A Delay-tolerant Proximal-Gradient Algorithm for Distributed Learning

Konstantin Mishchenko — KAUST

Franck Iutzeler — Univ. Grenoble Alpes

Jérôme Malick — CNRS and Univ. Grenoble Alpes

Massih Amini — Univ. Grenoble Alpes

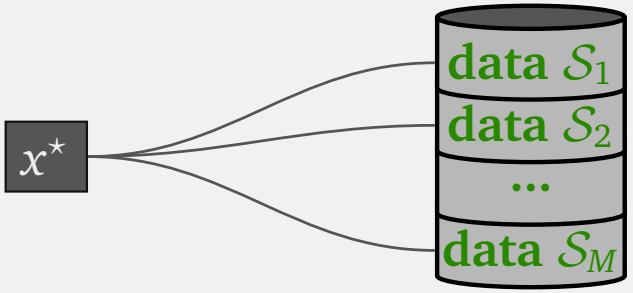
ICML 2018



Global objective

$$\min_{x\in\mathbb{R}^d} \frac{1}{m}\sum_{j=1}^{m} \ell_j(x) + g(x)$$

m examples
individual losses (ℓ_j)
emprical risk minimization
regularizer g



Local data

$$\min_{x\in\mathbb{R}^d} \sum_{i=1}^{M} \pi_i f_i(x) + g(x)$$

M data blocks
stored locally
local function (f_i)
$f_i(x) = \frac{1}{|\mathcal{S}_i|}\sum_{j\in\mathcal{S}_i} \ell_j(x)$
proportion $\pi_i = |\mathcal{S}_i|/m$ at i

Problem:	Large sum minimization	v.s.	Mid-sized distributed optimization
Optimization:	Variance-reduced sto. gradient	v.s.	this presentation

- Distributed Optimization

  Asynchronism

  Scarse Communications

  Conclusion

Problem: $\min_x \sum_{i=1}^M \pi_i f_i(x) + g(x)$ Direct extension of the prox. grad.:

Implementation Algorithm

Worker i update on local variable

$$x_i^{k+1/2} = x_i^k - \gamma \nabla f_i(x_i^k)$$

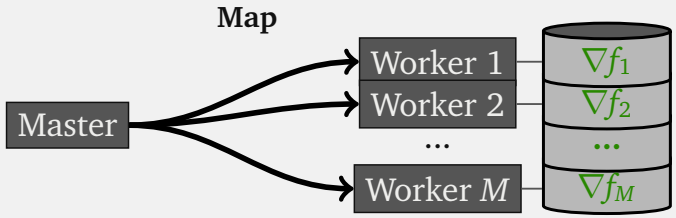
for all $i = 1, .., M$

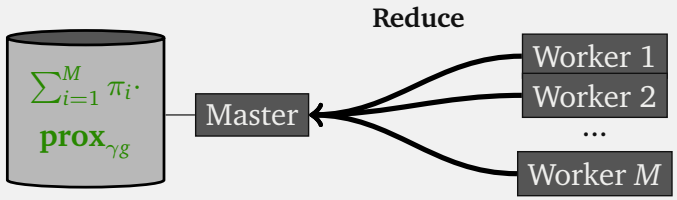
Master gathering of the local variables

$$\bar{x}^{k+1} = \sum_{i=1}^M \pi_i x_i^{k+1/2}$$

Master performs a proximity operation

$$x_1^{k+1} = .. = x_M^{k+1} = \mathbf{prox}_{\gamma g}\left(\bar{x}^{k+1}\right)$$





Distributed Proximal Gradient

Master:

Initialize $\bar{x} = \bar{x}^0$,
while *not converged* **do**
  when *all* workers have finished:
  Receive (x_i) from each of them
  $\bar{x} \leftarrow \sum_{i=1}^M \pi_i x_i$
  Broadcast $\bar{x}$ to all agents
  $k \leftarrow k + 1$
Interrupt all slaves
Output x

Worker i:

Initialize $x = x_i = \bar{x}$,
while *not interrupted by master* **do**
  Receive the most recent $\bar{x}$
  $z \leftarrow \mathbf{prox}_{\gamma g}(\bar{x})$
  $x_i \leftarrow z - \gamma \nabla f_i(z)$
  Send x_i to the master

$$f_i(x) = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \ell_j(x)$$

Distributed Proximal Gradient

Master:

Initialize $\bar{x}$
while *not converged* **do**
  when *all* workers have finished:
  Receive (x_i) from each of them
  $\bar{x} \leftarrow \sum_{i=1}^{M} \pi_i x_i$
  Broadcast $\bar{x}$ to all agents
  $k \leftarrow k+1$
Interrupt all slaves
Output x

Worker i:

Initialize $x = x_i = \bar{x}$,
while *not interrupted by master* **do**
  Receive the most recent $\bar{x}$
  $z \leftarrow \mathbf{prox}_{\gamma g}(\bar{x})$
  $x_i \leftarrow z - \gamma \nabla f_i(z)$
  Send x_i to the master

$$f_i(x) = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \ell_j(x)$$

Define time k as the number of master updates

x^k is the value of variable x at time k

Theorem

Let each f_i be L-smooth and μ-strongly convex. Then, for $\gamma \in (0, 2/(\mu+L)]$,

$$\|x^k - x^\star\|^2 \leq (1-\alpha)^k \|x^0 - x^\star\|^2$$

where $x^\star$ is the unique minimizer of the $\min_x \sum_{i=1}^{M} \pi_i f_i(x) + g(x)$ and $\alpha = 2\gamma\mu L/(\mu+L) \in (0,1]$.

Proof. It is exactly proximal gradient descent.

Synchronism: Master waits for *all* workers at each time

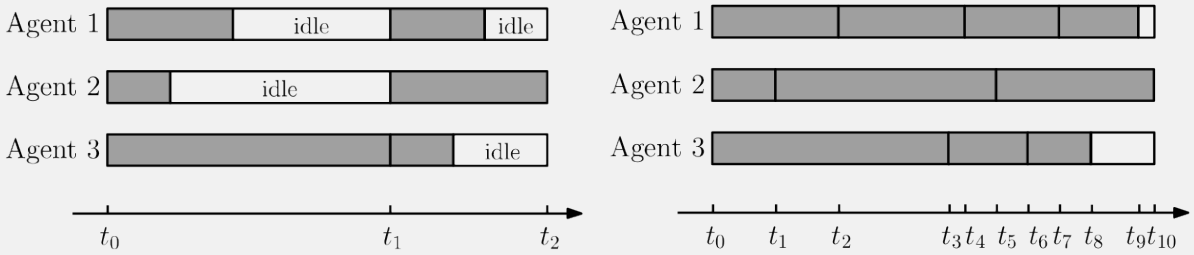


image: W. Yin

Communications: Sending may be more costly than computing a gradient



Context:
Federated Learning
Distributed Data
image: Google AI

Local updates may be:

▷ *fast* (or not, depending on $|\mathcal{S}_i|$),

▷ *late* (or not, depending on state),

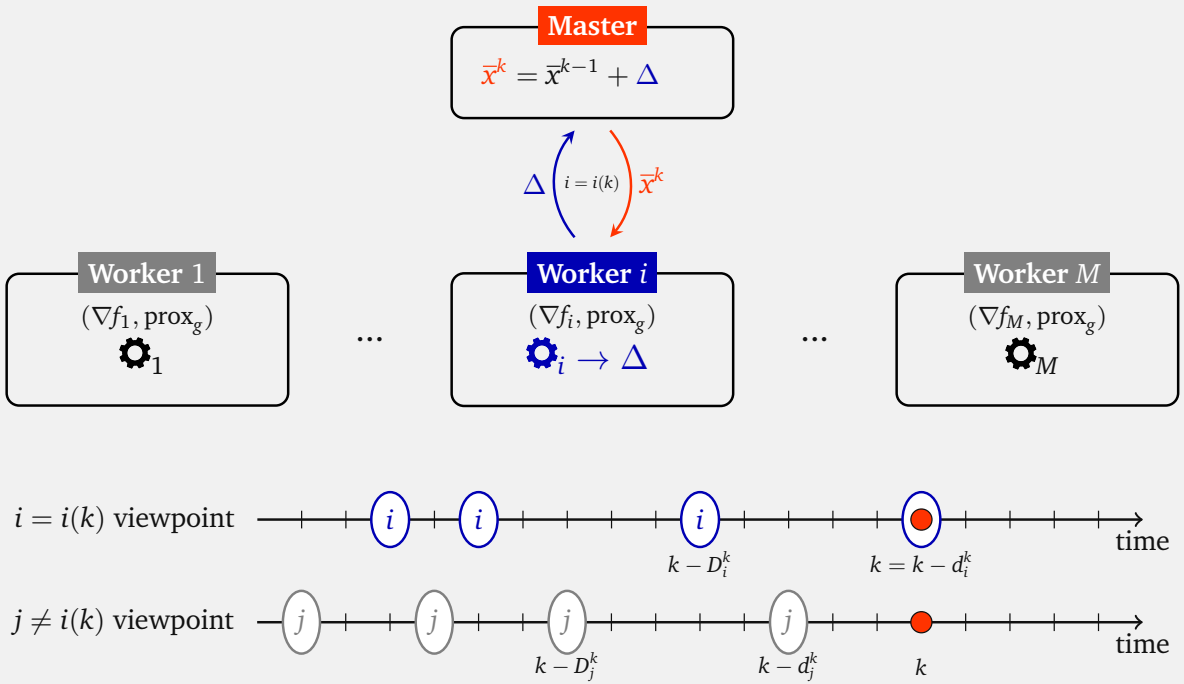
▷ *costly* (often).

We provide an *efficient* **Distributed Proximal Gradient** algorithm:

- **Asynchronous** delay-tolerant
- **Scarse** r comp./comm. tradeoff

communications

- Distributed Optimization
- **Asynchronism**
- Scarse Communications
- Conclusion

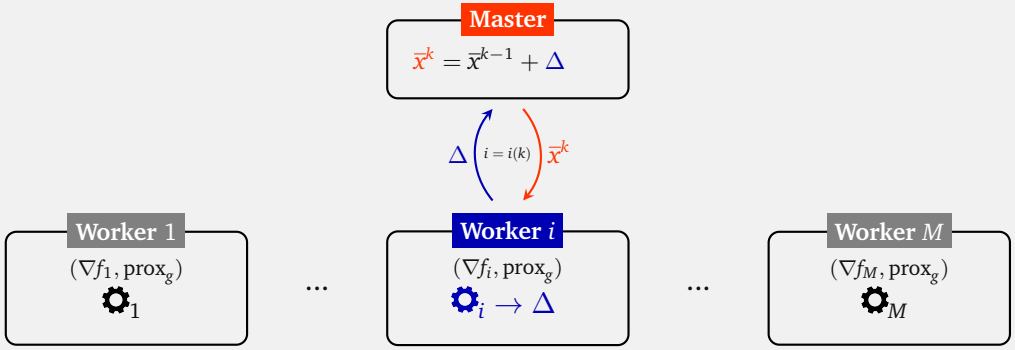


- *iteration* = receive from a worker + master update + send back
- *time* k = number of iterations
- *delay* d_i^k = time since last exchange with i

  $d_i^k = 0$ iff i updates at time k, $d_i^k = d_i^{k-1} + 1$ elsewhere
- *second delay* D_i^k = time since penultimate exchange with i

Algorithm = global communication scheme + local optimization method

what is $\bar{x}$ what is $\bullet_i$



`DAve` communication scheme

master variable $\bar{x}^k$ = combination of workers last contributions $(x_i^{k-d_i^k})_i$

one update/time = one worker contribution **but** all workers are always involved at the master

$$\bar{x}^k = \bar{x}^{k-1} + \Delta \text{ with } \Delta = \pi_i(x_i^k - x_i^{k-D_i^k}) \text{ for } i = i(k)$$

$$\text{i.e.} \quad \bar{x}^k = \sum_{i=1}^M \pi_i x_i^{k-d_i^k} = \sum_{i=1}^M \pi_i \blacksquare_i(\bar{x}^{k-D_i^k})$$

`PG` proximal gradient optimization method

one step of proximal gradient on regularizer g and local loss $f_i = \frac{1}{|\mathcal{S}_i|}\sum_{j\in\mathcal{S}_i}\ell_j$

$\blacksquare_i(\bar{x}) \to \Delta$

$$z \leftarrow \underset{\gamma g}{\mathbf{prox}}(\bar{x})$$

$$x_i \leftarrow z - \gamma\nabla f_i(z)$$

$$\Delta \leftarrow \pi_i\left(x_i - x_i^{prev}\right) \qquad x_i^{prev} \leftarrow x_i$$

DAve-PG

Master:

Initialize $\bar{x}$
while *not converged* **do**
  when a worker finishes:
  Receive adjustment Δ from it
  $\bar{x} \leftarrow \bar{x} + \Delta$
  Send $\bar{x}$ to the agent in return
  $k \leftarrow k + 1$
Interrupt all slaves
Output $x = \mathbf{prox}_{\gamma g}(\bar{x})$

Worker i:

Initialize $x = x_i = \bar{x}$,
while *not interrupted by master* **do**
  Receive the most recent $\bar{x}$
  $z \leftarrow \mathbf{prox}_{\gamma g}(\bar{x})$
  $x_i \leftarrow z - \gamma \nabla f_i(z)$
  $\Delta \leftarrow \pi_i \left(x_i - x_i^{prev} \right) \quad x_i^{prev} \leftarrow x_i$
  Send adjustment Δ to master

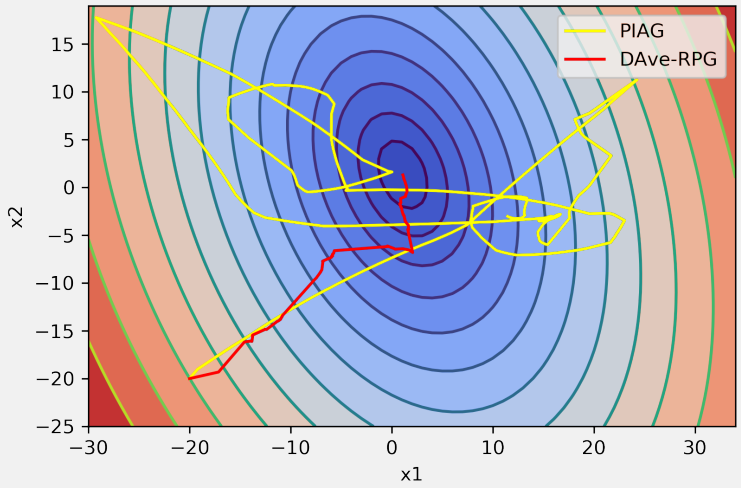
$f_i(x) = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \ell_j(x)$

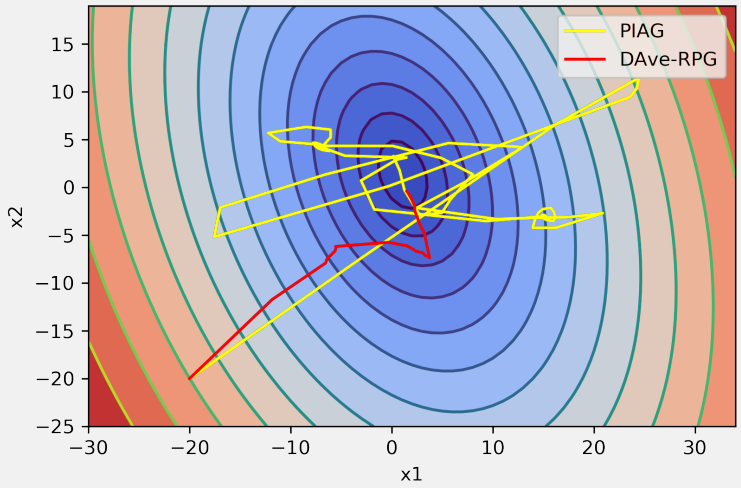
In practice:

- `MPI` blocking Send and Receive
- No computation/storage at the master
- $x = \mathbf{prox}_{\gamma g}(\bar{x})$ is the converging variable

	DAve-PG	PIAG
Combining:	iterates	gradients
Update x^k:	$\mathbf{prox}_{\gamma g}\left(\sum_{i=1}^M \pi_i x^{k-D_i^k} - \gamma \sum_{i=1}^M \pi_i \nabla f_i(x^{k-D_i^k})\right)$	$\mathbf{prox}_{\gamma g}\left(x^{k-1} - \gamma \sum_{i=1}^M \pi_i \nabla f_i(x^{k-D_i^k})\right)$

- Combining *iterates* is **more stable** than combining *gradients*
- Example: 2D quadratic functions on 5 worker
  but one worker 10x slower than the others
  - Stepsize γ of PIAG is 10x smaller due to delays
    the one for DAve-PG stays the same
    to be detailed later
  - DAve-PG is less chaotic and faster than PIAG





- A. Aytekin, H. Feyzmahdavian, and M. Johansson *Analysis and implementation of an asynchronous optimization algorithm for the parameter server*, arXiv:1610.05507.

- N. Vanli, M. Gurbuzbalaban, and A. Ozdaglar *A stronger convergence result on the proximal incremental aggregated gradient method*, arXiv:1611.08022.

Revisiting the clock:

- *epoch sequence* (k_m) = recursively defined by $k_0 = 0$ and

$$k_{m+1} = \min\{k : \text{ each worker made at least 2 updates on the interval } [k_m, k]\}$$
$$= \min\{k : k - D_i^k \geq k_m \text{ for all } i = 1, .., M\}$$

- *epoch time* m = number of epochs

Intuition: k_{m+1} is the first moment when $\bar{x}^k$ no longer depends directly on information prior to k_m. $\bar{x}^k = \sum_{i=1}^M \pi_i x^{k-D_i^k} - \gamma \sum_{i=1}^M \pi_i \nabla f_i(x^{k-D_i^k})$

Theorem

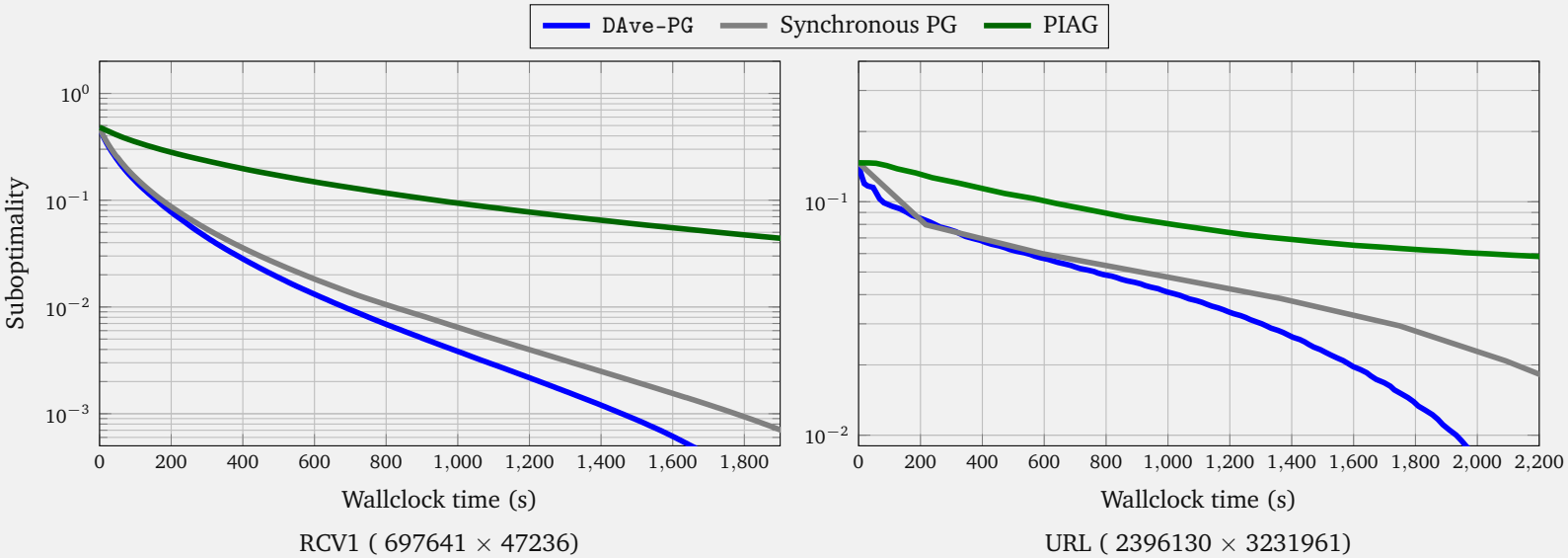
Let each f_i be L-smooth and μ-strongly convex. Then, for $\gamma \in (0, 2/(\mu + L)]$,

$$\forall k \geq k_m, \quad \|x^k - x^\star\|^2 \leq (1 - \alpha)^m \|x^0 - x^\star\|^2$$

where $x^\star$ is the unique minimizer of the $\min_x \sum_{i=1}^M \pi_i f_i(x) + g(x)$ and $\alpha = 2\gamma\mu L/(\mu + L)$.

Exact same result as the synchronous case but over the *epoch time* m, not k

- Logistic regression w/ elastic net $\frac{1}{m}\sum_{j=1}^{m}\log(1+\exp(-y_j z_j^T x)) + \lambda_1\|x\|_1 + \frac{\lambda_2}{2}\|x\|_2^2$
- 100 machines (1 CPU, 1 GB) in a cluster
- 10% of the data in machine one, even on the rest



Distributed Optimization

Asynchronism

Scarse Communications

Conclusion

To exchange less, a solution is to compute more.

DAve-RPG

Master:

Initialize $\bar{x}$
while *not converged* **do**
  when a worker finishes:
  Receive adjustment Δ from it
  $\bar{x} \leftarrow \bar{x} + \Delta$
  Send $\bar{x}$ to the agent in return
  $k \leftarrow k + 1$
Interrupt all slaves
Output $x = \mathbf{prox}_{\gamma g}(\bar{x})$

Worker i:

Initialize $x = x_i = \bar{x}$,
while *not interrupted by master* **do**
  Receive the most recent $\bar{x}$
  Select a number of repetitions p
  Initialize $\Delta = 0$
  for $q = 1$ **to** p **do**
    $z \leftarrow \mathbf{prox}_{\gamma g}(\bar{x} + \Delta)$
    $x_i \leftarrow z - \gamma \nabla f_i(z)$
    $\Delta \leftarrow \Delta + \pi_i \left(x_i - x_i^{prev}\right)$
    $x_i \leftarrow x_i^{prev}$
  Send the adjustment Δ to the master

$f_i(x) = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \ell_j(x)$

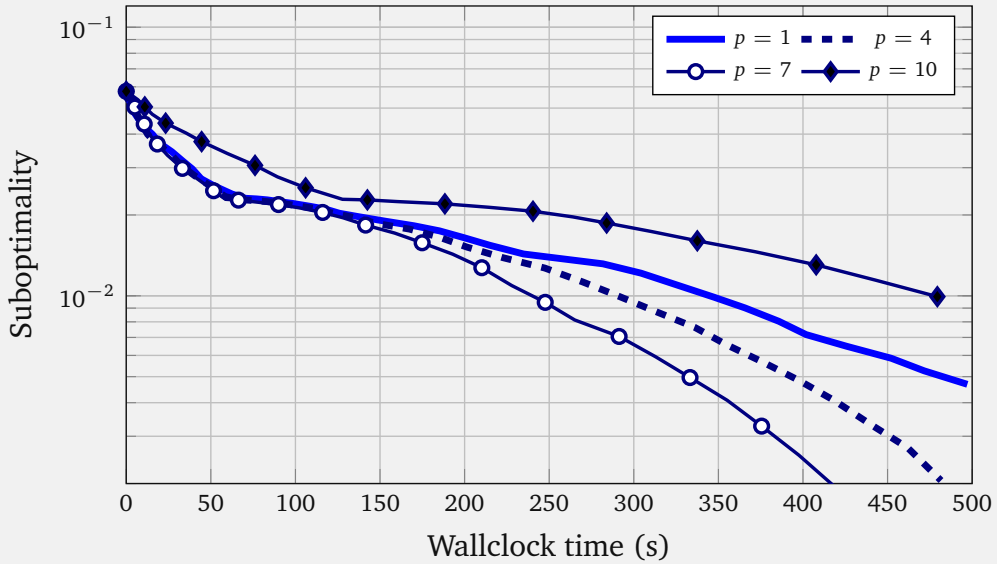
Difference with before:

- at *each* local step, the worker performs p proximal gradient steps
- controlled rate improvement by

$$\max_{\text{repetitions } p \text{ in the epoch}} 1 - \gamma\mu \sum_{q=1}^{p-1} (1-\gamma\mu)^{q-1} \min_i \pi_i^q$$

but the epochs become longer

- Logistic regression w/ elastic net $\frac{1}{m}\sum_{j=1}^{m}\log(1+\exp(-y_j z_j^T x)) + \lambda_1\|x\|_1 + \frac{\lambda_2}{2}\|x\|_2^2$
- 100 machines (1 CPU, 1 GB) in a cluster
- 10% of the data in machine one, even on the rest



- there is a compromise to find ...
- ... but p can be changed *without restrictions*

- Distributed Optimization
- Asynchronism
- Scarse Communications
- Conclusion

Distributed Delay-Tolerant Proximal Gradient Algorithm:

- Simple to implement
- Adaptable to performance/computation compromise
- General, adaptable *epoch* analysis

Poster # 155

Future works:

- Sparse communications
- Using identification to control the communications

Thank you! – Franck IUTZELER `http://www.iutzeler.org`