Multi-Block ADMM and its Applications

WORKSHOP OF MODERN TECHNIQUES OF VERY LARGE SCALE OPTIMIZATION

MAY 19, 2022

Yinyu Ye

Stanford University

Today's talk

- **Introduction to Multi-Block ADMM and ABIP**
- **RAC-ADMM implementation for Mixed-Integer Quadratic Programming**
- **Benefit of Data Exchange in Multi-Block ADMM Algorithm for Regression Estimation**

Introduction to ADMM

- Consider the following convex optimization problem

$$\min \ f(\boldsymbol{x})$$
$$\text{s.t.} \quad \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$$
$$\boldsymbol{x} \in \boldsymbol{X}$$

- Where f is a convex function, and $\boldsymbol{X}$ the Cartesian product of possibly non-convex, real, closed, nonempty sets.

- The corresponding augmented Lagrangian function is

$$L(\boldsymbol{x}, \boldsymbol{\lambda})_{\boldsymbol{x} \in \boldsymbol{X}} = f(\boldsymbol{x}) - \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \frac{\rho}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$$

where $\boldsymbol{\lambda}$ is the Lagrangian multipliers or dual variables, and $\rho > 0$ is the step size.

Multi-Block Cyclic ADMM (MB-ADMM)

- We could also partition the variables into multiple blocks. Let $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_b]$

- Direct extension of multi-block (cyclic) ADMM updates as follows

$$\text{ADMM} = \begin{cases} x_1^{k+1} & = \arg\min_{x_1 \in \mathcal{X}} \mathcal{L}_\beta(x_1, \ldots, x_b^k; \lambda^k) \\ \ldots & \\ x_b^{k+1} & = \arg\min_{x_l \in \mathcal{X}} \mathcal{L}_\beta(x_1^{k+1}, \ldots, x_b; \lambda^k) \\ & \\ \lambda^{k+1} & = \lambda^k - \beta(Ax^{k+1} - b) \end{cases}$$

The two-block (Glowinski & Marrocco 1975, Gabay & Mercier 1976,…); but the three-block ADMM may not converge (Chen et al. 2016)

Variable-Splitting (or Distributed) ADMM

- Variable-Splitting ADMM introduces **auxiliaries** to each block of variables.

- Consider the following optimization problem with **separable** objective

$$\begin{aligned} \min\ & \Sigma_{i=1}^{b} f_i(\boldsymbol{x}_i) \\ \text{s.t.}\quad & \Sigma_{i=1}^{b} \boldsymbol{A}_i \boldsymbol{x}_i = \boldsymbol{b} \\ & \boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

- Primal Variable-Splitting ADMM reformulates the problem as

$$\begin{aligned} \min\ & \Sigma_{i=1}^{b} f_i(\boldsymbol{x}) \\ \text{s.t.}\quad & \Sigma_{i=1}^{b} \boldsymbol{y}_i = \boldsymbol{b} \\ & \boldsymbol{A}_i \boldsymbol{x}_i - \boldsymbol{y}_i = 0 \quad \forall i \\ & \boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

Essentially a two-block problem since each block of x_i can be updated independently and all y_i's can be updated in a closed form (Eckstein&Bertsekas 1992)

Randomly Assembled Cyclic - ADMM (RAC-ADMM)

- In each cycle, RAC-ADMM first **randomly assembles variables into blocks**, then sequentially updates each of the blocks.
- Then update the multipliers the same way.
- RAC - ADMM is guaranteed to converge in **expectation**.

Sun et al. 2015, Mihic et al. 2020…

ADMM-BASED-INTERIOR-POINT

Solve the following LP:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

- ABIP: solves the homogeneous and self-dual embedding with barrier parameter μ decreasing in each major iteration and applies a customized ADMM for solving each μ-fixed convex inner-problem due to self-dual property (Lin et al. 2021)
- Rescale the constraint matrix A to $\tilde{A} = D_1^{-1} A D_2^{-1}$ with positive diagonal matrices D_1 and D_2 to decrease the condition number of A
- Apply the fixed frequency to Restart in solving the inner-problem
- Hybrid adaptive reduction of μ to balance the progresses of outer and inner iterations

Jiang et al. in progress, 2022

ABIP - NETLIB

- Selected 105 Netlib instances

- $\epsilon = 10^{-6}$, use the direct method, 10^6 max ADMM iterations

Method	# Solved	# IPM	# ADMM	Avg.Time (s)
ABIP	65	74	265418	87.07
+ restart	66	73	133353	35.37
+ rescale	**85**	72	105898	32.52
+ hybrid μ (=ABIP3+)	82	**23**	**96579**	**31.14**

Hybrid μ : If $\mu > \epsilon$ use the aggressive strategy, otherwise use the LOQO strategy; coded in C with Matlab interface

ABIP3+ decreases both # IPM iterations and # ADMM iterations remarkably (but the second-order IPMs solve them in a half second average time).

ABIP – PageRank I

- 84 instances, generated from sparse matrix datasets: DIMACS10, Gleich, Newman and SNAP

- $\epsilon = 10^{-6}$, use the indirect method, 3000 max ADMM iterations

	# Solved	Avg.Time (s)
PDLP (Julia)	84	9.81
ABIP3+	67	12.26

PDLP: Applegate et al, Google, 2022

Instance	# nodes	PDLP (Julia)	ABIP3+
adjnoun	113	2.18	0.03
cs4	22500	3.47	1.23
usroads	129165	10.23	6.77
wiki-Vote	8298	2.41	0.53

ABIP – PageRank II

The one generated by Google code possesses a staircase incident-matrix, in which cases ABIP3+ is significantly faster than PDLP!

# nodes	PDLP (Julia)	ABIP3+
10^4	9.16	0.87
10^5	143.85	26.08
10^6	2404.70	365.84

The second-order IPM and Simplex method typically fail in solving these large-scale LP problems

Today's talk

- **Introduction to Multi-Block ADMM and ABIP**
- **RAC-ADMM implementation for Mixed Integer Quadratic Programming**
- **Benefit of Data Exchange in Multi-Block ADMM Algorithm for Regression Estimation**

RAC-ADMM Implementation for Mixed-Integer LCQP

Solving:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \tfrac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x}+\mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x}=\mathbf{b} \\ & \mathbf{l}\le\mathbf{x}\le\mathbf{u} \quad x_i\in\mathbb{Z}, x_j\in\mathbb{R},\ i=1,\ldots,d,\ j=d+1,\ldots,n \end{aligned} \qquad (1)$$

(1) Approximate solution (RACQP-M)

- Good solutions for large mix-integer quadratic problems (MIQP) found fast
- Can not guarantee provable optimal solution; feasibility often met, but can not be guaranteed

(2) Exact solution (RACQP-B)

- Branch-and-bound(cut) based solver
- Either produce an optimal solution (within mipGap) or certificate of infeasibility
- Motivation is to generate the best-possible feasible solution in a limited time

This presentation focuses on (2)

RACQP-M

Extends RAC-ADMM based continuous QP solver (RACQP) to mixed-integer problems:

- Consists of a sequence of steps that work on improving the current (or initial) solution, which is then "destroyed" so as to be possibly improved again. This solve-perturb-solve sequence is repeated until termination criteria are met
- Direct method variant:
  - Applying partial Lagrangian methodology

$$\mathbf{x}_i^{k+1} = \arg\min\{L_\beta(\mathbf{x}_1^{k+1}, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_p^k; \mathbf{y}^k) |\ \mathbf{l} \le \mathbf{x}_i \le \mathbf{u},\ \mathbf{x}_i \in \mathcal{X}_i\},$$

    where $(\mathcal{X}_i)_j = \mathbb{Z}$ or $\mathbb{R}$ for integer and continuous variables respectively
  - Found to produce high quality results in a very short time
- Projection method variant:
  - Employs convex relaxation and projections onto non-convex sets (integrality constraints)
  - Designed for very hard discrete problems (e.g. cardinality constrained and satisfiability problems, Hamiltonian cycle problems etc.)
  - When applied to solving same problems it returns inferior results in terms of the objective value and feasibility, but is orders of magnitude faster

RACQP-B

- RACQP-B is a QP-based branch-and-bound solver, in which QP relaxations of the original mixed QP are solved using RACQP to produce bounds on the value of the objective function of an optimal solution.
- The aim of the solver is to demonstrate that when solving a large QP, RACQP can be a valid alternative to the simplex-based algorithm (which is the basis for almost all modern software for MIQP). Current implementation is a proof a concept:
  - Implementation done for mixed binary problems only
  - Basic branching algorithm used:
  - Current efforts in progress:
    - Branch-and-cut algorithm development
    - Advanced search strategies

Branch-and-Bound Method

- Let a polyhedron

$$\mathcal{P} := \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\,\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\} \quad (2)$$

where $\boldsymbol{A} \in R^{m \times n}$ and $\boldsymbol{b} \in R^m$. Without loss of generality, assume that binary variables, $x_i \in \{0, 1\}$, are indexed $1, \ldots s, s \leq n$ and describe a feasible region of a MBQP with

$$\mathcal{P}^B := \mathcal{P} \cap \mathbb{Z}^s \times \mathbb{R}^{n-p} \quad (3)$$

- The mixed-binary QP is then

$$\arg\min_{\mathbf{x} \in \mathcal{P}^B} f_0 = \mathbf{x}^T \mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \quad (4)$$

with a symmetric positive semidefinite matrix $\boldsymbol{H} \in R^{n \times n}$ and vector $\boldsymbol{c} \in R^n$. The branch-and-bound algorithm finds $\boldsymbol{x}^* \in \mathcal{P}^b$, which minimizes or proves that no such solution exists, i.e. $\mathcal{P}^b \neq \emptyset$. The algorithm assumes that the primal problem is bounded

Branch-and-Bound Method : Solving the Root Node

- The algorithm starts by solving the continuous relaxation of (4), i.e. $\boldsymbol{x} \in \mathcal{P}$, in order to find a fractional solution, $\hat{x} \in R^n$ and a lower bound $f_0(\hat{\boldsymbol{x}})$. (referred to as solving the root node):
  - If $\hat{\boldsymbol{x}} \in \mathcal{P}^b$ then the solution is said to be integer feasible and $\hat{\boldsymbol{x}}$ is the optimal solution of (4).
  - Otherwise, we seek, via rounding, for an upper bound, *UB*, which is the objective value of the best possible feasible solution that we can find, $\boldsymbol{x}^*$ (referred to as the incumbent)

Next $\mathcal{P}^b$ is split into two disjoint sets, $\mathcal{P}_1^b, \mathcal{P}_2^b$, such that $\mathcal{P}_1^b \cup \mathcal{P}_1^b \cap Z^s \times R^{n-p} = \mathcal{P}^b$. Each of the sets defines a sub-problem, replacing the parent problem with its children,

$$\min_{\mathbf{x}\in\mathcal{P}^B} f_0 = \min_{i=1,2}\left(\min_{\mathcal{P}_i^B \cap \mathbb{Z}^s\times\mathbb{R}^{n-p}} f_0\right) \qquad (5)$$

A sub-problem is usually referred to as a node of a search tree connected to its parent with an edge

B-and-B Method : Solving the Children Node

- Children nodes are added to the candidate list to be processed later. Every child node on the list is solved for a fractional solution $\hat{\boldsymbol{x}}$, with one of the following outcomes:
  - The sub-problem does not have a feasible solution $\hat{\boldsymbol{x}} \in R^n$ (infeasibility of problem can be certified, see e.g., Banjac et al. 2019): discard the result
  - The sub-problem has a solution , $\hat{\boldsymbol{x}} \in R^n$ and $f_0(\hat{\boldsymbol{x}}) \geq UB$, then the feasible region of the sub-problem *I*, $\mathcal{F} \subseteq \mathcal{P}_i^b \times Z^s \times R^{n-p}$, does not contain a solution better than $\boldsymbol{x}^*$: discard the result
  - The sub-problem has a solution $\hat{\boldsymbol{x}} \in R^n$ and $f_0(\hat{\boldsymbol{x}}) < UB$: add the children of this node to the candidate list.
  - The sub-problem has a feasible solution $\hat{\boldsymbol{x}}_i \in \mathcal{F}$: if $f_0(\hat{\boldsymbol{x}}) < UB$ set $\boldsymbol{x}^* = \hat{\boldsymbol{x}}$ and $UB = f_0(\hat{\boldsymbol{x}})$, otherwise proceed to search next node.
- The process is repeated until there are no more unprocessed nodes listed in the candidate list, in which case either the last incumbent is the optimal solution, or a feasible solution to (5) does not exist

Branch-and-Bound Method : Core Elements

- Branching methods:
  - Choose the variable closest to an integer
- Search strategy
  - Depth first until first incumbent, then focus on best bound (focus on proving optimality) or on finding better feasible solutions
- Upper bounding methodology
  - The performance of the solver depends on speed at which an upper bound is recovered (recall that if $f_0(\hat{x}) < UB$ we can discard nodes and branches). Leaving the algorithm to freely run until an UB is recovered may last a long time (many nodes may need to be processed).
  - Speeding-up the algorithm by employing a primal heuristic to construct a feasible solution.
- Lower bounding methodology
  - The performance of the solver depends on the quality of lower bounds – tighter the bound, the more nodes and branches can be discarded, reducing the number of nodes that need to be processed.
  - Currently in the process of developing/implementing the concept of valid inequalities of RACQP-B

Upper Bounding Methodology: Primal Heuristics

- Applying rounding heuristics to find a feasible solution after solving the relaxation at a node:
  - Nearest-neighbor rounding
  - Randomized rounding, $\text{Prob}(\text{round}(x_i) = 1) = x_i$
  - Dependent randomized rounding:
    - Geometric rounding (Ge et al. 2007): $\sum_{i=1}^{N} x_i = 1,\ \mathbf{x} \in \{0,1\}^N$
    - Pipage random rounding (Gandhi et al. 2006):

$$\sum_{i=1}^{N} y_{i,r} \le 1, \forall r \quad \sum_{j=1}^{J} x_{j,t} \ge 1, \forall t \quad \sum_{i=1}^{P_{r,t}} y_{i,r} - x_{j,t}, \forall r, t;\ \mathbf{y} \in \{0,1\}^N,\ \mathbf{x} \in \{0,1\}^M$$

    - Additional implementations that utilize constraint structure (see e.g. Bertsimas et al. 1999)
- Followed by the optimization of the remaining continuous variables by running RACQP for a couple of iterations (root relaxation usually ran with more iterations that the rest)

Experiments : Solving Randomly Generated LCQP I

- Created random MBQP with varying dimensions n, m and number of binary variables r: Density of H is 0.1, with entries generated from the uniform distribution $U(0,1)$, the linear cost c from the normal distribution $N(0,1)$, and the constraints $A \sim N(0,1)$.
- For each set of parameters 5 random problems were constructed

n	r	m	max. runtime [s]
500	50, 100	100, 200, 300	600
1000	50, 100	300, 500, 800	600
2000	50, 100	500, 1000	600
5000	50, 100	500, 1000	1800

Table 4: MBQP parameters

- Presolve option was turned off for Gurobi. Runs limited to one thread

Experiments : Solving Randomly Generated LCQP II

n	Gurobi					RACQP-B				
	Root LB time [s]	# Solved at root*	# Solved (mipGap)	# Processed nodes	Cost per node [s]	Root LB time [s]	# Solved at root*	# Solved (mipGap)	# Processed nodes	Cost per node [s]
500	0.25	0(30)	22	8518	0.06	0.14	14(30)	20	7621	0.05
1000	5.94	0(30)	18	2277	1.83	2.17	8(30)	18	2975	0.13
2000	215.16	0(20)	0	506	2.77	9.76	12(20)	16	2537	0.28
5000	192.12	0(20)	10	591	2.95	48.52	10(20)	15	1844	0.91

* The solution with $f_0(\hat{\mathbf{x}}) < mipGap$ found by solving/rounding the root node. mipGap = 0.1

Table 5: Summary of the results. Averages given.

- Rounding scheme used by RACQP finds good incumbents very fast.
- No feasible solution found by Gurobi at root node.
- In general, RACQP-B finds solutions very fast, but for some instances Gurobi was much faster due to good cuts (valid inequalities)

Search Strategy : Finding Better Feasible Solutions

- Adapting a simple perturbation scheme from RACQP-M (“escape” from a local minima) to leave the the current branch and start looking for a better feasible solution elsewhere in the search tree.
- The scheme explores the search space with some degree of bias towards the current best solution:
  - Decide on the next neighborhood to explore
    - Choose a random number of integer variables from the current best solution and fix them to their current values, making them to appear as constants
  - Find the new initial point within the neighborhood
    - $\boldsymbol{x}_{new}^{0}$ found by perturbing values of randomly chosen components of $\boldsymbol{x}_{best}$
    - Number of variables to change : chosen from a truncated exponential distribution (experimentally found to produce good results)
    - What variables to modify : at random, or guided by a problem structure, if known.
    - How to modify : swap, exchange, permute,...
- The implementation not fully integrated with RACQP-B; work in progress

Search Strategy : Experimental Results I

- Solving the maximum bisection problem. Perturbation mechanism: swapping

Instance name	Problem size	Density (**H**)	Best known Obj. val.	Gap[1]							
				run time = 5 min		run time = 10 min		run time = 30 min		run time = 60 min	
				Gurobi	RACQP-M	Gurobi	RACQP-M	Gurobi	RACQP-M	Gurobi	RACQP-M
G63	7000	$2{\cdot}10^{-3}$	26988	-0.263	-0.007	-0.160	-0.006	-0.160	-0.005	-0.037	-0.003
G67	10000	$5{\cdot}10^{-4}$	6938	-0.272	-0.016	-0.168	-0.014	-0.004	-0.011	-0.001	-0.010
G70	10000	$3{\cdot}10^{-4}$	9581	-0.009	-0.008	-0.007	-0.007	-0.006	-0.005	-0.003	-0.004
G77	14000	$3{\cdot}10^{-4}$	9918	-0.468	-0.015	-0.468	-0.013	-0.247	-0.012	-0.095	-0.010
G81	20000	$2{\cdot}10^{-4}$	14030	-0.280	-0.017	-0.280	-0.015	-0.253	-0.014	-0.214	-0.012
Average (across all G1-G81 instances):				-0.1228	-0.0101	-0.1046	-0.0088	-0.0735	-0.0073	-0.0513	-0.0065

Table 1: Max-Bisection, GSET instances [6]. Gap between best known results and RACQP/Gurobi objective values.

[1] $gap = (f(\mathbf{x}_S^*) - f(\mathbf{x}^*))/(1 + |f(\mathbf{x}^*)|)$ where $f(\mathbf{x}^*)$ and $f(\mathbf{x}_S^*)$ are objective values of the best known solution and of a solver, respectively

Search Strategy : Experimental Results II

- Solving the quadratic assignment problem (QAP). Perturbation mechanism: swapping

Instance name	Problem size(n)	Density (H)	Best known Obj val	Gap		
				Gurobi	RACQP	
				10 min	5 min	10 min
tai100a	10000	0.96	21043560	*	0.06	0.05
tai150b	22500	0.44	498896643	*	0.19	0.18
tho150	22500	0.42	8133398	*	0.09	0.06
wil100	10000	0.88	273038	1.19	0.03	0.02

Table 2: QAPLIB [3], selected large problems.

QAPLIB benchmark results summary	Gurobi	RACQP
Num. instances opt/best found	3	18
Num. instances gap < 0.01 (excluding opt/best)	0	17
Num. instances gap < 0.1 (excluding opt/best and < 0.01)	3	70

Table 3: Number of instances = 133. Max run-time: 10 min.

* Root relaxation not done

RACQP-B : Summary and Ongoing Work

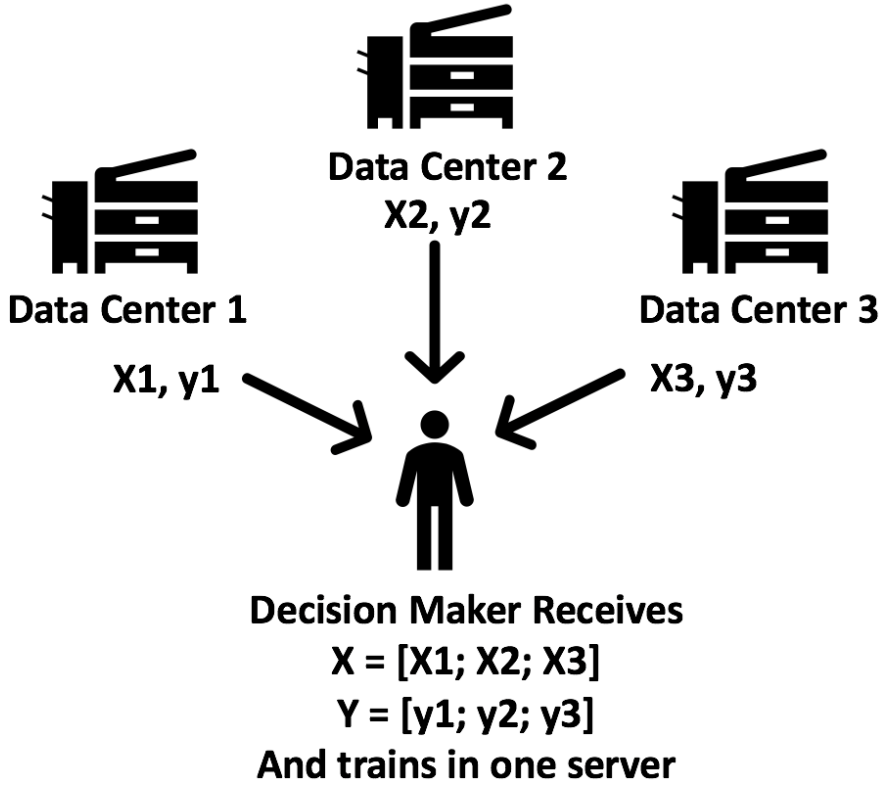
- Experiments show that:
  - Our perturbation scheme of search strategy is effective in finding new starting points, which can enable RACQP-B to leave a non-promising branch and move to another section of the search tree. We are in the process of fully integrating the scheme into the solver
  - Rounding schemes we used to generate feasible solution for UB are effective and better than of those used by Gurobi for the class of problems we addressed. We plan on adding more schemes as the use-case expands
  - RACQP-B is efficient and the cost of solving a node (relaxation + rounding) is much lower than of Gurobi, which uses dual simplex to solve the linearized QP. This can potentially offset the cost of generating valid inequalities we plan on adding to the solver (problem domain specific at first)
- Note that the current implementation is in Matlab, and planned migration to C/C++ will further improve the performance

Today's talk

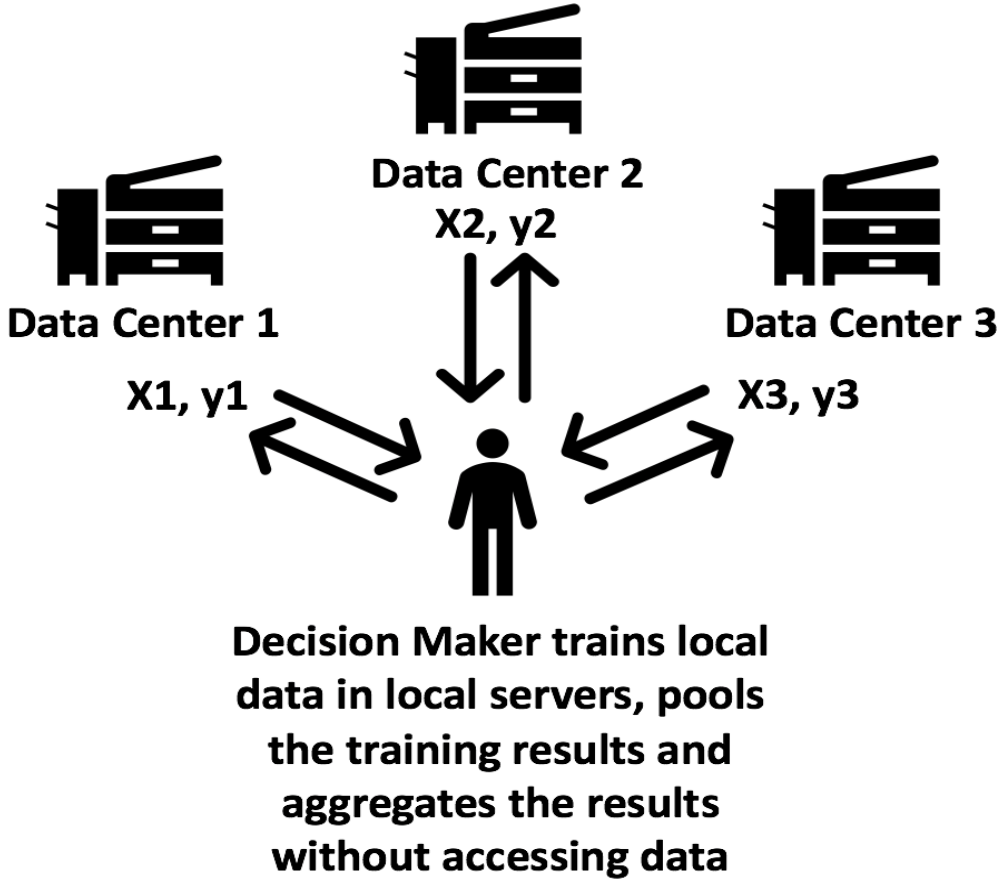
- **Introduction to Multi-Block ADMM and ABIP**
- **RAC-ADMM implementation for Mixed Integer Quadratic Programming**
- **Benefit of Data Exchange in Multi-Block ADMM Algorithm for Regression Estimation**

Statistical Learning Across Distributed Data Centers

- Centralized Learning
- All local data are uploaded to one server



- Decentralized Learning
- Local data cannot be exchanged



Statistical Learning Across Decentralized Centers

- Decentralized Learning : Methods that learn or train an algorithm across multiple **decentralized centers/blocks** holding **local data**.

- Pros: This protects **data privacy** and **data security**

- Cons: Many decentralized learning algorithms suffers from **slow convergence**.

Statistical Learning Model

- Each **center** i possess **model data matrix** $\boldsymbol{X}_i \in R^{s \times p}$ and **dependent variable** vector $\boldsymbol{y}_i \in R^{s \times 1}$.
- Let $(\boldsymbol{x}_{i,j}, y_{i,j})$ be the j^{th} data pair of the i^{th} data **center.**
- The decision maker tries to find the **global estimator** $\boldsymbol{\beta} \in R^{p \times 1}$ that minimizes the following **loss function**

$$\bullet \Sigma_{i=1}^{b} \Sigma_{j=1}^{s} f\left((\boldsymbol{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}\right)$$

where $f\left((\boldsymbol{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}\right)$ is the **loss function**.

Commonly Used Loss-Functions

- Commonly used loss function are **convex** in $\boldsymbol{\beta}$, including

  - **Least Square**

$$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x\beta} - \boldsymbol{y}||_2^2$$

  - **Ridge**

$$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x\beta} - \boldsymbol{y}||_2^2 + \alpha||\boldsymbol{\beta}||_2^2$$

  - **Lasso**

$$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x\beta} - \boldsymbol{y}||_2^2 + \alpha||\boldsymbol{\beta}||_1$$

  - **Elastic Net**

$$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x\beta} - \boldsymbol{y}||_2^2 + \alpha||\boldsymbol{\beta}||_1 + (1 - \alpha)\,||\boldsymbol{\beta}||_2^2$$

  - **Logistic**

$$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = log(1 - exp(-\boldsymbol{yx\beta}))$$

ADMM in Decentralized Learning

- The variable-splitting, or distributed, ADMM is widely used in **decentralized learning**.

- Compared with another commonly used algorithm, Stochastic Gradient Descend (SGD), consensus ADMM is more robust in **step-size** choice, and it is guaranteed to converge for any choice of step-size.

Recall Distributed MB-ADMM

- Introducing **local estimators** $\boldsymbol{\beta}_i$ to each center and reformulate the problem as

$$\sum_{i=1}^{b} \sum_{j=1}^{s} f((\mathbf{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}_i)$$

$$s.t. \quad \boldsymbol{\beta}_i - \boldsymbol{\beta} = 0 \quad \forall\, i = 1, \ldots, b$$

- Let λ_i be the dual with respect to the constraint $\boldsymbol{\beta}_i - \boldsymbol{\beta} = \mathbf{0}$, and ρ_p be the step-size to the primal distributed ADMM, the augmented Lagrangian is given by

$$L(\boldsymbol{\beta}_i, \boldsymbol{\beta}, \boldsymbol{\lambda}_i) = \sum_{i=1}^{b} \sum_{j=1}^{s} f((\mathbf{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}_i) + \sum_{i=1}^{b} \boldsymbol{\lambda}_j^T (\boldsymbol{\beta}_i - \boldsymbol{\beta}) + \sum_{i=1}^{b} \frac{\rho_p}{2} (\boldsymbol{\beta}_i - \boldsymbol{\beta})^T (\boldsymbol{\beta}_i - \boldsymbol{\beta})$$

Algorithm : Distributed ADMM

- The algorithm of primal distributed ADMM is as follows

Algorithm 1 Primal Distributed ADMM for solving (3)

Initialization: $t = 0$, step size $\rho_p \in \mathbb{R}^+$ $\beta_t \in \mathbb{R}^p$, $\lambda_{t,i} \in \mathbb{R}^p$, $\beta_{t,i} \in \mathbb{R}^p$ for all $i \in \{1, \ldots, b\}$, and stopping rule τ

while $t \leq \tau$ **do**

Each data center i updates $\beta_{t+1,i}$ in parallel by

$$\beta_{t+1,i} = \arg\min_{\beta_i \in \mathbb{R}^p} \; f((\mathbf{x}_{i,j}, y_{i,j}); \beta_i) + \lambda_{t,i}^T(\beta_i - \beta^t) + \frac{\rho_p}{2}(\beta_i - \beta_t)^T(\beta_i - \beta_t)$$

Decision maker updates

$$\beta_{t+1} = \frac{1}{b}\sum_j \beta_{t+1,i} + \frac{1}{b\rho_p}\sum_i \lambda_{t,i},\ \lambda_{t+1,i} = \lambda_{t,i} + \rho_p(\beta_{t+1,i} - \beta_{t+1})$$

end

Output: β_τ as global estimator

Distributed ADMM Suffers from Slow Convergence

- While consensus ADMM **does not exchange local data**, and enjoys benefit from **parallel computing**, it suffers from **slow convergence**.
- The following table reports the performance of GD, primal distributed ADMM, and Dual Randomly Assembled Cyclic ADMM (**DRAC-ADMM**) for L2 regression that we designed to carefully balance the trade-off between **data privacy** and **efficiency**.

Algorithms	Run Time (s)	Number of Iterations	Absolute Loss
Gradient Descend	100	9,817,048	1.71×10^{-1}
Primal Distributed ADMM	100	1,520,752	3.60×10^{-3}
DRAC-ADMM	100	4153	4.56×10^{-9}

Table 1 Algorithm Performances

Table 1 : Algorithm Performances on UCI machine learning repository (Dua and Graff(2017)) regression data YearPrediction-MSD (Chang and Lin (2011)) with number of observations n=463,715, and number of features p=90, number of local data centers = 4. Table 1 report the number of iterations, and absolute L2 loss defined by $AL = ||\boldsymbol{\beta}^* - \boldsymbol{\beta}'||_2$. For ADMM method the step-size we set equals to 1 and for GD the step-size is optimally chosen.

Outlines of Decentralized Learning: DRAC-ADMM

- Theory behind slow convergence for primal distributed ADMM
  - **Worst case** data structure of distributed ADMM for L2 regression and **Upper bound** on convergence rate of consensus ADMM
- Designing ADMM algorithm with data exchange that balance the trade-off between **privacy and efficiency**
  - Introducing Dual Randomly-Assembled Cyclic ADMM (**DRAC-ADMM**)
  - **Data exchange** is necessary – comparison with Randomly-Permuted ADMM and cyclic ADMM.
- **Numerical Results**
  - **L2 Regression**
  - **Logistic regression**

Least Square Regression

- Specifically, when $f((\boldsymbol{x}, y); \boldsymbol{\beta}) = ||\boldsymbol{x\beta} - y||_2^2$, the problem becomes a linearly constrained quadratic optimization.

$$\min_{\boldsymbol{\beta}_i, \boldsymbol{\beta}} \sum_{i=1}^{b} \frac{1}{2}(\mathbf{X}_i\boldsymbol{\beta}_i - \mathbf{y}_i)^T(\mathbf{X}_i\boldsymbol{\beta}_i - \mathbf{y}_i)$$

$$s.t.\ \boldsymbol{\beta}_i - \boldsymbol{\beta} = 0 \qquad \forall\ i$$

- Let $\boldsymbol{D}_i = \boldsymbol{X}_i^T\boldsymbol{X}_i$ and $\boldsymbol{c}_i = -\boldsymbol{X}_i^T\boldsymbol{y}_i$, primal distributed ADMM becomes

$$(\rho_p\mathbf{I} + \mathbf{D}_i)\boldsymbol{\beta}_i^{t+1} = \rho_p\boldsymbol{\beta}^t - \boldsymbol{\lambda}_i^t - \mathbf{c}_i$$

$$\boldsymbol{\beta}^{t+1} = \frac{1}{b}\sum_i \boldsymbol{\beta}_i^{t+1} + \frac{1}{b\rho p}\sum_i \boldsymbol{\lambda}_i^t$$

$$\boldsymbol{\lambda}_i^{t+1} = \boldsymbol{\lambda}_i^t + \rho_p(\boldsymbol{\beta}_i^{t+1} - \boldsymbol{\beta}^{t+1})$$

Why Data Exchange Helps Convergence I

- Consider the following example with $n = 4,\ p = 1,$ and number of data center $b = 2$. Here, two data centers share **similar structure of data**.

$$\tilde{\mathbf{X}}_1 = \begin{bmatrix} 0.99 \\ \\ 0.01 \end{bmatrix}, \quad \tilde{\mathbf{X}}_2 = \begin{bmatrix} 0.9 \\ \\ 0.1 \end{bmatrix},$$

- Normalized the model matrix $\boldsymbol{X}$ by the Frobenius norm $||\boldsymbol{X}||_F$

$$\mathbf{X}_1 = \begin{bmatrix} 0.7379 \\ \\ 0.0075 \end{bmatrix}, \quad \mathbf{X}_2 = \begin{bmatrix} 0.6708 \\ \\ 0.0745 \end{bmatrix}$$

Why Data Exchange Helps Convergence II

- The convergence rate under the previous data structure is **0.6661**, and one can show that if model matrix is normalized, with number of data centers equals to 2, the upper bound of convergence rate (worst case convergence rate) is **0.6667**.
- Convergence rate α: $$\limsup_{k\to\infty} \frac{1}{k} \log \|x_k - \mathbf{1}_N \otimes x_\star\| = \log \alpha$$
- However, if we apply **data exchange**

$$\mathbf{X}_1 = \begin{bmatrix} 0.7379 \\ 0.0075 \end{bmatrix}, \quad \mathbf{X}_2 = \begin{bmatrix} 0.6708 \\ 0.0745 \end{bmatrix}$$

- The convergence rate becomes **0.5264**, and one can show that the lower bound of convergence rate (best case of convergence rate) is **0.5000**.

Theory on Worst Case Data Structure

- Generally, under the following assumption

Assumption 1. *The regressor matrix* $\mathbf{X}$ *is normalized by its Frobenius norm* $\|\mathbf{X}\|_F$*, and the smallest and largest eigenvalue of* $\mathbf{X}^T\mathbf{X}$*,* $\underline{q}$ *and* $\bar{q}$ *are fixed, with* $\mathbf{X}_i^T\mathbf{X}_i > 0$ *for all* $i \in \{1, \ldots, b\}$.

- The worst-case data structure and the upper bound of convergence rate of distributed ADMM is given by (1)

Theorem 1. *For* $\rho_p > \bar{q}$*, the convergence rate of distributed ADMM is upper bounded by* $\frac{b\rho p}{b\rho p + \underline{q}}$*, and the upper bound is achieved when* $\mathbf{D}_i = \mathbf{D}_j$ *for all* $i, j \in \{1, \ldots, b\}$.

- Under such worst-case data structure, **any data exchange/swap** would benefit the speed of convergence for distributed ADMM.

M. Zhu & Y, “How Data Sharing Benefits Distributed Optimization and Learning,” forthcoming

Several Discussions

- The upper bound on convergence speed is increasing with respect to number of data centers and decreasing with smallest eigenvalue of covariance model matrix. This implies that **a greater number of data centers** and **ill conditioning of matrix** hurt convergence.

- Intuitively, for large step size, ADMM converges faster when each block **"differs"** from one another significantly.
  - When updating dual, ADMM takes average of all local estimators, which are essentially, the product of inverse matrix and vector. When each block differs from one another, it creates more momentum for dual updating .

- When applying **data augmentation** in machine learning with ADMM based optimization algorithm, one need to be careful as data augmentation are more likely to creates similar blocks.

Comparison Between Gradient Descend

- **Gradient Descend** is also widely used in decentralized (or **federated**) learning, with the bounds of distributed ADMM, we could compare performance between the two algorithms under different step-size.

Proposition 2. *For* $\rho_p \in (0, s_1) \cup (s_2, \infty)$, $\rho(M_p) < \rho(M_{GD})$, *where*

$$s_1 = \min\left(\frac{1}{\underline{q}} - \bar{q}, q_1\right), \quad s_2 = \frac{2b - \bar{q}\underline{q} + \sqrt{4b^2 + (\bar{q}\underline{q})^2}}{2b\bar{q}}$$

- Consensus ADMM is indeed more robust in step-size choice, and for gradient descend method, there is only a small range of sweet spot of step-size that leads to faster convergence.
- This suggests that higher order methods like ADMM could be **faster** compared with gradient descent method. What more could we do to further speed up ADMM?

Introducing Dual Randomly-Assembled Cyclic ADMM

- Inspired by Mihi ́c et al. (2020), we introduce the Dual Randomly-Assembled Cyclic ADMM (**DRAC-ADMM**)
- the least square regression problem is equivalent as

$$\min_{\zeta} \ \frac{1}{2}\zeta^T\zeta$$

$$s.t. \ \mathbf{X}\beta - \mathbf{y} = \zeta$$

- Let $\boldsymbol{t}$ be the dual variables, the dual is given by

$$\min_{\mathbf{t}} \ \frac{1}{2}\mathbf{t}^T\mathbf{t} + \mathbf{y}^T\mathbf{t}$$

$$s.t. \ \mathbf{X}^T\mathbf{t} = 0$$

- the dual variables serves as a label for each (potentially) exchanged data pair, and the randomization is more effective in the dual space.

Introducing Dual Randomly-Assembled Cyclic ADMM

Data Center 1

$(\boldsymbol{x}_{1,1}, y_{1,1}), (\boldsymbol{x}_{1,2}, y_{1,2}), (\boldsymbol{x}_{1,3}, y_{1,3});$

Local data

Data Center 2

$(\boldsymbol{x}_{2,1}, y_{2,1}), (\boldsymbol{x}_{2,2}, y_{2,2}), (\boldsymbol{x}_{2,3}, y_{2,3});$

Local data

Data Center 3

$(\boldsymbol{x}_{3,1}, y_{3,1}), (\boldsymbol{x}_{3,2}, y_{3,2}), (\boldsymbol{x}_{3,3}, y_{3,3});$

Local data

Global Data Pool

Introducing Dual Randomly-Assembled Cyclic ADMM

Data Center 1

$(\boldsymbol{x}_{1,1}, y_{1,1}), (\boldsymbol{x}_{1,2}, y_{1,2}),$

Local data

Cyclic updating

Data Center 2

$(\boldsymbol{x}_{2,1}, y_{2,1}),$ $(\boldsymbol{x}_{2,3}, y_{2,3});$

Local data

Cyclic updating

Data Center 3

$(\boldsymbol{x}_{3,2}, y_{3,2}), (\boldsymbol{x}_{3,3}, y_{3,3});$

Local data

Global Data Pool

$(\boldsymbol{x}_{1,3}, y_{1,3});$ $(\boldsymbol{x}_{2,2}, y_{2,2}),$ $(\boldsymbol{x}_{3,1}, y_{3,1}),$

Introducing Dual Randomly-Assembled Cyclic ADMM

Algorithm 2 DRAC-ADMM

Initialization: $t = 0$, step size $\rho_d \in \mathbb{R}^+$ $\mathbf{t}_t \in \mathbb{R}^n$, $\boldsymbol{\beta}_t \in \mathbb{R}^p$, and stopping rule τ

Randomly select $\alpha\%$ of total observations. Let $\mathbf{r} = [r^1, \ldots, r^m]$ be the index of selected data ($m = \lfloor \alpha\% n \rfloor$).

Let $\mathbf{r}^i = \mathbf{B}^i \cap \mathbf{r}$ be the index of selected data belongs to data center i.

while $t \leq \tau$ **do**

Random permute $\mathbf{r}$ to $\sigma_t(\mathbf{r})$, partition $\sigma_t(\mathbf{r}) = [\sigma_t^1(\mathbf{r}), \ldots, \sigma_t^b(\mathbf{r})]$ according to $|r^i|$ (size of $\mathbf{r}^i$).

For $i = 1, \ldots, b$

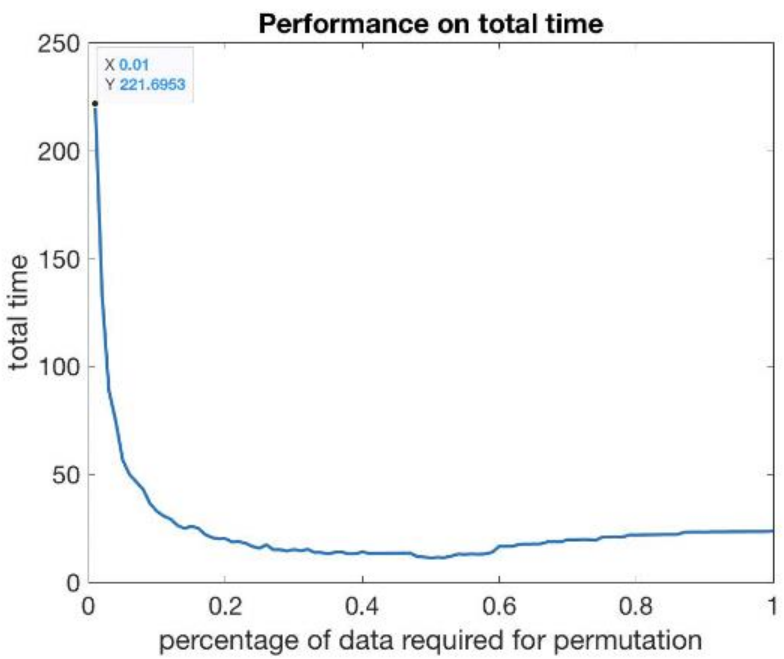
Let $\boldsymbol{\sigma}_t^i = (\mathbf{B}^i \cap \overline{\mathbf{r}^i}) \cup \sigma_t^i(\mathbf{r})$ Center i updates $\mathbf{t}_{t+1}^{\sigma_i^t} = \arg\min_{BFt \in \mathbb{R}^{|B_i|}} L(\mathbf{t}_{t+1}^{\sigma_1^t}, \ldots, \mathbf{t}_{t+1}^{\sigma_{i-1}^t}, \mathbf{t}, \mathbf{t}_t^{\sigma_{i+1}^t}, \ldots, \mathbf{t}_t^{\sigma_b^t}, \boldsymbol{\beta}^t)$

Decision maker updates $\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \rho_d \mathbf{X}^T \mathbf{t}_{t+1}$

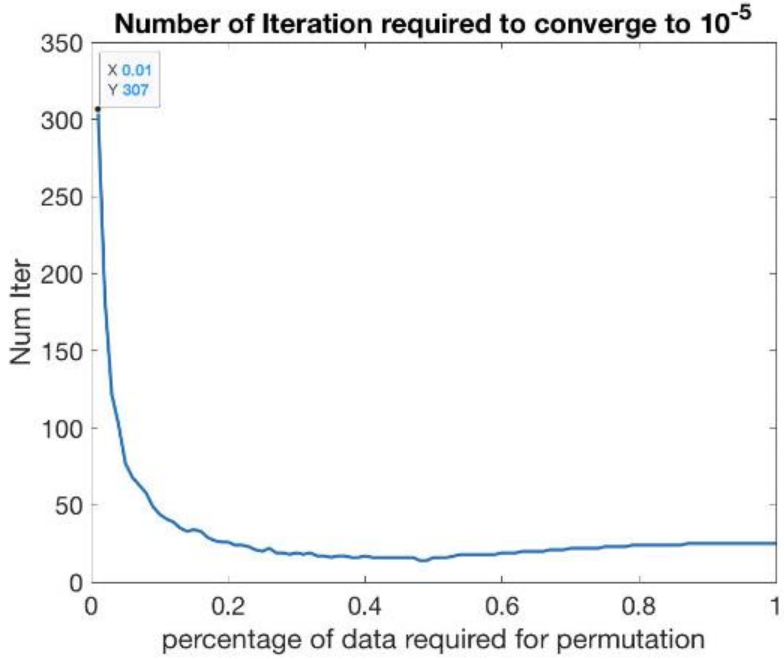
end

Output: $\boldsymbol{\beta}_\tau$ as global estimator

Trade-Off Between Privacy and Efficiency



(a) Time



(b) Number of itytaerations

Figure 1: Comparison between distributed multi-block ADMM and multi-block ADMM with data sharing. Left : Relationship between percentage of data shared and the time required for convergence. The required time for converging to the same target tolerance level with no data shared is 2403.72 seconds. Right: Relationship between percentage of data shared and the number of iterations required for convergence. The required number of iterations for converging to the same target tolerance level with no data shared is 3952. In this case, 1% of shared data provides a 10 times speed up.

UCI machine learning repository (Dua and Graff(2017)) regression data YearPrediction-MSD (Chang and Lin (2011)) with number of observations n=463,715, and number of features p=90, number of local data centers = 4.

Numerical Results

- L2 Regression
- Logistic Regression

L2 Regression: UCI ML regression repository

	Fix run time = 100 s		Fix number of iteration = 200	
	Primal distributed	DRAC-ADMM	Primal distributed	DRAC-ADMM
Bias Correction	1.60×10^{-3}	3.71×10^{-10}	3.20×10^{-3}	6.31×10^{-7}
Bike Sharing Beijing	8.43×10^{-4}	9.57×10^{-12}	2.03×10^{-2}	6.61×10^{-6}
Bike Sharing Seoul	2.60×10^{-3}	1.71×10^{-8}	8.87×10^{0}	5.80×10^{-3}
Wine Quality Red	3.45×10^{-15}	2.31×10^{-14}	8.10×10^{-3}	1.22×10^{-7}
Wine Quality White	7.36×10^{-15}	1.24×10^{-13}	2.40×10^{-3}	1.56×10^{-6}
Appliance Energy	5.02×10^{-12}	1.61×10^{-9}	7.56×10^{-1}	4.77×10^{-5}
Online News Popularity *	9.42×10^{-16}	3.23×10^{-15}	7.70×10^{-4}	4.63×10^{-8}
Portugal 2019 Election *	3.97×10^{-16}	4.97×10^{-14}	3.22×10^{-5}	1.99×10^{-10}
Relative Location of CT	1.65×10^{-13}	6.44×10^{-12}	1.29×10^{0}	4.79×10^{-4}
SEGMM GPU	2.63×10^{-13}	2.20×10^{-13}	4.60×10^{-3}	2.65×10^{-6}
Superconductivity Data	1.25×10^{-1}	2.98×10^{-6}	6.97×10^{-1}	4.99×10^{-4}
UJIIndoorLoc Data	3.76×10^{-1}	4.48×10^{-8}	8.45×10^{-1}	2.53×10^{-2}
Wave Energy Converters	3.40×10^{-3}	7.12×10^{-10}	7.70×10^{-3}	2.39×10^{-7}
Year Prediction MSD	3.60×10^{-3}	4.56×10^{-9}	3.91×10^{-2}	2.64×10^{-5}

* The covariance matrix's spectrum is of 10^{20}, which is hard for all algorithms to converge. We further scale each entry by $\sqrt{n}$.

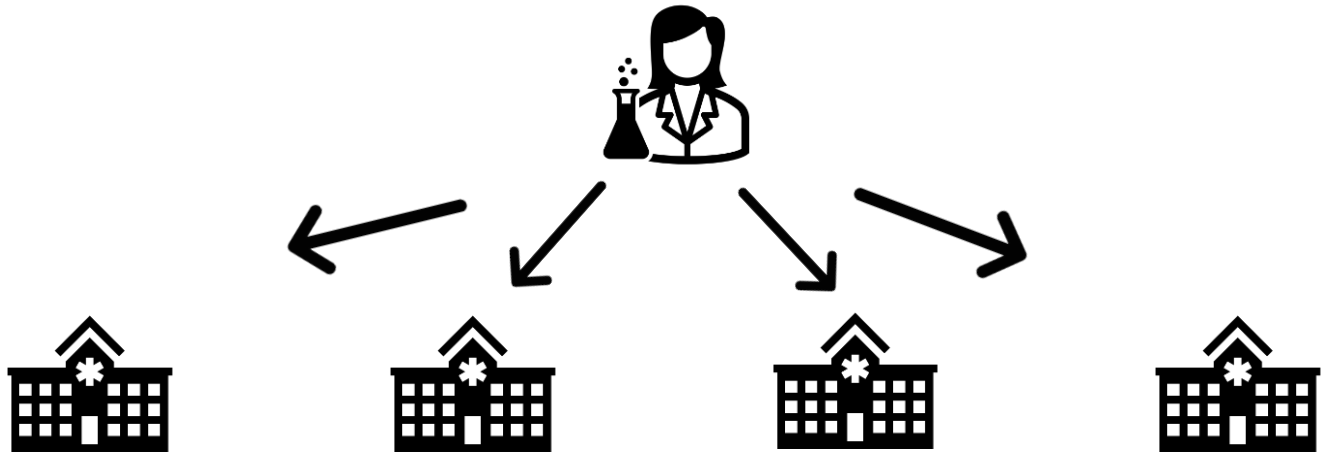
L2 Regression: UCI ML regression repository

- With 5% of access to global data, DRAC ADMM utilizes the benefit of data exchange, and outperforms primal distributed ADMM.

- Benefit of DRAC-ADMM
  - Manage to get a good quality of solution **within fewer iteration**, which further reduces the communication load across centers
  - Manage to get a good quality of solution **within a fixed time**.

Logistic Regression: Applications to Healthcare

- Real world problem
  - Researcher trying to conduct joint research at multiple hospitals



  - While researcher could operate at each data center, data could not be shared across data center – **cost per iteration is high** as the researcher would have to physically be at the data center for updating local variables per iteration

Logistic Regression: Applications to Healthcare

- p = 24; n = 2000; 4 hospitals; MAX-ITER = 50; fix time

	logreg*	Distributed ADMM	DRAC-ADMM (1%)	DRAC-ADMM (5%)	DRAC-ADMM (10%)
Objective Value	0.0215	0.0853	0.0022	8.86×10^{-4}	5.67×10^{-4}

* logreg is a two-block ADMM based algorithm, it requires access to the whole global data

Privacy-Preserved DRAC-ADMM

- Privacy would be a major concern when performing data sharing/exchange. The privacy-preserved DRAC-ADMM (PDRAC-ADMM) utilizes the random **Gaussian projection** for data required to be exchanged, which is known to be differentially private (Dwork et al. "Differential privacy and robust statistics," 2009)

- Let $\boldsymbol{R} \in R^{k \times k}$ be a square matrix with entries i.i.d. sampled from normal Gaussian, and $k = \alpha n$, where α is the percentage of global data.

Algorithms	Number of Iterations	Absolute Loss
Gradient Descent	100	38.71×10^{1}
Primal distributed ADMM	100	4.20×10^{-2}
PDRAC-ADMM	100	3.01×10^{-2}

Privacy-Preserved DRAC-ADMM

- As we add Gaussian projection to shared data, although the algorithm **protects privacy**, it **sacrifices the accuracy** of prediction.

- However, notice that PDRAC-ADMM shared similar performance with primal distributed ADMM. If data-exchange is not feasible and the Gaussian projection has to be enforced to shared data, PDRAC-ADMM could still obtain a fairly good quality of estimator.

Data Share in Distributed Preconditioned Conjugate Gradient (PCG) Method

- Data share could also be applied in **distributed conjugate gradient method** for better preconditioning

- L2 regression is same as solving

$$\sum_i \mathbf{X}_i^T \mathbf{X}_i \beta = \sum_i \mathbf{X}_i^T \mathbf{y}$$

Data Share in Distributed PCG Method I

Algorithm 1 Conjugate gradient method with local preconditioning

Initialization: $t = 0$, $\mathbf{r}_0^i = \mathbf{X}_i^T \mathbf{y}$; β_0; local pre-conditioner matrices $\mathbf{H}_i$, $\mathbf{p}_0^i = \mathbf{H}_i \mathbf{r}_0^i$ and stopping rule τ

Decision maker receives $\mathbf{r}_0^i$, $\mathbf{p}_0^i$, calculates $\mathbf{r}_0 = \sum_i \mathbf{r}_0^i$, $\mathbf{p}_0 = \sum_i \mathbf{p}_0^i$ and broadcasts it to each center

while $t \leq \tau$ **do**

Each data center calculates $\mathbf{r}_k^T \mathbf{H}_i \mathbf{r}_k$ and $\mathbf{p}_k^T \mathbf{X}_i^T \mathbf{X}_i \mathbf{p}_k$ and sends to decision maker

Decision maker updates $\alpha_k = \frac{\sum_i \mathbf{r}_k^T \mathbf{H}_i \mathbf{r}_k}{\sum_i \mathbf{p}_k^T \mathbf{X}_i^T \mathbf{X}_i \mathbf{p}_k}$ and sends to each data center

Each data center calculates $\alpha_k \mathbf{X}_i^T \mathbf{X}_i \mathbf{p}_k$ and sends to decision maker

Decision maker updates $\beta_{k+1} = \beta_k + \alpha_k \mathbf{p}_k$, $\mathbf{r}_{k+1} = \mathbf{r}_k - \sum_i \alpha_k \mathbf{X}_i^T \mathbf{X}_i \mathbf{p}_k$ and sends to each data center

Each data center calculates $\mathbf{H}_i \mathbf{r}_{k+1}$, $\mathbf{r}_{k+1}^T \mathbf{H}_i \mathbf{r}_{k+1}$ and sends to decision maker

Decision maker updates $t_k = \frac{\sum_i \mathbf{r}_k^T \mathbf{H}_i \mathbf{r}_k}{\sum_i \mathbf{r}_{k+1}^T \mathbf{H}_i \mathbf{r}_{k+1}}$ and $\mathbf{p}_{k+1} = \sum_i \mathbf{H}_i \mathbf{r}_{k+1} + t_k \beta_k$ and sends to each data center

end

Output: β_τ as global estimator

Data Share in Distributed PCG Method II

- Preconditioning could be **arbitrarily bad without data share**.

Proposition 1. *The condition number of* $\mathbf{HA}$ *could be arbitrarily large if we construct preconditioning matrix* $\mathbf{H}$ *without data share; while with data share, the condition number of* $\mathbf{HA}$ *decreases and converges to* 1 *for solving* $\mathbf{Ax} = \mathbf{b}$.

Zhu & Y, "How Data Sharing Benefits Distributed Optimization and Learning," forthcoming

Data Share in distributed PCG method III

- Idea : if the **distribution of data are different** across centers, local preconditioning give **biased estimate of Hessian** matrix
- Consider

$$\mathbf{X}_1 = [\mathbf{x}_1^1; \ldots; \mathbf{x}_i^1; \ldots; \mathbf{x}_b^1] \text{ with } \mathbf{x}_i^1 = \frac{1}{\sqrt{b}}(1, \xi_i)$$

$$\mathbf{X}_2 = [\mathbf{x}_1^2; \ldots; \mathbf{x}_j^2; \ldots; \mathbf{x}_b^2] \text{ with } \mathbf{x}_j^2 = \frac{1}{\sqrt{b}}(1, \xi_j).$$

ξ_i and ξ_j are i.i.d. Gaussian random variables $\epsilon_1 N(0,1)$ and $\epsilon_2 N(0,1)$.

- Let $\varepsilon = \varepsilon_1 = 1/\varepsilon_2$. When each data center has different distributions of data, one can show that the condition number of matrix without preconditioning is $\frac{1+\varepsilon^2}{2\varepsilon}$, the condition number of matrix with local preconditioning is $\frac{(1+\varepsilon^2)^2}{4\varepsilon^2}$, when ε is small, the matrix condition number could be arbitrarily bad

Data Share in Distributed PCG Method IV

- If we have access to sample of data from each center X_{σ_i}, we could construct the preconditioning matrix as

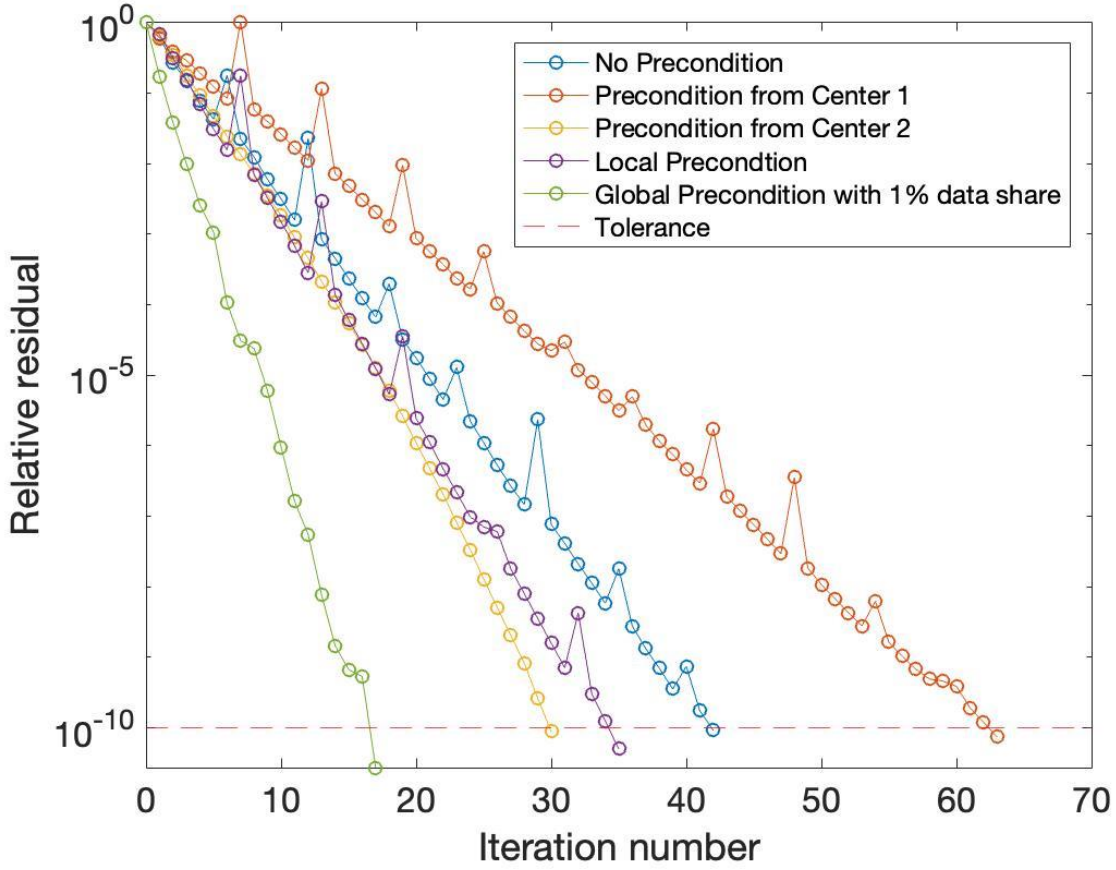
$$\mathbf{H}_i = \frac{1}{b}\mathbf{X}_i^T\mathbf{X}_i + \sum_{j\neq i}\frac{s}{r_i}\mathbf{X}_{\sigma_i}^T\mathbf{X}_{\sigma_i}$$

- Intuitively, we use the **sampled data** to sketch the data from other centers.

- With data share PCG, the condition number of matrix **converges to 1** under previous example

Data Share in Distributed PCG Method V

Numerical evidence on showing data share helps PCG method.



Setup : 2 data centers, each data center has number of observations n = 1000, feature dimensionality p = 500;
Center 1 data i.i.d. follows gaussian distributed, center 2 data follows i.i.d. uniform distribution

Thank you!

COMMENTS AND QUESTIONS?
KMIHIC@ALUMNI.STANFORD.EDU
MINGXIZ@STANFORD.EDU
YYYE@STANFORD.EDU