ARock: an algorithm framework for asynchronous parallel coordinate updates

Zhimin Peng[†], Yangyang Xu[‡], Ming Yan[†], **Wotao Yin**[†]

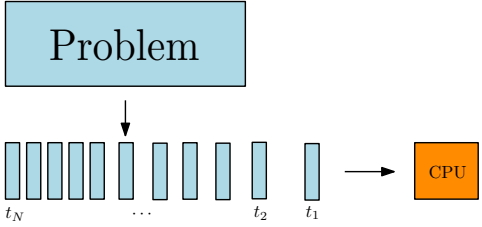
([†]UCLA Math, [‡]U.Waterloo DCO)

UCLA CAM Report 15-37

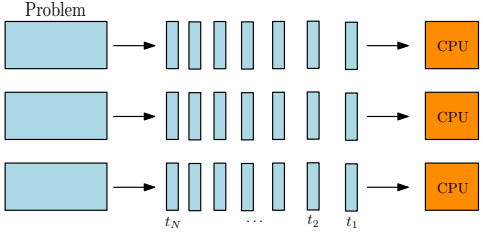
June 24, 2015

Background

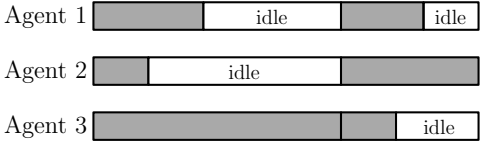
Serial computing



Parallel computing

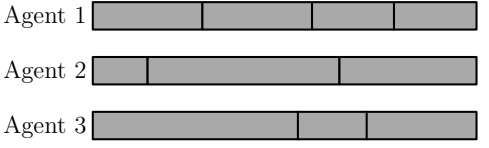


Sync-parallel versus async-parallel



Synchronous

(new iteration starts after the last agent finishes)



Asynchronous

(all agents are non-stop)

ARock:
an algorithmic framework of async-parallel coordinate updates

The fixed-point problem

- Hilbert space $\mathcal{H}$. Operator $T : \mathcal{H} \to \mathcal{H}$.
- find $x \in \mathcal{H}$ such that

$$x = Tx$$

- equivalent problem: let $S := I - T$; find $x \in \mathcal{H}$ such that

$$0 = Sx$$

- abstracts many problems:
  - convex optimization;
  - statistical regression;
  - optimal control;
  - linear and nonlinear systems of equations;
  - ordinary and partial differential equations.

Krasnosel'skiĭ–Mann (KM) iteration

- **require**: nonexpansive operator T, that is

$$\|Tx - Ty\| \leq \|x - y\|, \quad \forall x, y \in \mathcal{H}$$

- **iteration**:

$$x^{k+1} = (1 - \lambda)x^k + \lambda Tx^k$$

- **equivalent form** with $S = I - T$:

$$x^{k+1} = x^k - \lambda Sx^k$$

- **special cases:** gradient descent, proximal-point algorithm, many operator-splitting algorithms such as Douglas-Rachford and ADMM

Parallel coordinate update

- suppose $\mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_m$
- totally m agents (workstations, CPUs, cores)
- agents i update $x_i \in \mathcal{H}_i$ in parallel:

$$\begin{array}{ll} \text{agent } 1\text{:} & \\ \text{agent } 2\text{:} & \\ \vdots & \\ \text{agent } m\text{:} & \end{array} \quad \begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \\ \vdots \\ x_m^{k+1} \end{bmatrix} = \begin{bmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_m^k \end{bmatrix} - \eta_k \begin{bmatrix} (Sx^k)_1 \\ (Sx^k)_2 \\ \vdots \\ (Sx^k)_m \end{bmatrix}$$

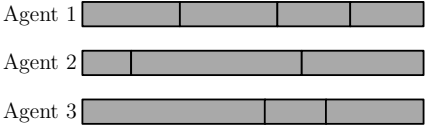
- **require**: each $(Sx)_i$ is much easier to compute than Sx (otherwise, parallel computing does not save time)

ARock: Async-parallel coordinate KM

- suppose $\mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_m$
- totally p agents; each agent randomly picks $i \in \{1, \ldots, m\}$ and updates x_i:

$$\begin{bmatrix} x_1^{k+1} \\ \vdots \\ x_i^{k+1} \\ \vdots \\ x_m^{k+1} \end{bmatrix} = \begin{bmatrix} x_1^{k} \\ \vdots \\ x_i^{k} \\ \vdots \\ x_m^{k} \end{bmatrix} - \begin{bmatrix} 0 \\ \vdots \\ \eta_k (S\hat{x}^k)_i \\ \vdots \\ 0 \end{bmatrix}$$

- $\hat{x}^k$: the result of reading x from global memory
- x^k: the status of x in global memory right before it is updated



Random coordinate selection

- each coordinate x_i is selected with probability p_i, where $\min p_i > 0$
- **cost of randomness**:
  - agents cannot cache data, global memory required (with exceptions)
- **benefits of randomness**:
  - enforce the update frequencies p_i (even if the agents have different speeds and the coordinates have different complexities); automatic load balance
  - breaks a pattern, often faster than the fixed cyclic order

Applications and numerical results

Linear equations (asynchronous Jacobi)

- **require**: invertible square matrix A with nonzero diagonal entries
- let D be the diagonal part of A; then

$$Ax = b \iff \underbrace{(I - D^{-1}A)x + D^{-1}b}_{Tx} = x.$$

- T is nonexpansive if $\|I - D^{-1}A\|_2 \leq 1$, i.e., A is diagonal dominating
- $x^{k+1} = Tx^k$ recovers the Jacobi algorithm

Algorithm 1: ARock for linear equations

Input : shared variables $x \in \mathbb{R}^n$, $K > 0$;

set global iteration counter $k = 0$;

while $k < K$, *every agent asynchronously and continuously* **do**

- **sample** $i \in \{1, \ldots, m\}$ uniformly at random;
- **add** $-\frac{\eta_k}{a_{ii}}(\sum_j a_{ij}\, \hat{x}_j^k - b_i)$ to shared variable x_i;
- update the global counter $k \leftarrow k + 1$;

Numerical comparison

- **problem**: solve

$$Ax = b,$$

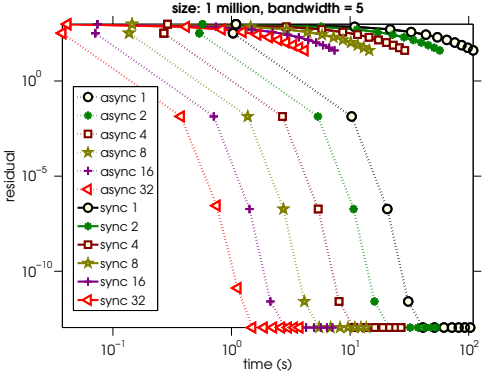
where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ are taken from the two datasets

Name	Type	Size (n)	Bandwidth (w)
Dataset I	sparse	1,000,000	5
Dataset II	dense	5,000	N/A

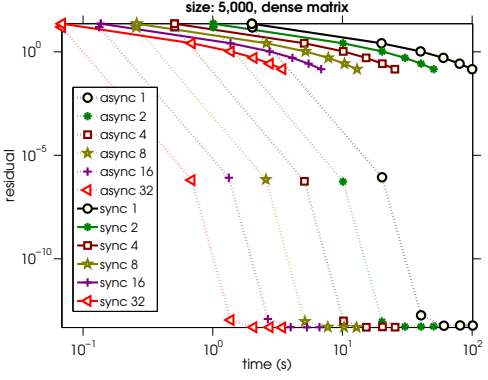
- **we compare**:
  - ARock (async)
  - Jacobi (sync)

running on $1, 2, 4, \ldots, 32$ cores on a workstation

Residual-vs-time plot



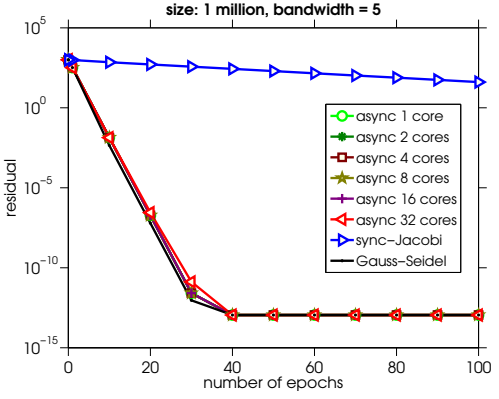
sparse A and 100 epochs



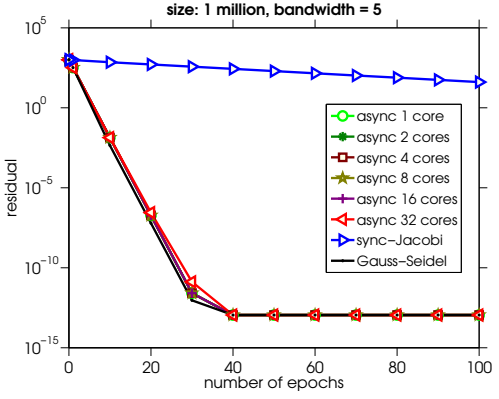
dense A and 50 epochs

- ARock (async) and Jacobi (sync) both have almost linear speedup
- ARock (async) is much faster due to asynchronicity and its Gauss-Seidel kind efficiency (next slide)

Residual-vs-epoch plot



sparse A



dense A

ARock matches Gauss-Seidel's epoch efficiency

Minimizing smooth functions

- **require**: convex and Lipschitz differentiable function f
- if ∇f is L-Lipschitz, then

$$\underset{x}{\text{minimize}}\ f(x) \iff x = \underbrace{\left(I - \frac{2}{L}\nabla f\right)}_{T} x.$$

  where T is nonexpansive

- ARock will be very faster when $\nabla_{x_i} f(x)$ is easy to compute

Minimizing composite functions

- **require**: convex smooth function g and convex (possibly nonsmooth) function f
- **proximal map**: $\mathbf{prox}_{\gamma f}(y) = \arg\min f(x) + \frac{1}{2\gamma}\|x-y\|^2$.

$$\underset{x}{\text{minimize}}\ f(x)+g(x) \iff x = \underbrace{\mathbf{prox}_{\gamma f} \circ (I - \gamma \nabla g)}_{T}\, x.$$

- ARock will be very fast given
  - easy-to-compute $\nabla_{x_i} g(x)$
  - either separable or easy-to-compute f (e.g., ℓ_1 and $\ell_{1,2}$)

Example: sparse logistic regression

- n features, N labeled samples
- each sample $x_i \in \mathbb{R}^n$ has its label $b_i \in \{1, -1\}$
- ℓ_1 regularized logistic regression:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ \lambda \|x\|_1 + \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + \exp(-b_i \cdot a_i^T x)\right), \tag{1}$$

- compare sync-parallel and ARock (async-parallel) on two datasets:

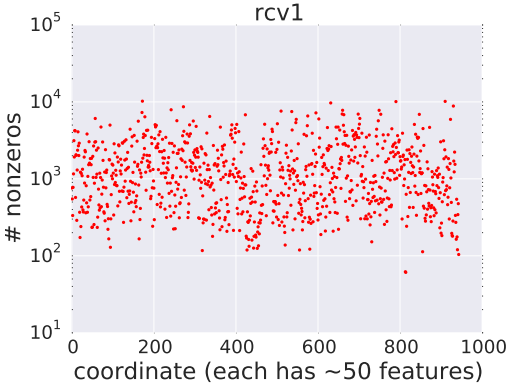
Name	N (#samples)	n (# features)	# nonzeros in $\{a_1, \ldots, a_N\}$
rcv1	20,242	47,236	1,498,952
news20	19,996	1,355,191	9,097,916

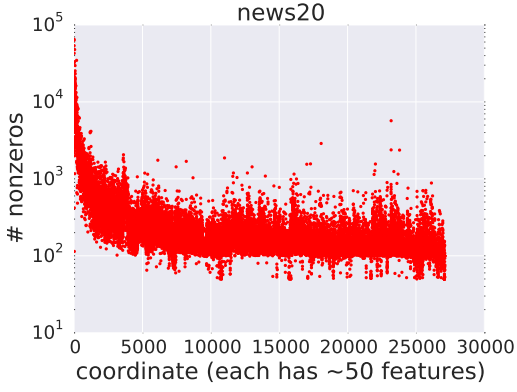
Speedup tests

#cores	rcv1				news20			
	Time (s)		Speedup		Time (s)		Speedup	
	async	sync	async	sync	async	sync	async	sync
1	122.0	122.0	1.0	1.0	591.1	591.3	1.0	1.0
2	63.4	104.1	1.9	1.2	304.2	590.1	1.9	1.0
4	32.7	83.7	3.7	1.5	150.4	557.0	3.9	1.1
8	16.8	63.4	7.3	1.9	78.3	525.1	7.5	1.1
16	9.1	45.4	13.5	2.7	41.6	493.2	14.2	1.2
32	4.9	30.3	**24.6**	**4.0**	22.6	455.2	**26.1**	**1.3**

- reasons of sync's poor speedup:
  - load imbalance (next slide): as more cores are used in parallel, it is more likely that one of them handles a coordinate corresponding to a large number of nonzeros in the samples
  - before each new iteration, all cores wait for the last core to finish
- ARock (asyn) has nearly linear speedup, not affected by load imbalance

Sparsity pattern and load imbalance





- each dot gives the # nonzeros in each coordinate (about 50 features)
- left: range of # nonzero: $10^2 - 10^4$
- right: range of # nonzero: $10^{1.8} - 10^5$
- larger ratio $\Rightarrow$ worse load balance

More applications

Minimizing composite functions

- **require**: both f and g are convex (possibly nonsmooth) functions
- **reflective proximal map**: $\mathbf{refl}_{\gamma f} := 2\mathbf{prox}_{\gamma f} - I$
- the maps $\mathbf{refl}_{\gamma f}$, $\mathbf{refl}_{\gamma g}$ and thus $\mathbf{refl}_{\gamma f} \circ \mathbf{refl}_{\gamma g}$ are nonexpansive

$$\text{minimize}\, f(x) + g(x) \iff z = \underbrace{\mathbf{refl}_{\gamma f} \circ \mathbf{refl}_{\gamma g}}_{T_{\mathrm{PRS}}}(z),\ x = \mathbf{prox}_{\gamma g}(z).$$

- T_{PRS} is known as the Peaceman-Rachford splitting operator
- also works with the Douglas-Rachford splitting operator: $\frac{1}{2} I + \frac{1}{2} T_{\mathrm{PRS}}$
- ARock will be very fast given
  - separable $\mathbf{refl}_{\gamma f}$
  - easy-to-compute $(\mathbf{refl}_{\gamma g})_i$

Parallel/distributed ADMM

- **require**: m convex functions f_i (possibly nonsmooth)
- **consensus problem**:

$$\begin{aligned} & \underset{x}{\text{minimize}} \ \textstyle\sum_{i=1}^{m} f_i(x) + g(x) \\ \iff \ & \underset{x_i, y}{\text{minimize}} \ \textstyle\sum_{i=1}^{m} f_i(x_i) + g(y) \\ & \text{subject to} \ \begin{bmatrix} I & & & \\ & I & & \\ & & \ddots & \\ & & & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} - \begin{bmatrix} I \\ I \\ \vdots \\ I \end{bmatrix} y = 0 \end{aligned}$$

- Douglas-Rachford-ARock to the dual problem $\Rightarrow$ async-parallel ADMM:
  - m f_i-subproblems are solved in the async-parallel fashion
  - y and z_i (dual variables) are updated in global memory

Algorithm 2: ARock (async-parallel ADMM) for consensus optimization

Input : shared variables $y^0, z_i^0,\ \forall i$, and $K > 0$

while $k < K$, *every agent asynchronously and continuously* **do**

- **sample** i from $\{1, ..., m\}$ with equal probability;
- **locally compute** $(\hat{w}_{d_g}^k)_i$, $\hat{x}_i^k$, and $(\hat{w}_{d_f}^k)_i$ by (2a)–(2c), respectively;
- **update global** z_i^{k+1} and $\hat{y}^{k+1}$ by (3a) and (3b), respectively;
- update the global counter $k \leftarrow k + 1$;

- **local computation**:

$$(\hat{w}_{d_g}^k)_i = \hat{z}_i^k + \gamma\, \hat{y}^k, \tag{2a}$$

$$\hat{x}_i^k = \arg\min_{x_i} f_i(x_i) - \langle 2(\hat{w}_{d_g}^k)_i - \hat{z}_i^k, x_i\rangle + \tfrac{\gamma}{2}\|x_i\|^2, \tag{2b}$$

$$(\hat{w}_{d_f}^k)_i = 2(\hat{w}_{d_g}^k)_i - \hat{z}_i^k - \gamma\, \hat{x}_i^k. \tag{2c}$$

- **global update**:

$$z_i^{k+1} = z_i^k + \eta_k\left((\hat{w}_{d_f}^k)_i - (\hat{w}_{d_g}^k)_i\right) \tag{3a}$$

$$\hat{y}^{k+1} = \hat{y}^k + \frac{1}{\gamma m}(\hat{z}_i^k - \hat{z}_i^{k+1}) \tag{3b}$$

Async-parallel decentralized ADMM

- **a graph of connected agents**: $G = (V, E)$.
- decentralized consensus optimization problem:

$$\begin{aligned} &\underset{x_i \in \mathbb{R}^d, i \in V}{\text{minimize}} \ f(\mathbf{x}) := \textstyle\sum_{i \in V} f_i(x_i) \\ &\text{subject to } x_i = x_j, \ \forall (i,j) \in E \end{aligned}$$

- **ADMM reformulation**: constraints $x_i = y_{ij}, \ x_j = y_{ij}, \quad \forall (i,j) \in E$
- apply ARock
  - version 1: nodes asynchronously activate
  - version 2: edges (and nodes of each edge) asynchronously activate
  - both versions: each agent keeps f_i private and talks to its neighbors

notation:

- $E(i)$ all edges of agent i, $E(i) = L(i) \cup R(i)$
- $L(i)$ neighbors j of agent i, $j < i$
- $R(i)$ neighbors j of agent i, $j > i$

Algorithm 3: ARock for the decentralized consensus problem

Input : each agent i sets $x_i^0 \in \mathbb{R}^d$, dual variables $z_{e,i}^0$ for $e \in E(i)$, $K > 0$.

while $k < K$, *any activated agent* i **do**

receive $\hat{z}_{li,l}^k$ from neighbors $l \in L(i)$ and $\hat{z}_{ir,r}^k$ from neighbors $r \in R(i)$;

update local $\hat{x}_i^k$, $z_{li,i}^{k+1}$ and $z_{ir,i}^{k+1}$ according to (4a)–(4c), respectively;

send $z_{li,i}^{k+1}$ to neighbors $l \in L(i)$ and $z_{ir,i}^{k+1}$ to neighbors $r \in R(i)$.

$$\hat{x}_i^k \in \arg\min_{x_i} f_i(x_i) + \big(\textstyle\sum_{l\in L(i)} \hat{z}_{li,l}^k + \sum_{r\in R(i)} \hat{z}_{ir,r}^k\big) x_i + \frac{\gamma}{2}|E(i)|\|x_i\|^2, \tag{4a}$$

$$z_{ir,i}^{k+1} = z_{ir,i}^k - \eta_k((\hat{z}_{ir,i}^k + \hat{z}_{ir,r})/2 + \gamma \hat{x}_i^k), \quad \forall r \in R(i), \tag{4b}$$

$$z_{li,i}^{k+1} = z_{li,i}^k - \eta_k((\hat{z}_{li,i}^k + \hat{z}_{li,l})/2 + \gamma \hat{x}_i^k), \quad \forall l \in L(i). \tag{4c}$$

Literature

Brief history

- The first async-parallel algorithm appeared in 1969 for solving linear equations.
- It was extended to fixed-point problems under the **absolute-contraction**[1] type of assumption.
- For 20–30 years, mainly solve linear, nonlinear and differential equations.
- Some recent work solves statistical regression, machine learning, and sensor network problems.

[1] An operator $T : \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz contractive if $|T(x) - T(y)| \leq A|x - y|$, component-wise, where $|x|$ denotes the vector with components $|x_i|$, $i = 1, \ldots, n$, and $A \in \mathbb{R}^{n \times n}$ is a matrix with a spectral radius strictly less than 1.

Recent work

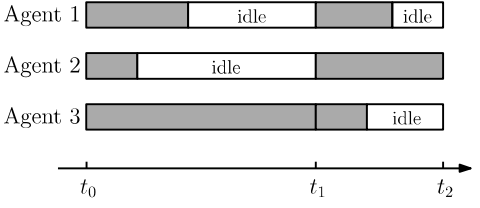
- Bertsekas-Tsitsiklis'89: Async-parallel gradient-projection method
- Liu et al.'13: async-parallel stochastic coordinate descent for minimizing convex smooth functions
- Liu and Wright'14: async-parallel stochastic proximal coordinate descent algorithm for minimizing convex composite objective functions
- Hsieh et al.'15: async-parallel implementation of LIBLINEAR (for ℓ_2 regularized empirical risk minimization)
- Other async-parallel / async-ADMM methods: Wei-Ozdaglar'13, Iutzeler et al'13, Zhang-Kwok'14, Hong'14,

ARock contributions

- A framework for nonexpansive operators that have fixed-points
- Applications: async-parallel algorithms for
  - linear equations, (smooth and nonsmooth) function minimization, distributed and decentralized optimization ...
- Similar to recent work, random coordinate updates: automatic load balance
- Analysis:
  - almost sure convergence of x^k to $x^* \in \operatorname{Fix} T$
  - linear convergence (when S is strongly monotone)
  - fixed step sizes
- Open-source C code for reproducible research

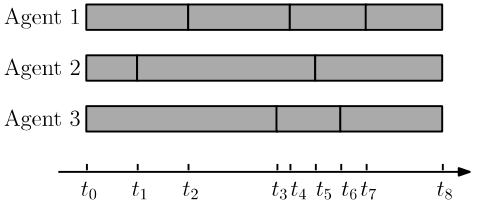
Under the hood

Iteration is redefined



Synchronous

new iteration = all agents finish



Asynchronous

new iteration = any agent finishes

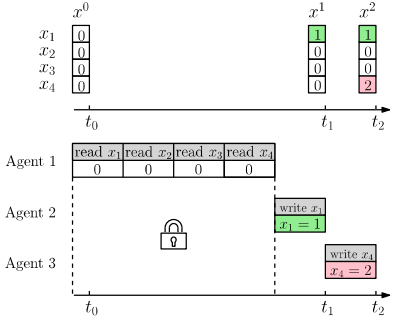
Reading consistency

- multiple agents *simultaneously* read and write x in global memory.
- while an agent reads x into its cache, x might be updated by other agents.

definitions: let $x^0, ..., x^k, ...$ be the states of x in the memory

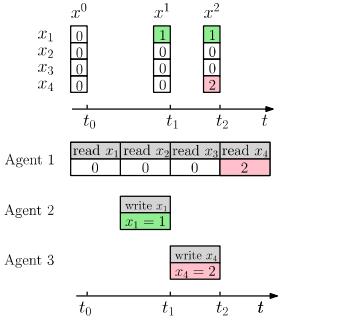
- $\hat{x}^k$ is called **consistent** if $\hat{x}^k = x^j$ for some $j \leq k$.
- $\hat{x}^k$ is called **inconsistent** if $\hat{x}^k \neq x^j$ for every $j \leq k$.

Reading consistency and memory lock



Agent 1 read $[0, 0, 0, 0]^T = x^0$

consistent read



Agent 1 read $[0, 0, 0, 2]^T \notin \{x^0, x^1, x^2\}$

inconsistent read

ARock allows inconsistent read

Atomic coordinate update

- when each coordinate update is **atomic** (single CPU instruction), the read of *each single coordinate* is consistent, that is,

$$x_i^k = \hat{x}_i^k + \underbrace{\sum_{d \in J_i(k)} (x_i^{d+1} - x_i^d)}_{\text{interim changes of } x_i}$$

  - $\hat{x}_i^k$: the result of read
  - x_i^k: the status of x_i right before it is updated
  - $J_i(k)$: the index set of the interim changes of x_i

- since k increases for each coordinate update, we have $J_i(k) \cap J_j(k) = \emptyset$. therefore, let $J(k) = \cup_{i=1}^m J_i(k)$, we have

$$x^k = \hat{x}^k + \sum_{d \in J(k)} (x^{d+1} - x^d)$$

- we assume that $|J(k)| \leq \tau$ for all k

Special cases of ARock

- if $p = m = 1$ (one agent and one coordinate), ARock reduces to the KM iteration.
- if $p = m > 1$ and $\tau = 0$ (no delay), ARock reduces to sync-parallel coordinate update.
- if $p = 1$ (only one agent), ARock reduces to Nesterov's randomized coordinate update.

Analysis challenges and techniques

Challenges

- asynchrony $\Rightarrow$ staled information used in the update
- inconsistency $\Rightarrow$ $\hat{x}^k$ may not equal a status of x ever existed
- coordinate update $\Rightarrow$ search direction only on one coordinate
- no objective function $\Rightarrow$ must play with $\|z^k - z^*\|^2$ and $\|Tz^k - z^k\|^2$

Techniques

- bounded delay or infinite delay with a light tail
- a new metric $\Rightarrow$ a non-negative almost supermartingale
- staled $\hat{x}^k$ related to the current x^k through atomic updates
- random selection: expected progress over all coordinates

Thank you!

Reference: Zhimin Peng, Yangyang Xu, Ming Yan, Wotao Yin. UCLA CAM 15-37.

Website: http://www.math.ucla.edu/~wotaoyin/arock