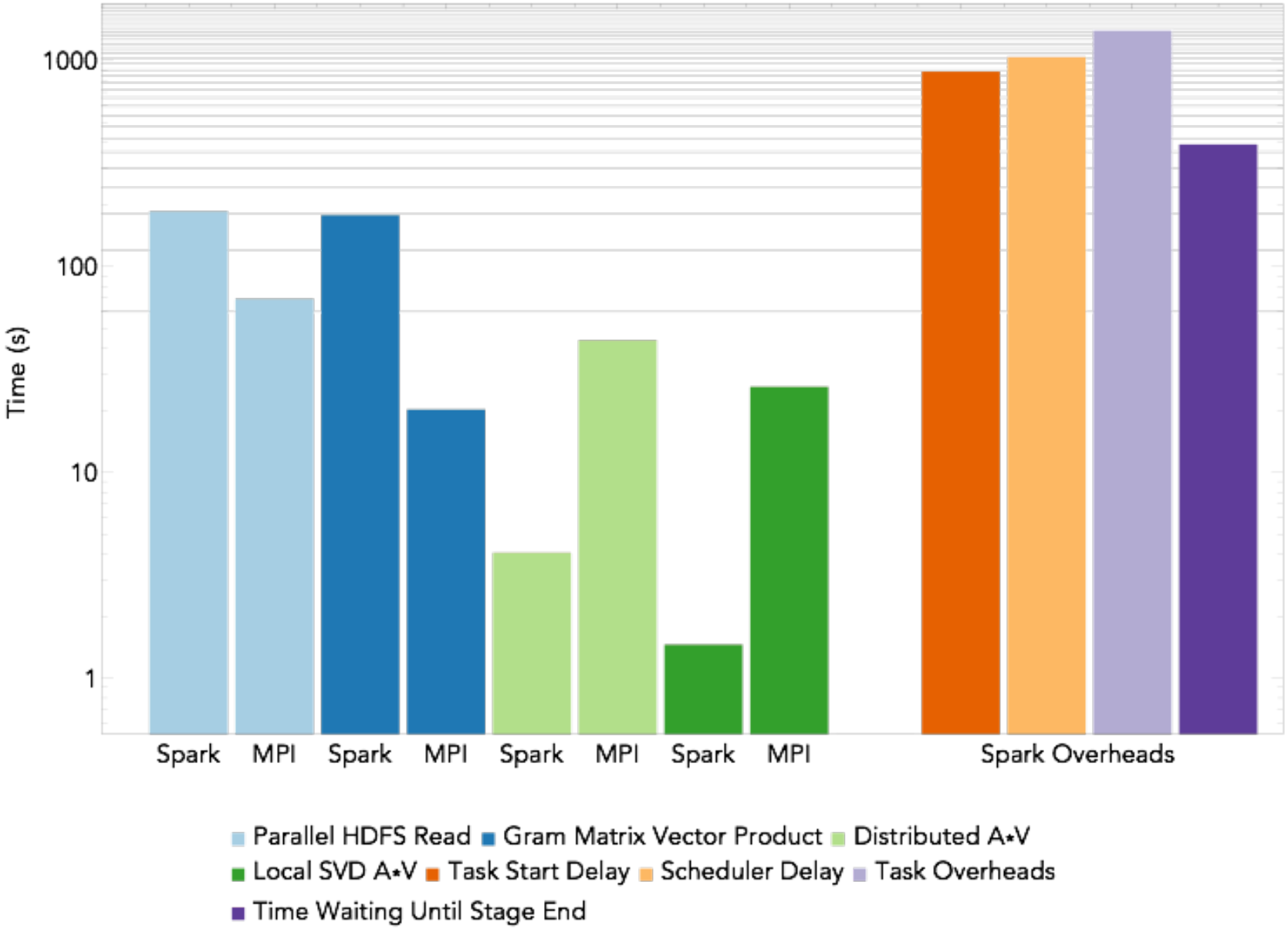
task overhead time = (fetch wait time) + (executor deserialize time) + (result serialization time) + (shuffle write time)

time waiting until stage end = (time waiting for final task in stage to end)

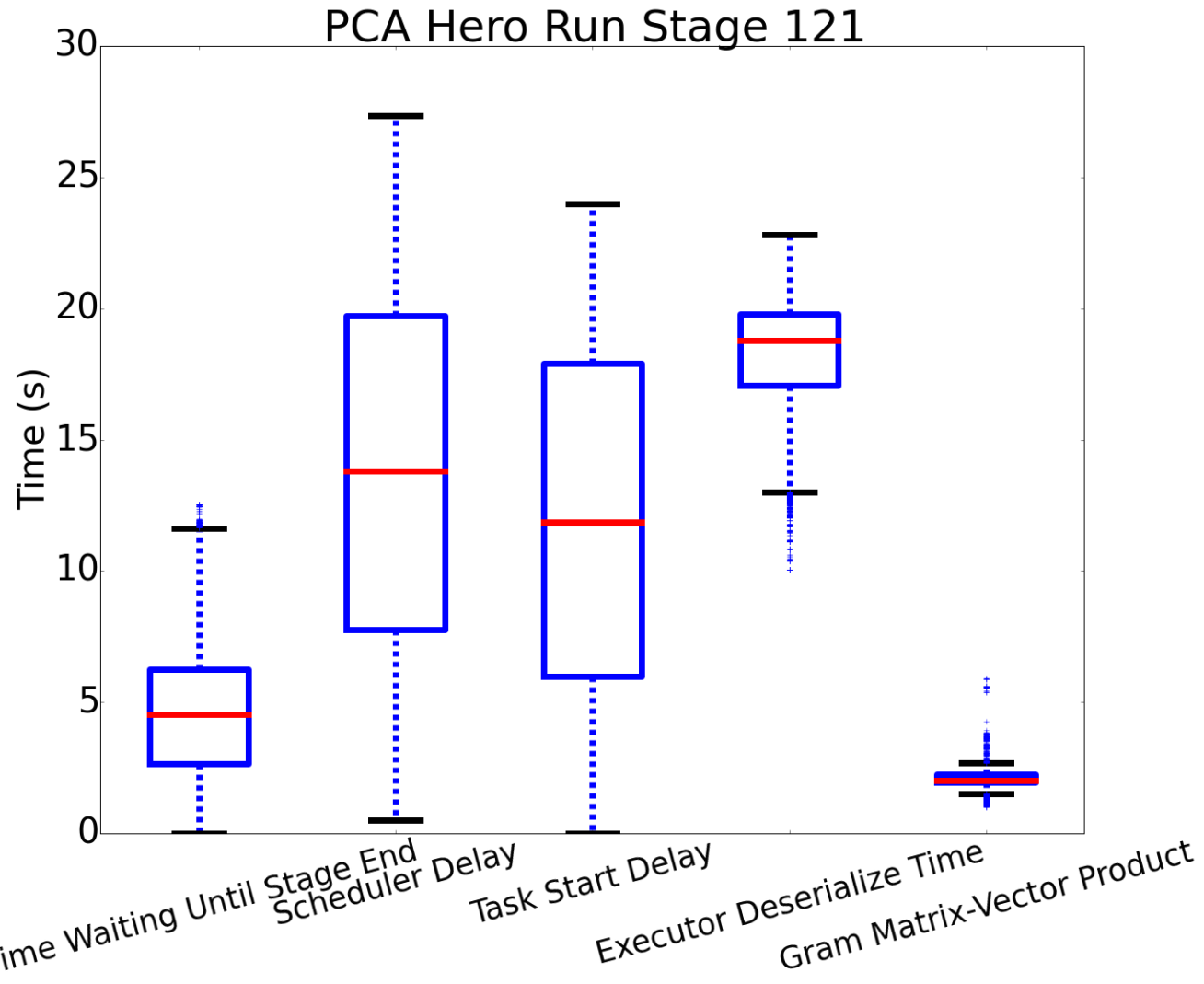
PCA Run Times: rank 20 PCA of 2.2TB Climate



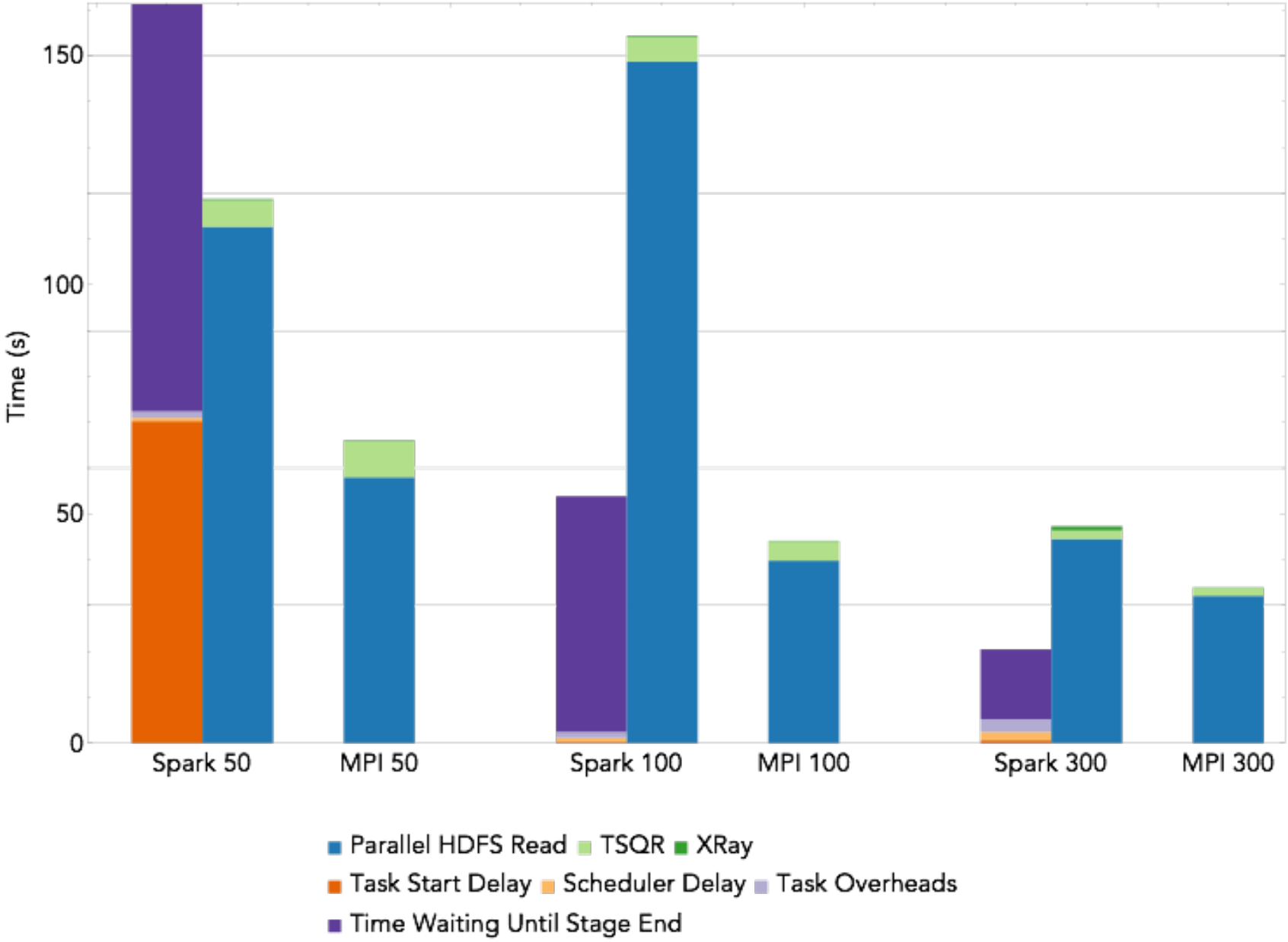
Rank 20 PCA of 16 TB Climate using 48K+ cores



Spark PCA Overheads: 16 TB Climate,1522 nodes



NMF Run Times: rank 10 NMF of 1.6TB Daya Bay



Overview

Linear Algebra in Spark for science problems
- CX and SVD/PCA implementations and performance
- Applications of the CX and PCA matrix decompositions
- To mass spec imaging, climate science, etc.

The Next Step: Alchemist
- Combining Spark and MPI

Communication-avoiding LA/ML
- Going beyond CA-LA to CA-ML

MPI vs Spark: Lessons Learned

- Algorithm choice and data layout choices are ***constrained by the bulk synchronous, data parallel programming model of Spark and its core data structure, the RDD***
- Even with favorable data (tall and skinny) and well-adapted algorithms, ***Spark LA is 2x-26x slower than MPI when IO is included***
- ***Spark overheads are orders of magnitude higher than the computations*** in PCA (time till stage end, scheduler delay, task start delay, executor deserialize time). A more efficient algorithm is needed

The Next Step: Alchemist

- Since Spark is 4+x slower than MPI, propose sending the matrices to MPI codes, then receiving the results
- For efficiency, want as little overhead as possible (File I/O, RAM, network usage, computational efficiency)

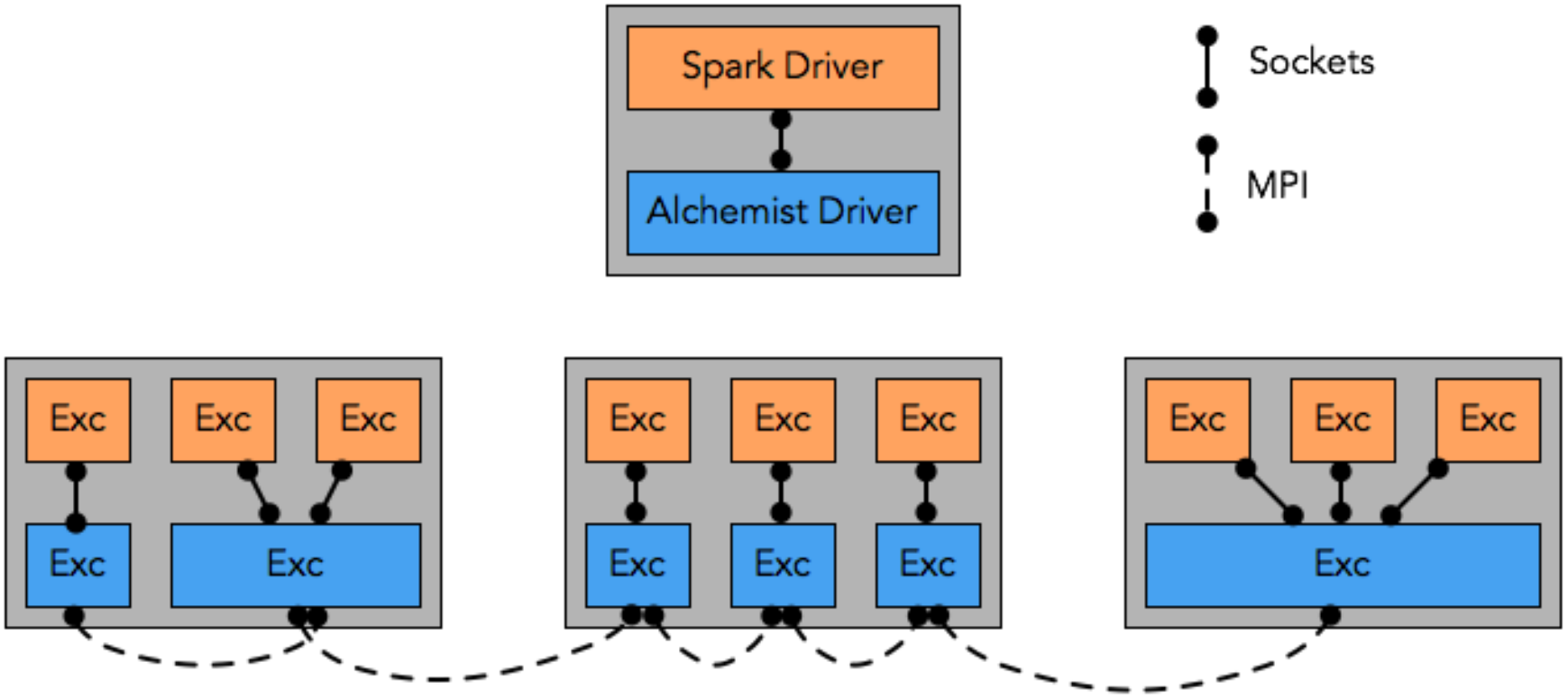
Strawman approaches:

1. Write to HDFS: *slow file I/O, manual data layout*
2. Apache Ignite (and Alluxio, etc.): *requires using C/C++ interfaces, manual data layout, extra copy in memory, TCP/IP*

Our approach:

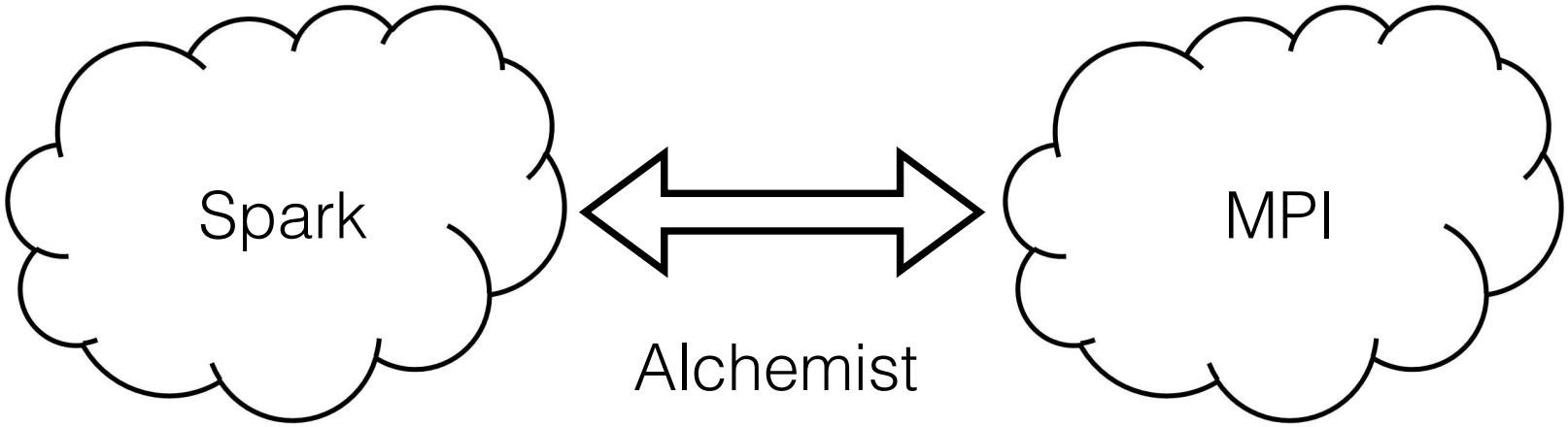
Use in-memory transfer, and transparently provide data relayout

Current Alchemist Architecture



- Exploit locality to reduce communication
  - Allow for hybrid OpenMP/MPI

Using Alchemist



Spark:

1) Sends the metadata for input and output matrices to Alchemist

2) Sends the matrix to Alchemist using sockets

3) Waits on a matrix from the Alchemist gateway using sockets

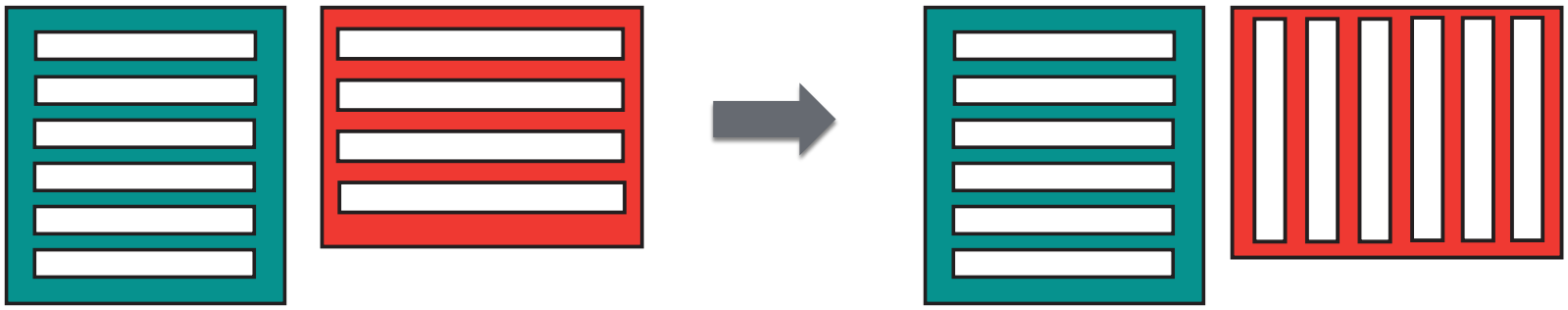
Alchemist:

1) Repartitions the matrix for MPI using Elemental

2) Executes the MPI codes

3) Repartitions the output and returns to Spark

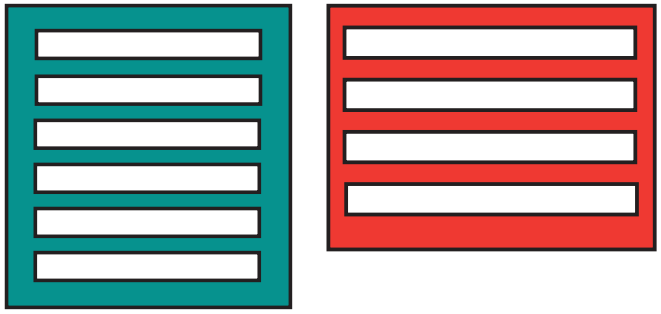
Example: Matrix Multiplication



Requires expensive shuffles in Spark:

- Matrices/RDDs are row partitioned
- One must be converted to be column-partitioned
- This requires an all-to-all shuffle that often fails ***even for matrices that could fit in memory on one executor***

Example: Matrix Multiplication



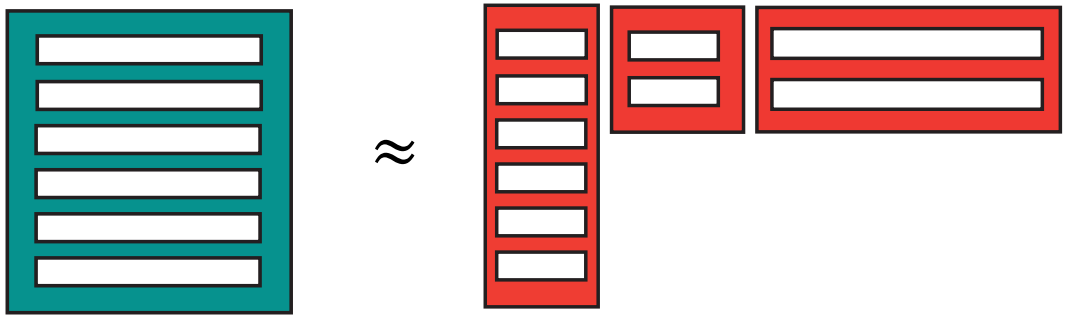
A: 100K-by-10K (8 GB)
B: 10K-by-100K (8 GB)
C=AB: 100K-by-100K (80 GB)

Setup:
-15 Spark and 15 Alchemist nodes
-128 GB RAM and 32 cores per node

	Send	*Compute*	*Receive*
Alchemist	7.78 s	106s	38s
Spark	-	Fail after 30 min	-

Example: Truncated SVD

A: 100K-by-5K (4 GB)



Setup:
-15 Spark and 15 Alchemist nodes
-128 GB RAM and 32 cores per node

	Send	*Compute*	*Receive*
Alchemist	15.7 s	31.8s	5.5s
Spark	-	636.3s	-

Overview

Linear Algebra in Spark for science problems
- CX and SVD/PCA implementations and performance
- Applications of the CX and PCA matrix decompositions
- To mass spec imaging, climate science, etc.

The Next Step: Alchemist
- Combining Spark and MPI

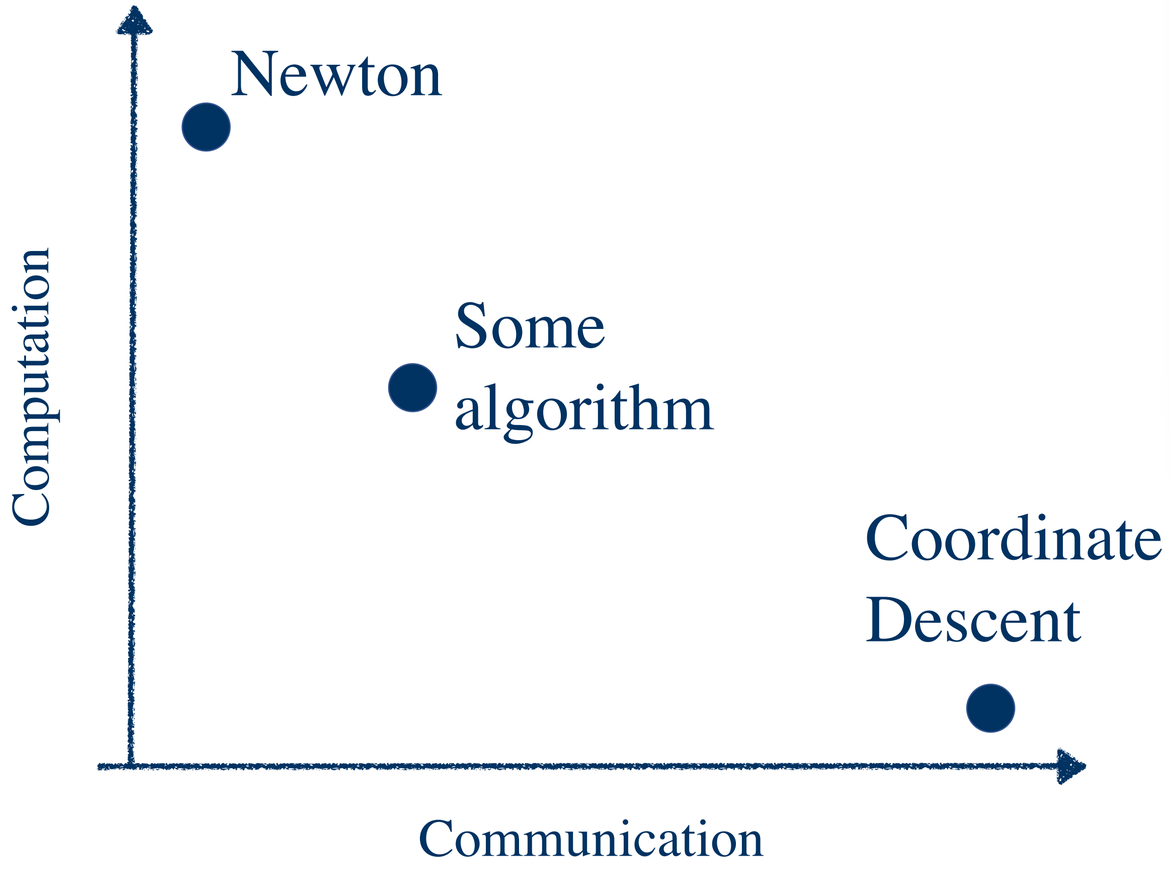
Communication-avoiding LA/ML
- Going beyond CA-LA to CA-ML

Motivation

Need for faster optimization/ML algorithms with less communication

Processor speed << Communication speed
Gap is growing

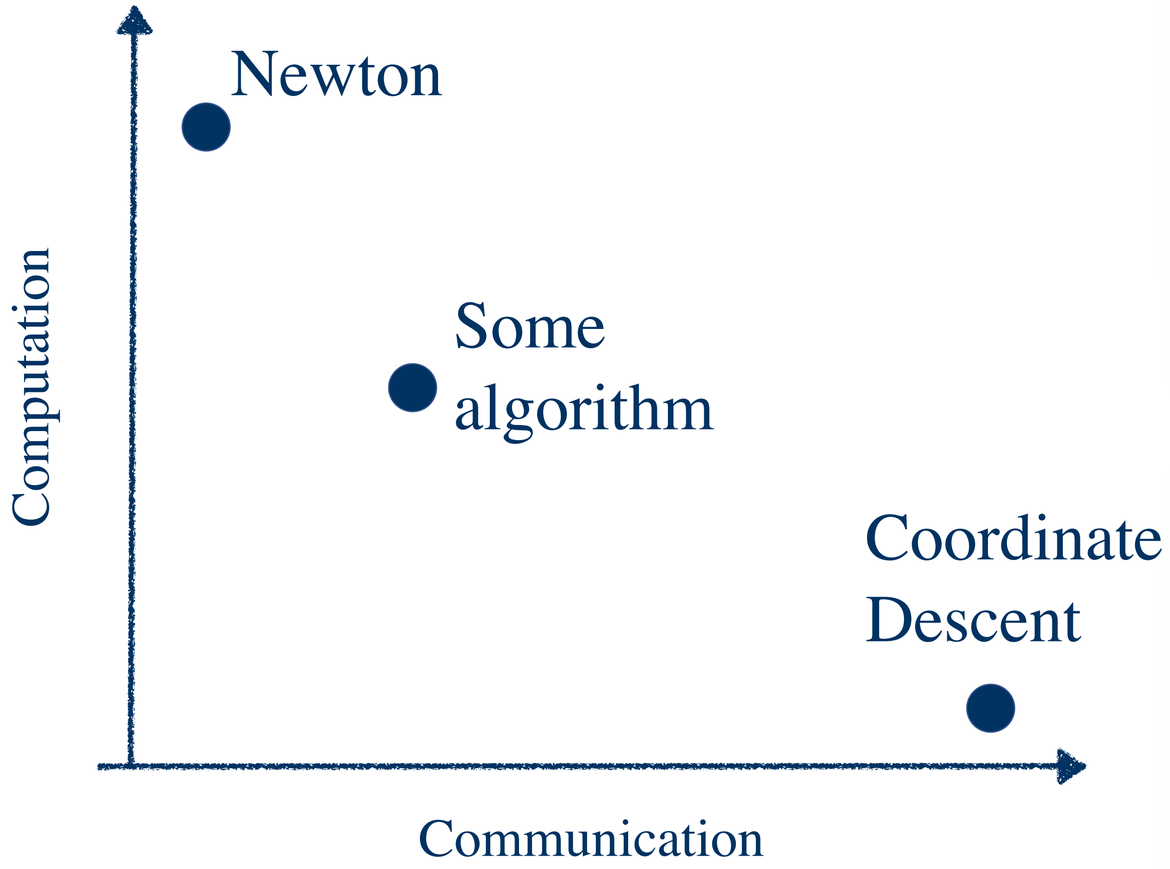
Trade-offs and existing approaches



Current approach:

choose an algorithm based on computation and communication trade-off

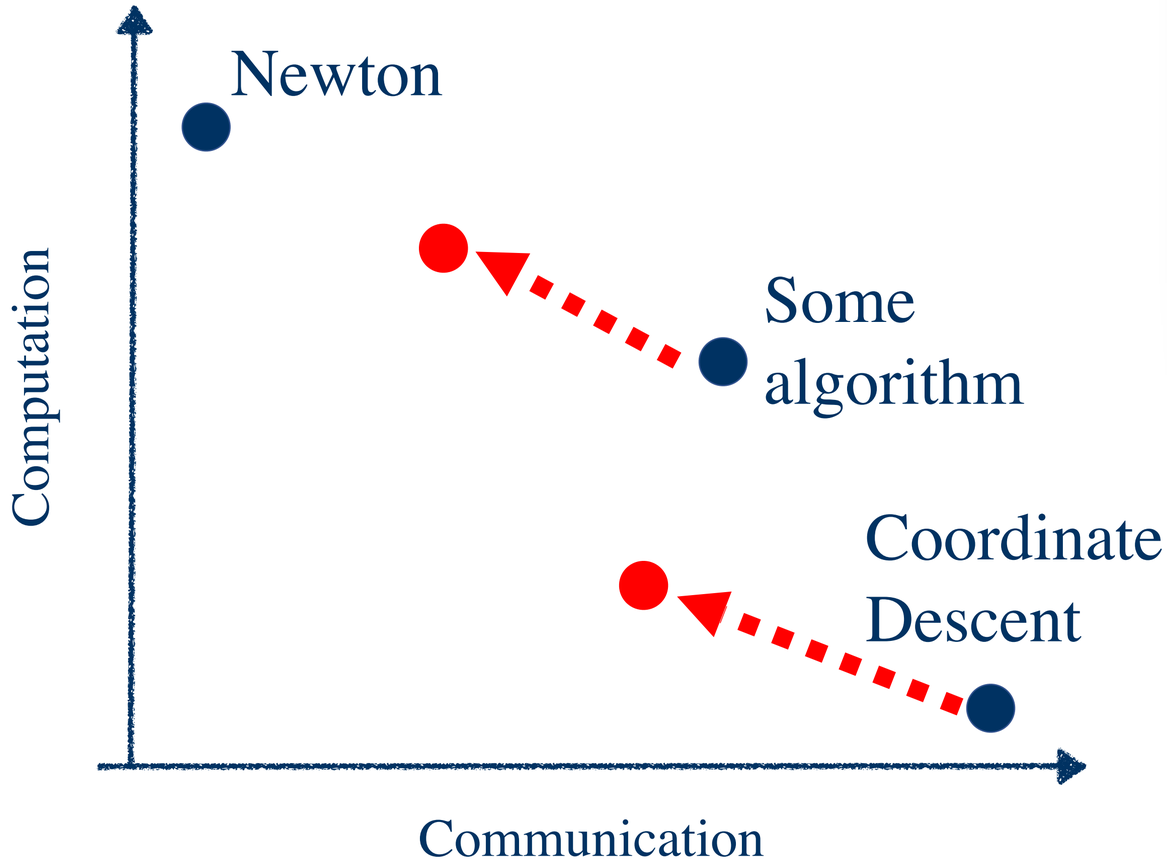
Trade-offs and existing approaches



What happens if there is no algorithm with the required trade-off?

We need to wait until a mathematician comes up with a solution

Our approach



Take existing algorithms and make them communication avoiding

Outline of the approach and results

Choose your favorite algorithm

Re-organize it to make it communication avoiding

=

Load balanced processors

Scalability to 1000+ of processors or more

For what problems?

Optimization/ML

$$\text{minimize } \lambda g(x) + \frac{1}{2}\|Ax - b\|_2^2$$

- Sparse regression $g(x) = \|x\|_1$
- Elastic net $g(x) = \frac{\eta}{2}\|x\|_2^2 + (1-\eta)\|x\|_1$
- Group lasso $g(x) = \sum_{j=1}^{J} \|x_j\|_{K_j}$
- Sparse group lasso

Linear Regression

$\text{minimize } \|Ax - b\|_2^2$

An example: coordinate descent

Pseudo-code

1 communication per iteration

- Sample a column of data

- Compute partial derivative

- Update solution

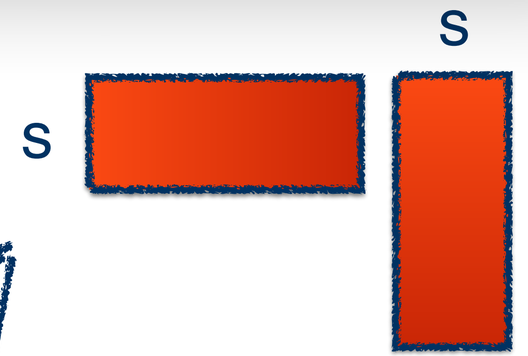
- Repeat

An example: communication avoiding coordinate descent

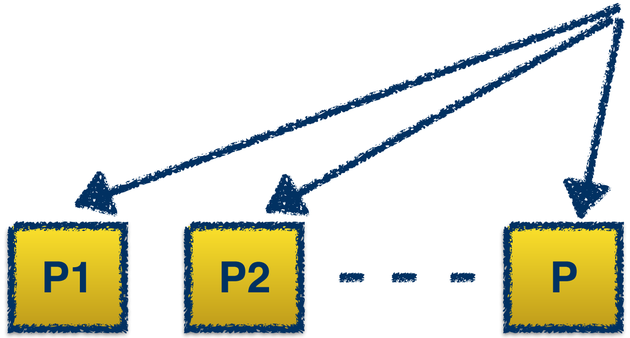
Pseudo-code

1 communication round per s iterations

- Compute in parallel anticipated computations for the next “s” iterations



- Redundantly store the result in all processors



- Each processor independently computes the next “s” iterations
- Repeat

More details about the results

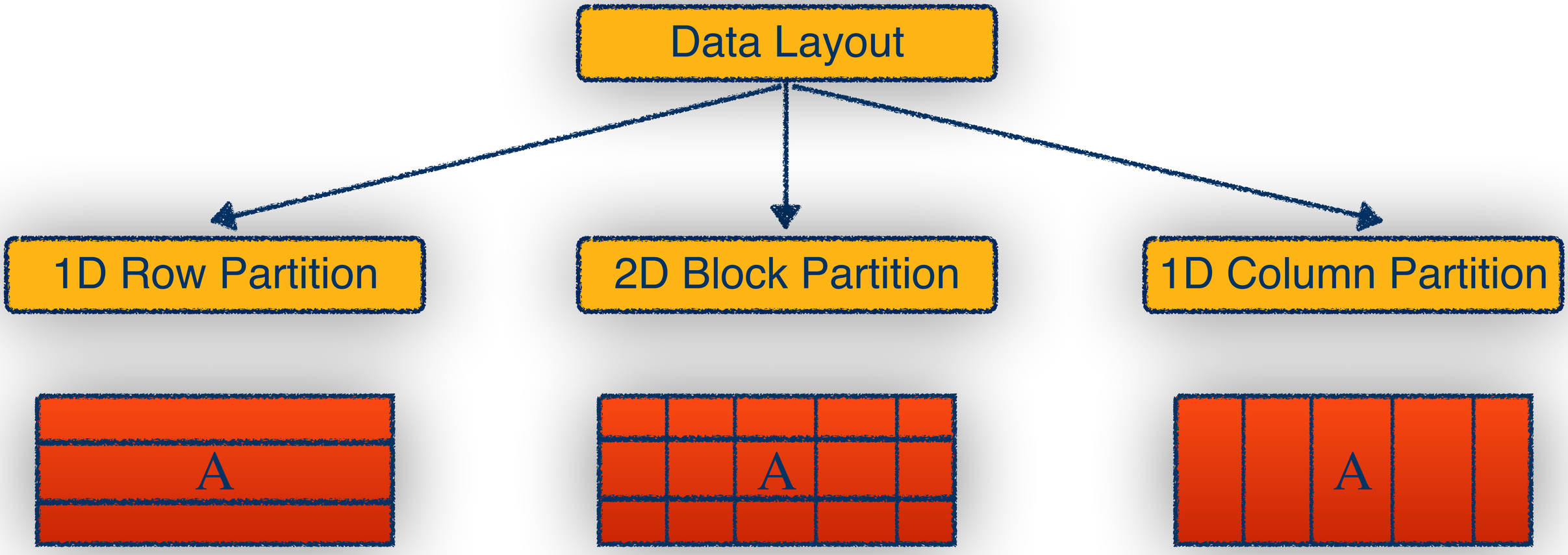
Decrease communication by a factor of s

No free lunch: increase message size and flops by a factor of s

Flops are distributed across processors

Logarithmic dependence of communication cost **on number of processors**

Scalable results for all data layouts



* Best performance depends on dataset and algorithm

Other examples

- Block coordinate descent
- Accelerated block coordinate descent
- Gradient descent
- Any proximal method

Datasets

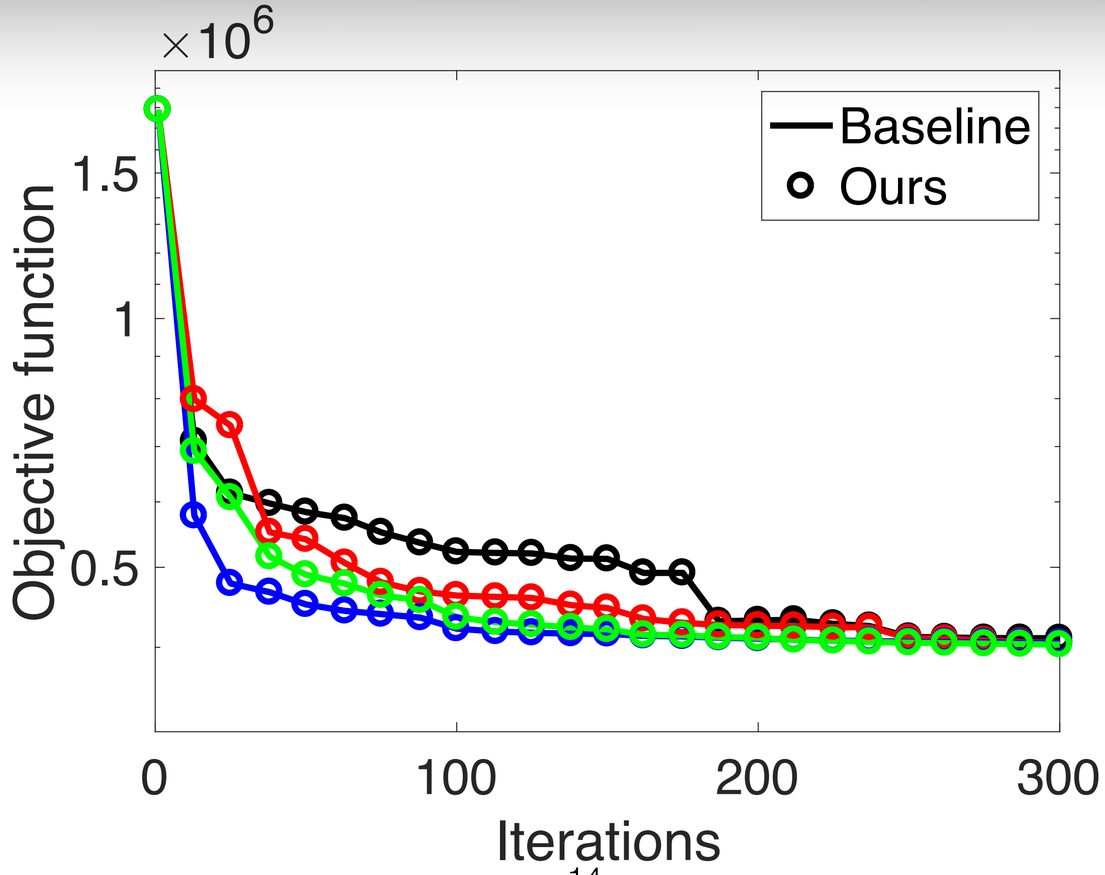
Summary of (LIBSVM) datasets

Name	#Features	#Data points	Density of non-zeros
url	3,231,961	2,396,130	0.0036%
epsilon	2,000	400,000	100%
news20	62,021	15,935	0.13%
covtype	54	581,012	22%

C++ using the Message Passing Interface (MPI). Intel MKL library for sparse and dense BLAS routines. All methods were tested on a Cray XC30.

Convergence of re-organized algorithms

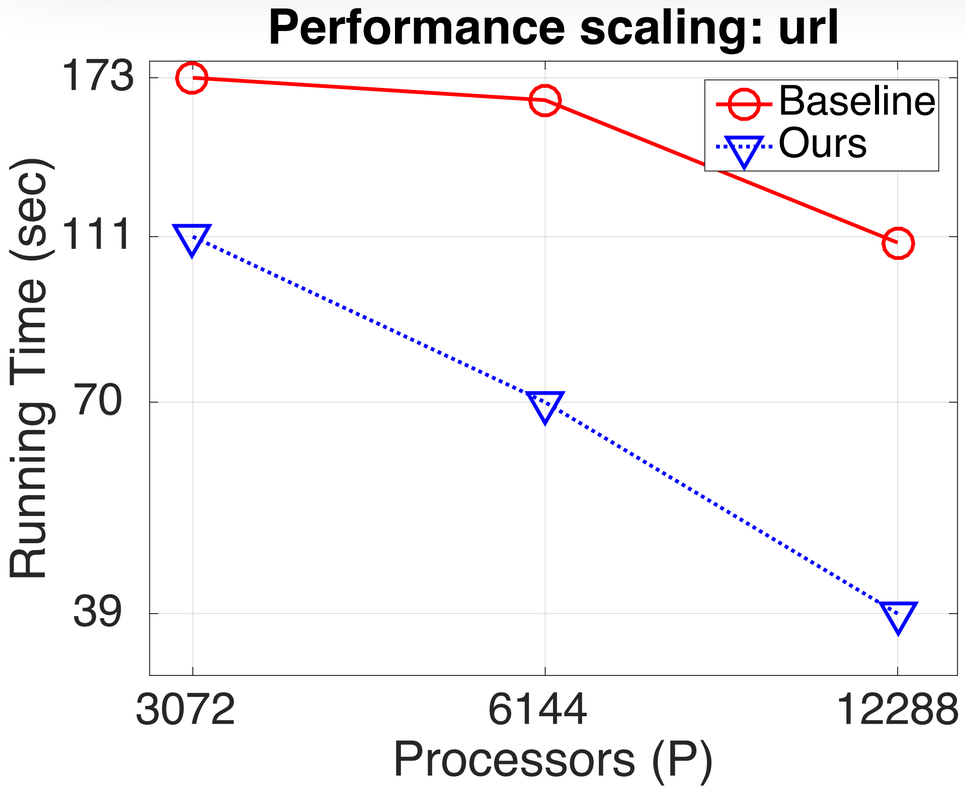
Convergence rate remains the same in exact arithmetic
Empirically stable convergence: no divergence between methods

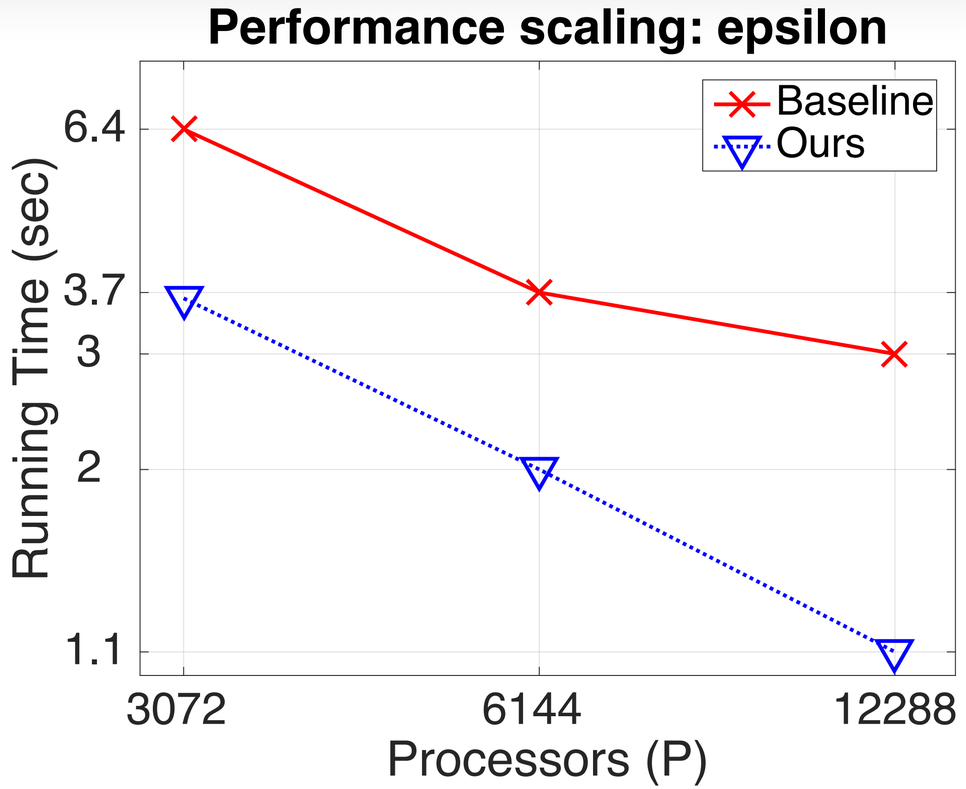


Scalability performance

The more processors the better

The gap between CA and non-CA increases w.r.t. #processors

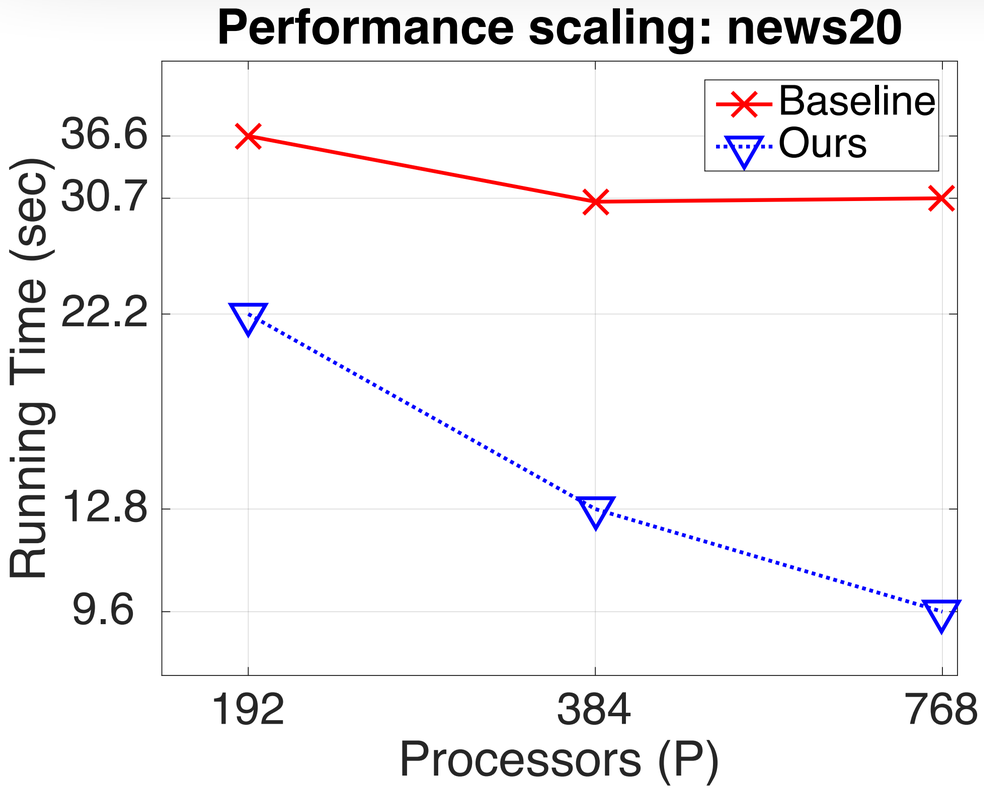


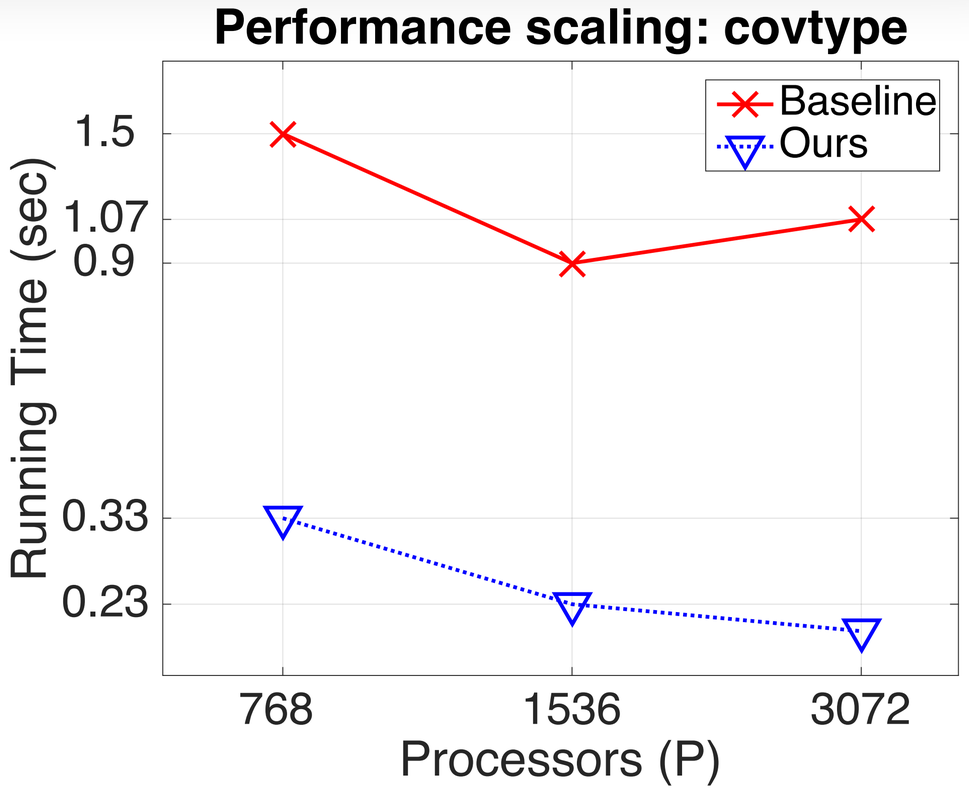


Scalability performance

The more processors the better

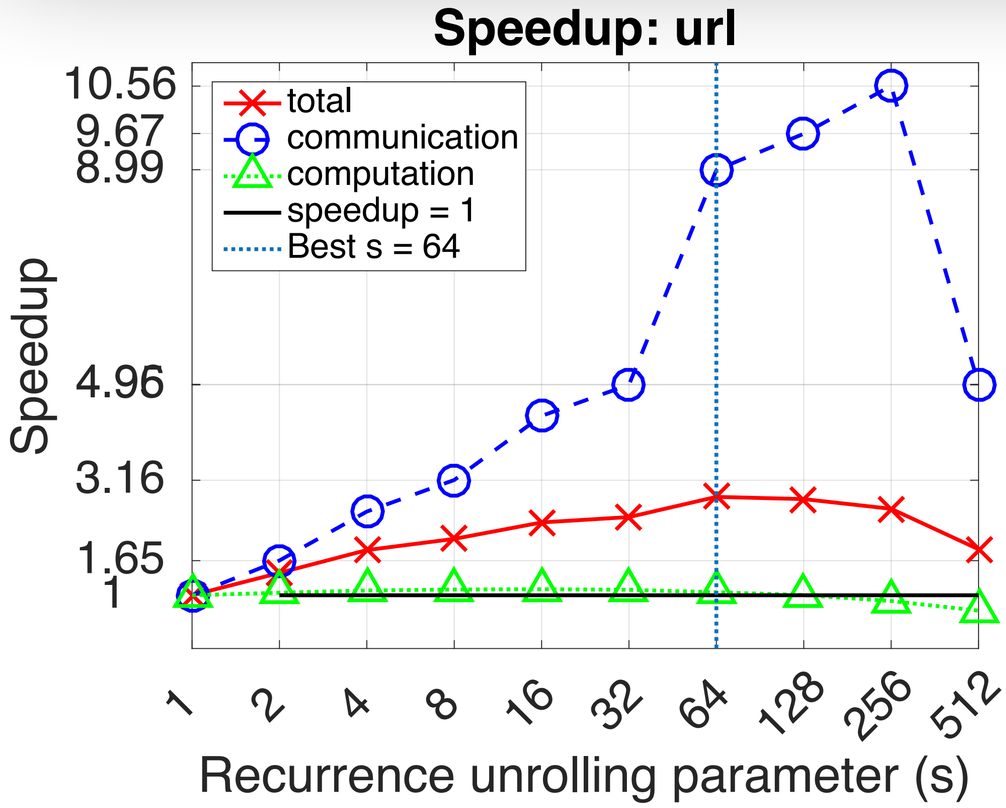
The gap between CA and non-CA increases w.r.t. #processors

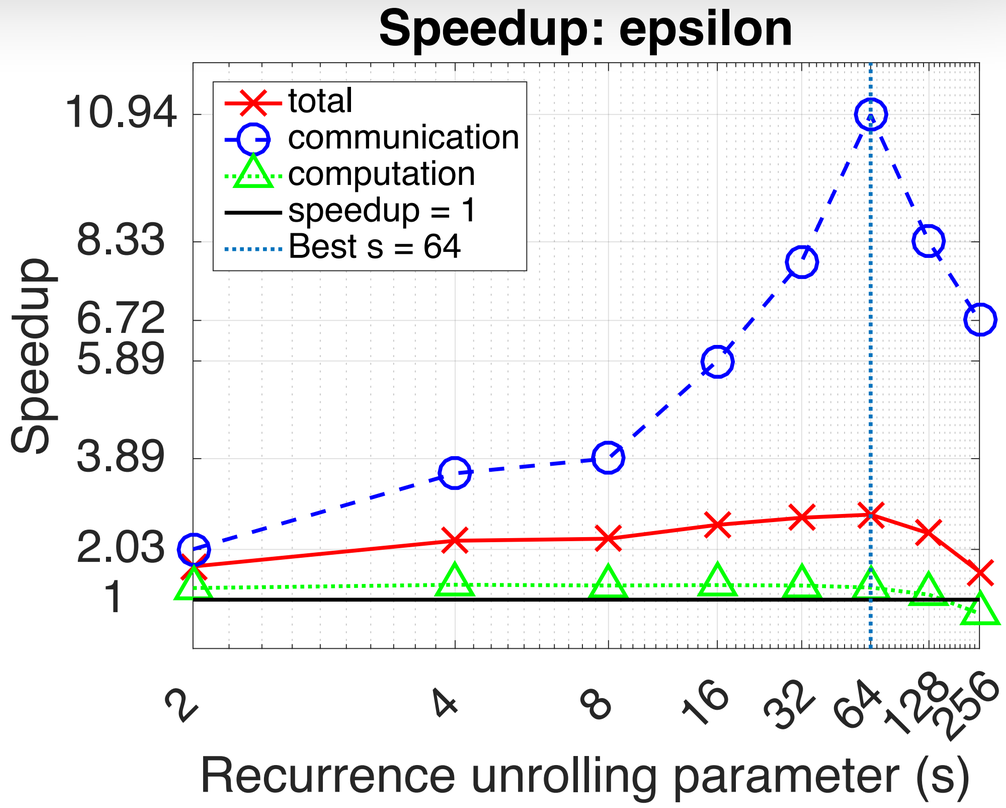




Speed up breakdown

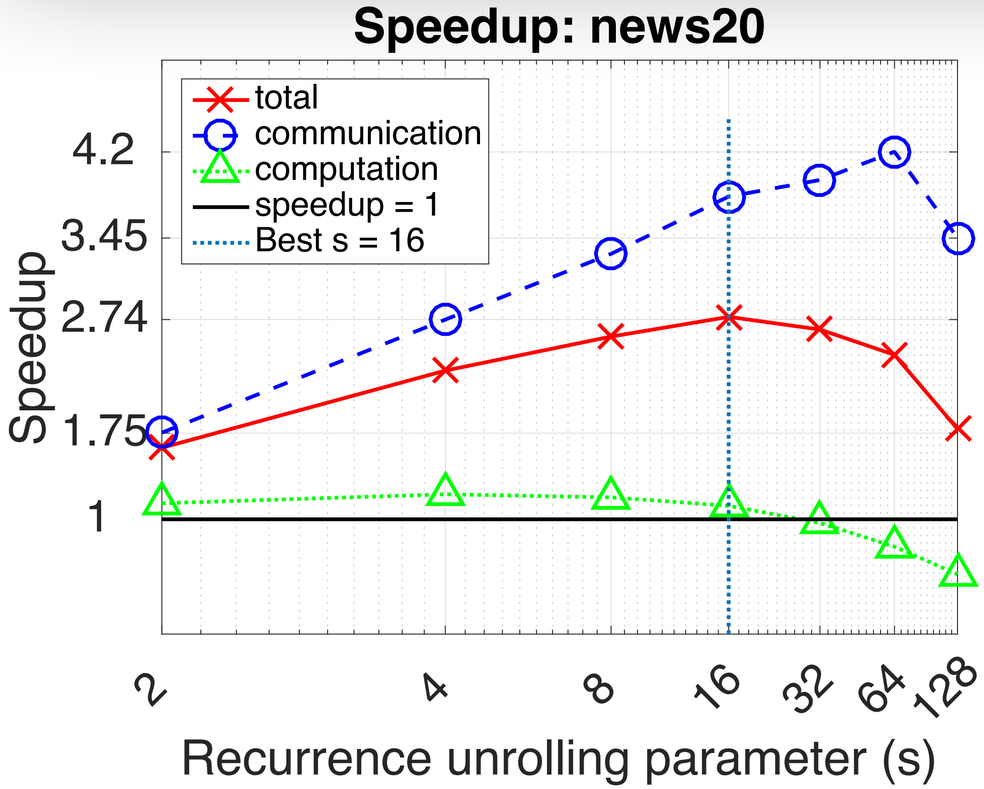
Large communication speedup until bandwidth takes a hit
Computation is maintained due to local cache-efficient (BLAS-3) computations

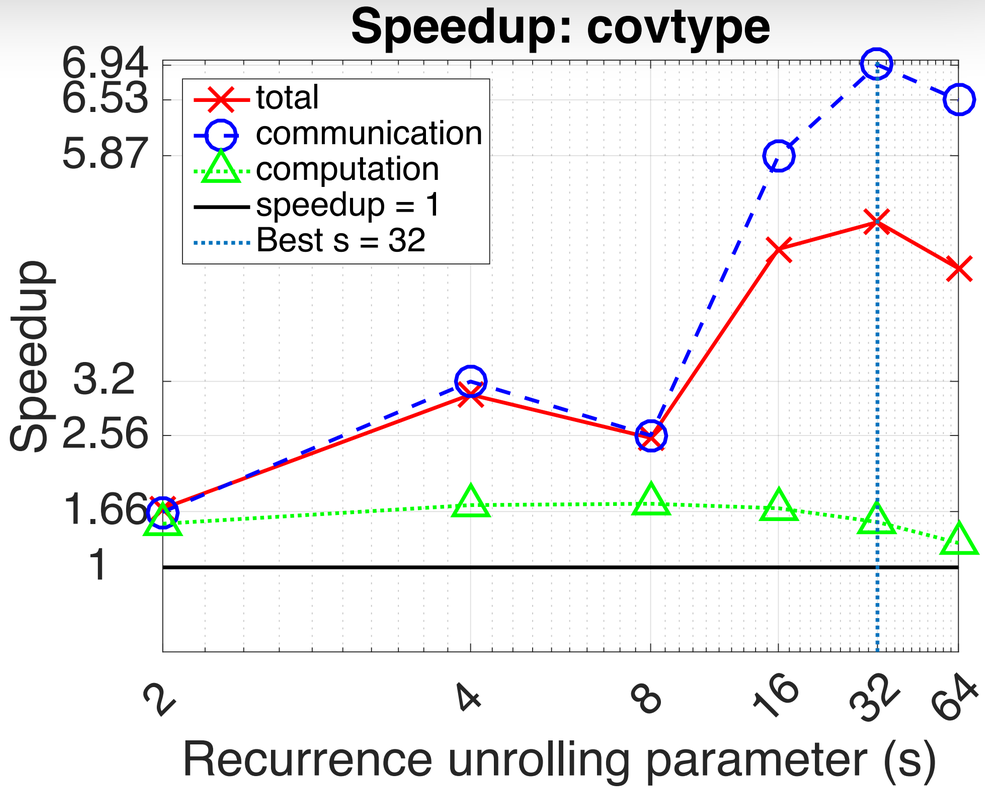




Speed up breakdown

Large communication speedup until bandwidth takes a hit
Computation is maintained due to local cache-efficient (BLAS-3) computations





Summary

Generalize from linear algebra to optimization/ML

Provably avoid communication

Scalability to 10,000+ processors

Applies to many algorithms

Overview

Linear Algebra in Spark for science problems

- CX and SVD/PCA implementations and performance
- Applications of the CX and PCA matrix decompositions
- To mass spec imaging, climate science, etc.

The Next Step: Alchemist

- Combining Spark and MPI

Communication-avoiding LA/ML

- Going beyond CA-LA to CA-ML