Convex Optimization

(EE227A: UC Berkeley)

Lecture 24
(Parallel, Distributed – II)

18 Apr, 2013

Suvrit Sra

Admin

- ♠ Reviews due **19 Apr 2013** by 5pm

  Login to easychair as "PC-Member" to enter reviews

Admin

- Reviews due **19 Apr 2013** by 5pm

  Login to easychair as “PC-Member” to enter reviews

- Please take the review seriously:
  - As a reviewer–it’ll be graded
  - As an author–your peers are providing valuable feedback

Admin

- Reviews due **19 Apr 2013** by 5pm

  Login to easychair as "PC-Member" to enter reviews

- Please take the review seriously:
  - As a reviewer–it'll be graded
  - As an author–your peers are providing valuable feedback

- Make up for missed lectures:
  - I'll ask you to view two video lectures
  - HW5 slightly delayed; shorter, simpler
  - Will include questions related to videos

Admin

- Reviews due **19 Apr 2013** by 5pm

  Login to easychair as "PC-Member" to enter reviews

- Please take the review seriously:
  - As a reviewer–it'll be graded
  - As an author–your peers are providing valuable feedback

- Make up for missed lectures:
  - I'll ask you to view two video lectures
  - HW5 slightly delayed; shorter, simpler
  - Will include questions related to videos

- Project **poster** presentations:

Soda 306 HP Auditorium Fri May 10, 2013 4pm – 8pm

Parallel computation – high level views

- Intuition from prev lecture: degree of separability strongly correlated with degree of parallelism possible

Parallel computation – high level views

- Intuition from prev lecture: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism

Parallel computation – high level views

- Intuition from prev lecture: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$

Parallel computation – high level views

- Intuition from prev lecture: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations

Parallel computation – high level views

- Intuition from prev lecture: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations
- Distributed optimization across machines: synchronization and communication biggest burden;

Parallel computation – high level views

- Intuition from prev lecture: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations
- Distributed optimization across machines: synchronization and communication biggest burden; node failure, network failure, load-balancing, etc.

Parallel computation – high level views

- Intuition from prev lecture: degree of separability strongly correlated with degree of parallelism possible
- Not insisting on exact computation allows more parallelism
- Suppose f is the fraction of sequential computation. Then speedup for **any** number of processors (cores) is $\leq 1/f$
- Parallel optimization on multi-core machines: shared memory architecture. Main penalty: synchronization / atomic operations
- Distributed optimization across machines: synchronization and communication biggest burden; node failure, network failure, load-balancing, etc.
- Synchronous vs. asynchronous computation

Poor man's parallelism

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

- Introduce (local) variables $(x_1, \ldots, x_m)$

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

- Introduce (local) variables $(x_1, \ldots, x_m)$
- Problem is now over $\mathcal{H}^m := \mathcal{H} \times \mathcal{H} \times \cdots \times \mathcal{H}$ (m-times)

Separable optimization

$$\min \quad f(x) := \sum_{i=1}^{m} f_i(x) \quad x \in \mathbb{R}^n.$$

Product space trick

- Introduce (local) variables $(x_1, \ldots, x_m)$
- Problem is now over $\mathcal{H}^m := \mathcal{H} \times \mathcal{H} \times \cdots \times \mathcal{H}$ (m-times)
- **Consensus** constraint: $x_1 = x_2 = \ldots = x_m$

$$\begin{aligned} \min_{(x_1,\ldots,x_m)} \quad & \sum_i f_i(x_i) \\ \text{s.t.} \quad & x_1 = x_2 = \cdots = x_m. \end{aligned}$$

Separable optimization

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mathbb{I}_{\mathcal{B}}(\boldsymbol{x})$$

where $\boldsymbol{x} \in \mathcal{H}^m$ and $\mathcal{B} = \{\boldsymbol{z} \in \mathcal{H}^m \mid \boldsymbol{z} = (x, x, \ldots, x)\}$

Separable optimization

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mathbb{I}_{\mathcal{B}}(\boldsymbol{x})$$

where $\boldsymbol{x} \in \mathcal{H}^m$ and $\mathcal{B} = \{\boldsymbol{z} \in \mathcal{H}^m \mid \boldsymbol{z} = (x, x, \ldots, x)\}$

- Can solve using DR method

Separable optimization

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mathbb{I}_{\mathcal{B}}(\boldsymbol{x})$$

where $\boldsymbol{x} \in \mathcal{H}^m$ and $\mathcal{B} = \{\boldsymbol{z} \in \mathcal{H}^m \mid \boldsymbol{z} = (x, x, \ldots, x)\}$

- Can solve using DR method
- Each component of $f_i(x_i)$ independently in parallel
- Communicate / synchronize to ensure consensus

The ADMM view

Let us see separable objective with constraints

The ADMM view

Let us see separable objective with constraints

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c. \end{aligned}$$

The ADMM view

Let us see separable objective with constraints

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c. \end{aligned}$$

- Objective function is separable into two sets x and z variables
- The constraint prevents a trivial decoupling

The ADMM view

Let us see separable objective with constraints

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c. \end{aligned}$$

- Objective function is separable into two sets x and z variables
- The constraint prevents a trivial decoupling
- Introduce **augmented lagrangian** (AL)

$$L_\rho(x, z, y) := f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$$

The ADMM view

Let us see separable objective with constraints

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c. \end{aligned}$$

- Objective function is separable into two sets x and z variables
- The constraint prevents a trivial decoupling
- Introduce **augmented lagrangian** (AL)

$$L_\rho(x,z,y) := f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$$

- Now, a Gauss-Seidel style update to the AL

The ADMM view

Let us see separable objective with constraints

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c. \end{aligned}$$

- Objective function is separable into two sets x and z variables
- The constraint prevents a trivial decoupling
- Introduce **augmented lagrangian** (AL)

$$L_\rho(x, z, y) := f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$$

- Now, a Gauss-Seidel style update to the AL

$$x_{k+1} = \operatorname{argmin}_x L_\rho(x, z_k, y_k)$$

The ADMM view

Let us see separable objective with constraints

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c. \end{aligned}$$

- Objective function is separable into two sets x and z variables
- The constraint prevents a trivial decoupling
- Introduce **augmented lagrangian** (AL)

$$L_\rho(x, z, y) := f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$$

- Now, a Gauss-Seidel style update to the AL

$$\begin{aligned} x_{k+1} &= \operatorname{argmin}_x L_\rho(x, z_k, y_k) \\ z_{k+1} &= \operatorname{argmin}_z L_\rho(x_{k+1}, z, y_k) \end{aligned}$$

The ADMM view

Let us see separable objective with constraints

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c. \end{aligned}$$

- Objective function is separable into two sets x and z variables
- The constraint prevents a trivial decoupling
- Introduce **augmented lagrangian** (AL)

$$L_\rho(x, z, y) := f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$$

- Now, a Gauss-Seidel style update to the AL

$$\begin{aligned} x_{k+1} &= \operatorname{argmin}_x L_\rho(x, z_k, y_k) \\ z_{k+1} &= \operatorname{argmin}_z L_\rho(x_{k+1}, z, y_k) \\ y_{k+1} &= y_k + \rho(Ax_{k+1} + Bz_{k+1} - c) \end{aligned}$$

ADMM – scaled version

- The AL is

$$L_\rho(x,z,y) := f(x)+g(z)+y^T(Ax+Bz-c)+\frac{\rho}{2}\|Ax+Bz-c\|_2^2$$

ADMM – scaled version

- The AL is

$$L_\rho(x,z,y) := f(x)+g(z)+y^T(Ax+Bz-c)+\tfrac{\rho}{2}\|Ax+Bz-c\|_2^2$$

- Combine linear and quadratic terms in L_ρ, so we have

$$L_\rho(x,z,y) = f(x)+g(z)+\tfrac{\rho}{2}\|Ax+Bz-c+d\|_2^2 + \text{constants}$$

  where we use $d_k = (1/\rho)y_k$ as a new variable.
- **Exercise:** Verify above algebra.

ADMM – scaled version

- The AL is

$$L_\rho(x,z,y) := f(x)+g(z)+y^T(Ax+Bz-c)+\frac{\rho}{2}\|Ax+Bz-c\|_2^2$$

- Combine linear and quadratic terms in L_ρ, so we have

$$L_\rho(x,z,y) = f(x)+g(z)+\frac{\rho}{2}\|Ax+Bz-c+d\|_2^2 + \text{constants}$$

  where we use $d_k = (1/\rho)y_k$ as a new variable.
- **Exercise:** Verify above algebra.

Scaled ADMM

$$x_{k+1} \quad = \quad \operatorname{argmin}_x f(x)+\frac{\rho}{2}\|Ax+Bz_k-c+d_k\|_2^2$$

ADMM – scaled version

- The AL is

$$L_\rho(x,z,y) := f(x)+g(z)+y^T(Ax+Bz-c)+\frac{\rho}{2}\|Ax+Bz-c\|_2^2$$

- Combine linear and quadratic terms in L_ρ, so we have

$$L_\rho(x,z,y) = f(x)+g(z)+\frac{\rho}{2}\|Ax+Bz-c+d\|_2^2 + \text{constants}$$

where we use $d_k = (1/\rho)y_k$ as a new variable.

- **Exercise:** Verify above algebra.

Scaled ADMM

$$\begin{aligned} x_{k+1} &= \operatorname{argmin}_x f(x) + \frac{\rho}{2}\|Ax+Bz_k-c+d_k\|_2^2 \\ z_{k+1} &= \operatorname{argmin}_z g(z) + \frac{\rho}{2}\|Ax_{k+1}+Bz-c+d_k\|_2^2 \end{aligned}$$

ADMM – scaled version

- The AL is

$$L_\rho(x,z,y) := f(x)+g(z)+y^T(Ax+Bz-c)+\tfrac{\rho}{2}\|Ax+Bz-c\|_2^2$$

- Combine linear and quadratic terms in L_ρ, so we have

$$L_\rho(x,z,y) = f(x)+g(z)+\tfrac{\rho}{2}\|Ax+Bz-c+d\|_2^2 + \text{constants}$$

where we use $d_k = (1/\rho)y_k$ as a new variable.

- **Exercise:** Verify above algebra.

Scaled ADMM

$$\begin{aligned} x_{k+1} &= \operatorname{argmin}_x f(x) + \tfrac{\rho}{2}\|Ax + Bz_k - c + d_k\|_2^2 \\ z_{k+1} &= \operatorname{argmin}_z g(z) + \tfrac{\rho}{2}\|Ax_{k+1} + Bz - c + d_k\|_2^2 \\ d_{k+1} &= d_k + (Ax_{k+1} + Bz_{k+1} - c) \end{aligned}$$

ADMM – Parallel / distributed version

$$\boxed{\min f(x) = \sum_i f_i(x)}$$

ADMM – Parallel / distributed version

$$\boxed{\min f(x) = \sum_i f_i(x)}$$

Product space form for ADMM

$$\begin{aligned} \min_{x_1,\ldots,x_m,z} \quad & \sum_{i=1}^m f_i(x_i) \\ \text{s.t.} \quad & x_i - z = 0, \quad i = 1,\ldots,m. \end{aligned}$$

ADMM – Parallel / distributed version

$$\boxed{\min f(x) = \sum_i f_i(x)}$$

Product space form for ADMM

$$\begin{aligned} \min_{x_1,\ldots,x_m,z} \quad & \sum_{i=1}^m f_i(x_i) \\ \text{s.t.} \quad & x_i - z = 0, \quad i = 1,\ldots,m. \end{aligned}$$

♠ **Local** variables x_i – one vector per processor / cluster node

ADMM – Parallel / distributed version

$$\boxed{\min f(x) = \textstyle\sum_i f_i(x)}$$

Product space form for ADMM

$$\begin{aligned} \min_{x_1,\ldots,x_m,z} \quad & \textstyle\sum_{i=1}^m f_i(x_i) \\ \text{s.t.} \quad & x_i - z = 0, \quad i = 1, \ldots, m. \end{aligned}$$

- ♠ **Local** variables x_i – one vector per processor / cluster node
- ♠ z is the **global**, shared variable

ADMM – Parallel / distributed version

$$\boxed{\min f(x) = \sum_i f_i(x)}$$

Product space form for ADMM

$$\begin{aligned} \min_{x_1,\ldots,x_m,z} \quad & \sum_{i=1}^m f_i(x_i) \\ \text{s.t.} \quad & x_i - z = 0, \quad i = 1,\ldots,m. \end{aligned}$$

- **Local** variables x_i – one vector per processor / cluster node
- z is the **global**, shared variable
- $x_i - z = 0$ is called *consensus constraint*

ADMM – Parallel / distributed version

Augmented Lagrangian

$$L_\rho(\boldsymbol{x}, z, y) := \sum_{i=1}^{m} \Big(f_i(x_i) + y_i^T(x_i - z) + \frac{\rho}{2}\|x_i - z\|_2^2\Big)$$

ADMM – Parallel / distributed version

Augmented Lagrangian

$$L_\rho(\boldsymbol{x}, z, y) := \sum_{i=1}^{m} \big(f_i(x_i) + y_i^T(x_i - z) + \tfrac{\rho}{2}\|x_i - z\|_2^2\big)$$

ADMM updates

$$[x_i]_{k+1} \quad = \quad \operatorname{argmin}_{x_i} f_i(x_i) + [y_i]_k^T(x_i - z_k) + \tfrac{\rho}{2}\|x_i - z_k\|_2^2$$

ADMM – Parallel / distributed version

Augmented Lagrangian

$$L_\rho(\boldsymbol{x}, z, y) := \sum_{i=1}^{m} \big(f_i(x_i) + y_i^T(x_i - z) + \tfrac{\rho}{2}\|x_i - z\|_2^2\big)$$

ADMM updates

$$\begin{aligned}
[x_i]_{k+1} &= \operatorname{argmin}_{x_i} f_i(x_i) + [y_i]_k^T(x_i - z_k) + \tfrac{\rho}{2}\|x_i - z_k\|_2^2 \\
z_{k+1} &= \tfrac{1}{m}\sum_i \Big([x_i]_{k+1} + \tfrac{1}{\rho}[y_i]_k\Big)
\end{aligned}$$

ADMM – Parallel / distributed version

Augmented Lagrangian

$$L_\rho(\boldsymbol{x}, z, y) := \sum_{i=1}^{m} \left(f_i(x_i) + y_i^T(x_i - z) + \tfrac{\rho}{2}\|x_i - z\|_2^2\right)$$

ADMM updates

$$\begin{aligned}
[x_i]_{k+1} &= \operatorname{argmin}_{x_i} f_i(x_i) + [y_i]_k^T(x_i - z_k) + \tfrac{\rho}{2}\|x_i - z_k\|_2^2 \\
z_{k+1} &= \tfrac{1}{m}\sum_i \left([x_i]_{k+1} + \tfrac{1}{\rho}[y_i]_k\right) \\
[y_i]_{k+1} &= [y_i]_k + \rho([x_i]_{k+1} - z_{k+1})
\end{aligned}$$

ADMM – Parallel / distributed version

Augmented Lagrangian

$$L_\rho(\boldsymbol{x}, z, y) := \sum_{i=1}^{m} \left(f_i(x_i) + y_i^T (x_i - z) + \tfrac{\rho}{2} \|x_i - z\|_2^2 \right)$$

ADMM updates

$$\begin{aligned}
[x_i]_{k+1} &= \operatorname{argmin}_{x_i} f_i(x_i) + [y_i]_k^T (x_i - z_k) + \tfrac{\rho}{2} \|x_i - z_k\|_2^2 \\
z_{k+1} &= \tfrac{1}{m} \sum_i \left([x_i]_{k+1} + \tfrac{1}{\rho} [y_i]_k \right) \\
[y_i]_{k+1} &= [y_i]_k + \rho([x_i]_{k+1} - z_{k+1})
\end{aligned}$$

Exercise: Verify the above updates (use unscaled ADMM)

ADMM – Parallel / distributed version

Augmented Lagrangian

$$L_\rho(\boldsymbol{x}, z, y) := \sum_{i=1}^{m} \left(f_i(x_i) + y_i^T(x_i - z) + \tfrac{\rho}{2}\|x_i - z\|_2^2\right)$$

ADMM updates

$$\begin{aligned}
[x_i]_{k+1} &= \operatorname{argmin}_{x_i} f_i(x_i) + [y_i]_k^T(x_i - z_k) + \tfrac{\rho}{2}\|x_i - z_k\|_2^2 \\
z_{k+1} &= \tfrac{1}{m}\sum_i \left([x_i]_{k+1} + \tfrac{1}{\rho}[y_i]_k\right) \\
[y_i]_{k+1} &= [y_i]_k + \rho([x_i]_{k+1} - z_{k+1})
\end{aligned}$$

Exercise: Verify the above updates (use unscaled ADMM)

- The x_i updates in parallel; synchronize to update z and y

Asynchronous methods

Trivial methods so far

$$\boxed{\min \quad f(x) = \sum_{i=1}^{m} f_i(x) \quad x \in \mathcal{X}}$$

Trivial methods so far

$$\boxed{\min \quad f(x) = \sum_{i=1}^m f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^m g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

Trivial methods so far

$$\boxed{\min \quad f(x) = \sum_{i=1}^m f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^m g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

♣ The sum has m components

Trivial methods so far

$$\boxed{\min \quad f(x) = \sum_{i=1}^m f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^m g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

- ♣ The sum has m components
- ♣ Trivial parallelization: compute each $g_i(x_k)$ on diff. processor

Trivial methods so far

$$\boxed{\min \quad f(x) = \sum_{i=1}^m f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^m g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

- ♣ The sum has m components
- ♣ Trivial parallelization: compute each $g_i(x_k)$ on diff. processor
- ♣ Then collect the answers on a master node

Trivial methods so far

$$\boxed{\min \quad f(x) = \sum_{i=1}^{m} f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^{m} g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

- ♣ The sum has m components
- ♣ Trivial parallelization: compute each $g_i(x_k)$ on diff. processor
- ♣ Then collect the answers on a master node
- ♣ Update α_k and x_{k+1} in serial

Trivial methods so far

$$\boxed{\min \quad f(x) = \textstyle\sum_{i=1}^m f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^m g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

- ♣ The sum has m components
- ♣ Trivial parallelization: compute each $g_i(x_k)$ on diff. processor
- ♣ Then collect the answers on a master node
- ♣ Update α_k and x_{k+1} in serial
- ♣ Share / Broadcast x_{k+1} and repeat

Trivial methods so far

$$\boxed{\min \quad f(x) = \textstyle\sum_{i=1}^m f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^m g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

- ♣ The sum has m components
- ♣ Trivial parallelization: compute each $g_i(x_k)$ on diff. processor
- ♣ Then collect the answers on a master node
- ♣ Update α_k and x_{k+1} in serial
- ♣ Share / Broadcast x_{k+1} and repeat
- ♣ Highly **synchronized** computation

Trivial methods so far

$$\boxed{\min \quad f(x) = \sum_{i=1}^m f_i(x) \quad x \in \mathcal{X}}$$

$$x_{k+1} = P_{\mathcal{X}}(x_k - \alpha_k \sum_{i=1}^m g_i(x_k)),$$

where $g_i \in \partial f_i(x_k)$ — so that $\sum_i g_i \in \partial f(x_k)$

- ♣ The sum has m components
- ♣ Trivial parallelization: compute each $g_i(x_k)$ on diff. processor
- ♣ Then collect the answers on a master node
- ♣ Update α_k and x_{k+1} in serial
- ♣ Share / Broadcast x_{k+1} and repeat
- ♣ Highly **synchronized** computation
- ♣ Makes sense if computing a single subgradient takes much longer than the involved costs of *synchronization*

Partially asynchronous methods

If even one of the processors is slow in computing its subgradient $g_i(x_k)$, the whole update gets blocked due to synchronization

Partially asynchronous methods

If even one of the processors is slow in computing its subgradient $g_i(x_k)$, the whole update gets blocked due to synchronization

Asynchronous updates

$$x_{k+1} = x_k - \alpha_k \sum_{i=1}^{m} g_i(k - \tau_i)$$

where $g_i(k - \tau_i)$ is a **delayed subgradient**.

Notation: We write $g_i(k) \equiv g_i(x_k)$

- Master slave architecture

Partially asynchronous methods

- ♣ If no delay, then $\tau_i = 0$ — synchronized case

Partially asynchronous methods

- ♣ If no delay, then $\tau_i = 0$ — synchronized case
- ♣ Each processor can have its own arbitrary delay τ_i

Partially asynchronous methods

- ♣ If no delay, then $\tau_i = 0$ — synchronized case
- ♣ Each processor can have its own arbitrary delay τ_i
- ♣ If $g_i(k)$ **not available** from node i, **don't block** the update

Partially asynchronous methods

- ♣ If no delay, then $\tau_i = 0$ — synchronized case
- ♣ Each processor can have its own arbitrary delay τ_i
- ♣ If $g_i(k)$ **not available** from node i, **don't block** the update
- ♣ instead we go ahead and use the **most recently available** subgradient $g_i(k - \tau_i)$ from processor i

Partially asynchronous methods

- ♣ If no delay, then $\tau_i = 0$ — synchronized case
- ♣ Each processor can have its own arbitrary delay τ_i
- ♣ If $g_i(k)$ **not available** from node i, **don't block** the update
- ♣ instead we go ahead and use the **most recently available** subgradient $g_i(k - \tau_i)$ from processor i
- ♣ Partial asynchrony: delays can be arbitrary but **bounded**

Partially asynchronous methods

- ♣ If no delay, then $\tau_i = 0$ — synchronized case
- ♣ Each processor can have its own arbitrary delay τ_i
- ♣ If $g_i(k)$ **not available** from node i, **don't block** the update
- ♣ instead we go ahead and use the **most recently available** subgradient $g_i(k - \tau_i)$ from processor i
- ♣ Partial asynchrony: delays can be arbitrary but **bounded**
- ♣ Key idea to analyze: view asynchronous method as an iterative gradient-method with deterministic or stochastic errors.

Partially asynchronous methods

- If no delay, then $\tau_i = 0$ — synchronized case
- Each processor can have its own arbitrary delay τ_i
- If $g_i(k)$ **not available** from node i, **don't block** the update
- instead we go ahead and use the **most recently available** subgradient $g_i(k - \tau_i)$ from processor i
- Partial asynchrony: delays can be arbitrary but **bounded**
- Key idea to analyze: view asynchronous method as an iterative gradient-method with deterministic or stochastic errors.

Delays impact speed of convergence

Partially asynchronous methods

- If no delay, then $\tau_i = 0$ — synchronized case
- Each processor can have its own arbitrary delay τ_i
- If $g_i(k)$ **not available** from node i, **don't block** the update
- instead we go ahead and use the **most recently available** subgradient $g_i(k-\tau_i)$ from processor i
- Partial asynchrony: delays can be arbitrary but **bounded**
- Key idea to analyze: view asynchronous method as an iterative gradient-method with deterministic or stochastic errors.

Delays impact speed of convergence

Delay τ, leads to convergence rate: $O(\sqrt{\tau/T})$.

Partially asynchronous methods

Algorithm 1: Projected subgradient

$$g_{\mathsf{avg}}(k) := \sum\nolimits_i \lambda_i g_i(k - \tau_i)$$

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathcal{X}} \quad \left\{ \langle g_{\mathsf{avg}}(k),\, x \rangle + \frac{1}{\alpha_k} \|x - x_k\|_2^2 \right\}$$

Partially asynchronous methods

Algorithm 1: Projected subgradient

$$g_{\mathsf{avg}}(k) := \sum\nolimits_i \lambda_i g_i(k - \tau_i)$$

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathcal{X}} \left\{ \langle g_{\mathsf{avg}}(k),\, x\rangle + \frac{1}{\alpha_k}\|x - x_k\|_2^2 \right\}$$

Algorithm 2: Mirror descent version

$$x_{k+1} = \operatorname*{argmin}_{x} \left\{ \langle g_{\mathsf{avg}}(k),\, x\rangle + \frac{1}{\alpha_k} D_\phi(x, x_k) \right\}$$

$D_\phi(x, y)$ is some strongly convex Bregman divergence

Partially asynchronous methods

- Method also works for stochastic optimization, if $g_i(k - \tau_i)$ is a stochastic subgradient.

Partially asynchronous methods

- Method also works for stochastic optimization, if $g_i(k - \tau_i)$ is a stochastic subgradient.
- Since i.i.d. sampling of subgradients assumed, each processor can sample its own subgradients concurrently; subsequent averaging to use g_{avg} reduces variance.

Partially asynchronous methods

- Method also works for stochastic optimization, if $g_i(k - \tau_i)$ is a stochastic subgradient.
- Since i.i.d. sampling of subgradients assumed, each processor can sample its own subgradients concurrently; subsequent averaging to use g_{avg} reduces variance.
- Convergence rates depend on: *network topology*, *delay process*, and *objective smoothness* (by choosing stepsize α_k)

Comparison: syn vs asyn

- Suppose we iterate $x = Ax$ where $A = \begin{bmatrix} a & b \\ b & a \end{bmatrix}$

Comparison: syn vs asyn

- Suppose we iterate $x = Ax$ where $A = \begin{bmatrix} a & b \\ b & a \end{bmatrix}$
- The iteration updates are

$$\begin{aligned} x_1 &\leftarrow ax_1 + bx_2 \\ x_2 &\leftarrow bx_1 + ax_2. \end{aligned}$$

Comparison: syn vs asyn

- Suppose we iterate $x = Ax$ where $A = \begin{bmatrix} a & b \\ b & a \end{bmatrix}$
- The iteration updates are

$$\begin{aligned} x_1 &\leftarrow ax_1 + bx_2 \\ x_2 &\leftarrow bx_1 + ax_2. \end{aligned}$$

- Suppose processor 1 updates x_1; processor 2 updates x_2
- After updates, x_1 and x_2 communicated to each other

Comparison: syn vs asyn

- Suppose we iterate $x = Ax$ where $A = \begin{bmatrix} a & b \\ b & a \end{bmatrix}$
- The iteration updates are

$$\begin{aligned} x_1 &\leftarrow ax_1 + bx_2 \\ x_2 &\leftarrow bx_1 + ax_2. \end{aligned}$$

- Suppose processor 1 updates x_1; processor 2 updates x_2
- After updates, x_1 and x_2 communicated to each other
- Say **update** requires 1 unit of time, and **communication** requires $\tau \geq 1$ units of time

Comparison: syn vs asyn

- Synchronous: values received at times $\tau + 1, 2(\tau + 1), \ldots$

Comparison: syn vs asyn

- Synchronous: values received at times $\tau + 1, 2(\tau + 1), \ldots$
- Say $x_i(t)$ is value at processor i at time t

Comparison: syn vs asyn

- Synchronous: values received at times $\tau + 1, 2(\tau + 1), \ldots$
- Say $x_i(t)$ is value at processor i at time t
- So in the **synchronous** case we have

$$x_1(t+1) \leftarrow a x_1(t-\tau) + b x_2(t-\tau)$$
$$x_2(t+1) \leftarrow b x_1(t-\tau) + a x_2(t-\tau).$$

Comparison: syn vs asyn

- Synchronous: values received at times $\tau + 1, 2(\tau + 1), \ldots$
- Say $x_i(t)$ is value at processor i at time t
- So in the **synchronous** case we have

$$x_1(t+1) \leftarrow a x_1(t-\tau) + b x_2(t-\tau)$$
$$x_2(t+1) \leftarrow b x_1(t-\tau) + a x_2(t-\tau).$$

- Asynchronous: processor i updates own variable regardless of whether it has the latest value from the other processor
- Thus, in the **asynchronous** case we have

$$x_1(t+1) \leftarrow a x_1(t) + b x_2(t-\tau)$$
$$x_2(t+1) \leftarrow b x_1(t-\tau) + a x_2(t).$$

- In both cases, use base case: $x_i(t) = x_i(0)$ for $-D \leq t < 0$

Comparison: syn vs asyn

- Can be shown if $|a| + |b| < 1$ then both syn and asyn converge to $x^* = (0, 0)$

Comparison: syn vs asyn

- Can be shown if $|a| + |b| < 1$ then both syn and asyn converge to $x^* = (0, 0)$
- Say we have $\rho > 0$ such that

$$|a|\rho^{-\tau} + |b|\rho^{-\tau} \leq \rho,$$

  then the synchronous sequence $x_i(t)$ satisfies

$$|x_i(t)| \leq C\rho^t, \quad \forall t = 0, 1, \ldots,$$

  where $C = \max\{|x_1(0)|, |x_2(0)|\}$
- **Exercise:** Use induction on t to prove above claim.

Comparison: syn vs asyn

- Smallest synchronous ρ is $\rho_S = (|a| + |b|)^{1/(\tau+1)}$

Comparison: syn vs asyn

- Smallest synchronous ρ is $\rho_S = (|a| + |b|)^{1/(\tau+1)}$
- For asynchronous case, if we have

$$|a| + |b|\rho^{-\tau} \leq \rho,$$

then inductively can show that $|x_i(t)| \leq C\rho^t$ (same C as above)

Comparison: syn vs asyn

- Smallest synchronous ρ is $\rho_S = (|a| + |b|)^{1/(\tau+1)}$
- For asynchronous case, if we have

$$|a| + |b|\rho^{-\tau} \leq \rho,$$

  then inductively can show that $|x_i(t)| \leq C\rho^t$ (same C as above)
- Smallest valid ρ is $\rho_A > 0$ that solves $|a| + |b|\rho_A^{-\tau} = \rho_A$

Comparison: syn vs asyn

- Smallest synchronous ρ is $\rho_S = (|a| + |b|)^{1/(\tau+1)}$
- For asynchronous case, if we have

$$|a| + |b|\rho^{-\tau} \leq \rho,$$

  then inductively can show that $|x_i(t)| \leq C\rho^t$ (same C as above)
- Smallest valid ρ is $\rho_A > 0$ that solves $|a| + |b|\rho_A^{-\tau} = \rho_A$
- **Verify** that $\rho_A \leq \rho_S$

Comparison: syn vs asyn

- Smallest synchronous ρ is $\rho_S = (|a| + |b|)^{1/(\tau+1)}$
- For asynchronous case, if we have

$$|a| + |b|\rho^{-\tau} \leq \rho,$$

  then inductively can show that $|x_i(t)| \leq C\rho^t$ (same C as above)
- Smallest valid ρ is $\rho_A > 0$ that solves $|a| + |b|\rho_A^{-\tau} = \rho_A$
- **Verify** that $\rho_A \leq \rho_S$
- Thus, the asynchronous version converges faster

Comparison: syn vs asyn

- Smallest synchronous ρ is $\rho_S = (|a| + |b|)^{1/(\tau+1)}$
- For asynchronous case, if we have

$$|a| + |b|\rho^{-\tau} \leq \rho,$$

  then inductively can show that $|x_i(t)| \leq C\rho^t$ (same C as above)
- Smallest valid ρ is $\rho_A > 0$ that solves $|a| + |b|\rho_A^{-\tau} = \rho_A$
- **Verify** that $\rho_A \leq \rho_S$
- Thus, the asynchronous version converges faster
- But it requires more message transmissions

Distributed optimization

Foundations of distributed computation

http://videolectures.net/nipsworkshops2010_tsitsiklis_aad/

Implementation oriented talk

http://videolectures.net/nipsworkshops2012_smola_parameter_server/

References

- ♠ D. P. Bertsekas and J. N. Tsitsiklis (1997). *Parallel and Distributed Computation: Numerical methods*
- ♠ S. Boyd. (2012). *ADMM notes*
- ♠ A. Agarwal and J. Duchi (2011). *Distributed delayed stochastic optimization*