Privacy and Fault-Tolerance in Distributed Optimization

Nitin Vaidya
University of Illinois at Urbana-Champaign

Acknowledgements

Shripad Gade

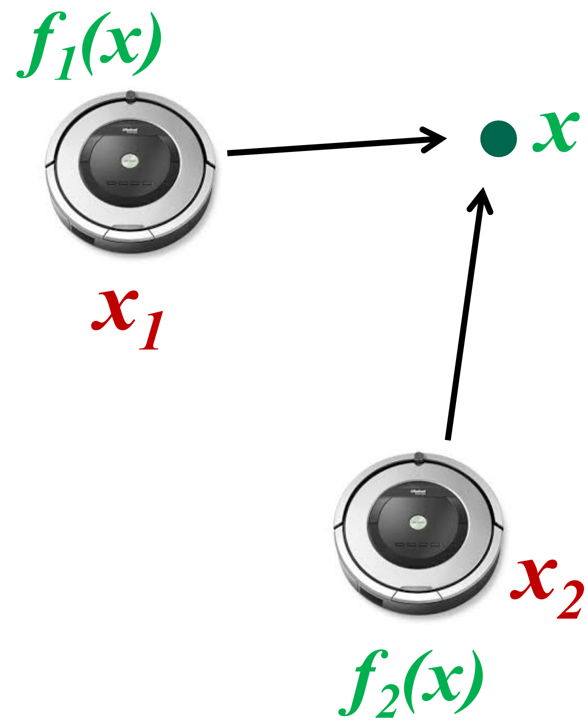
Lili Su

$$argmin \sum_{i} f_i(x)$$

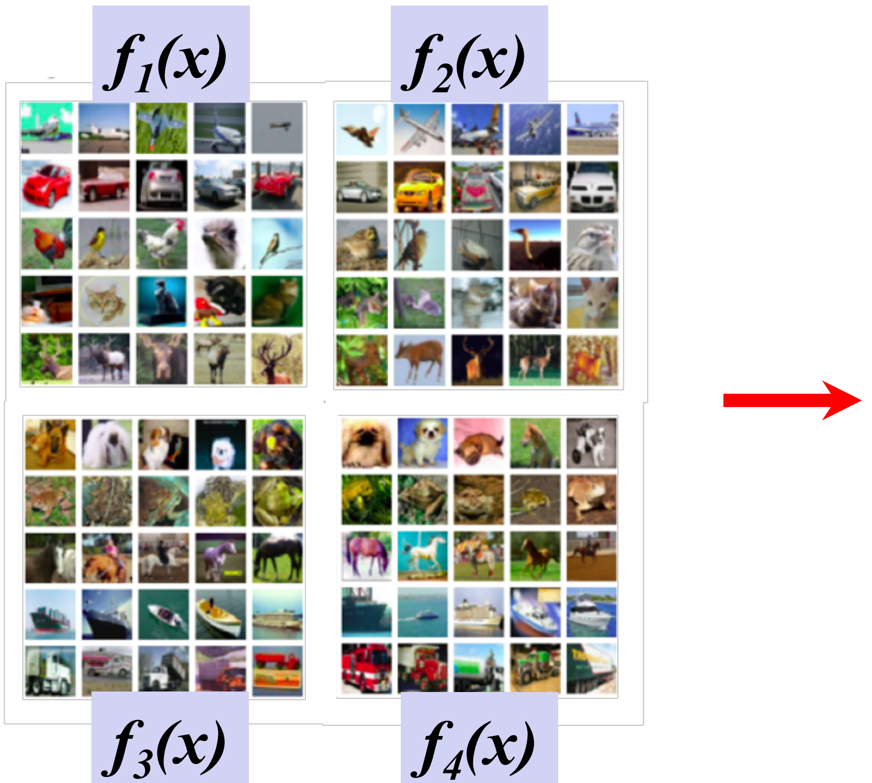
Applications

- $f_i(x)$ = cost for robot i to go to location x
- Minimize total cost of rendezvous

$$argmin \sum_i f_i(x)$$



Applications



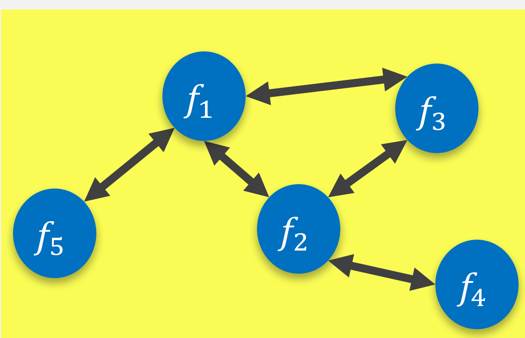
Learning

Minimize cost

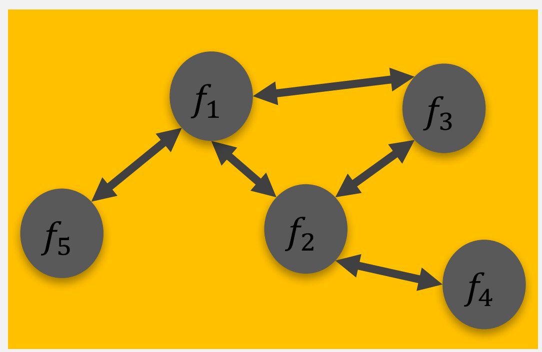
$$\sum_i f_i(x)$$

Outline

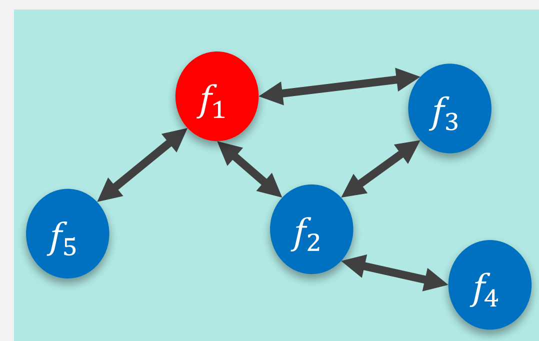
$$argmin \sum_{i} f_i(x)$$



Distributed Optimization

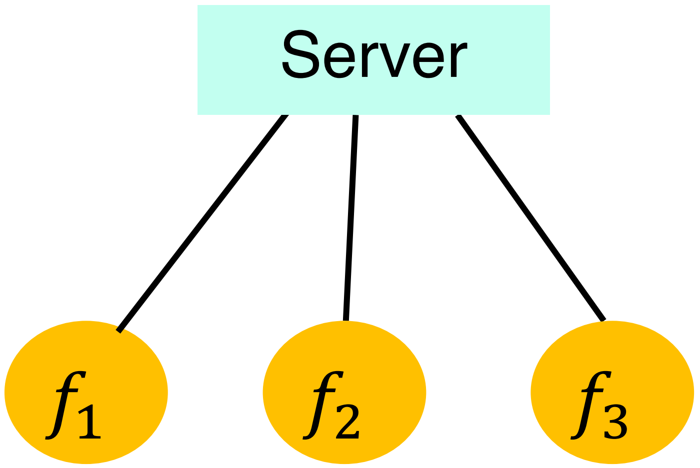


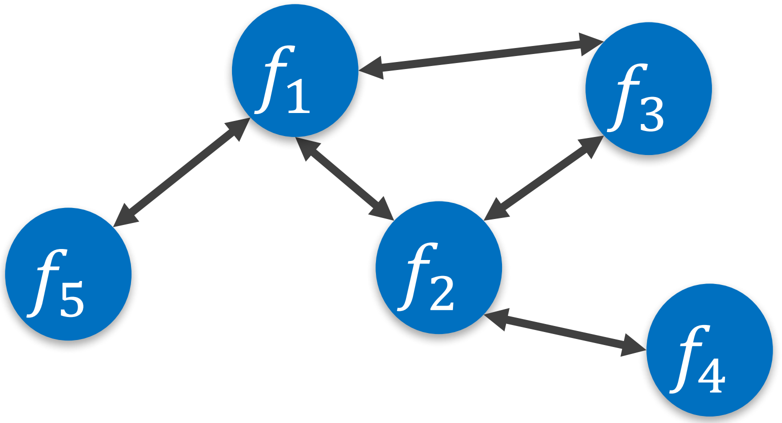
Privacy



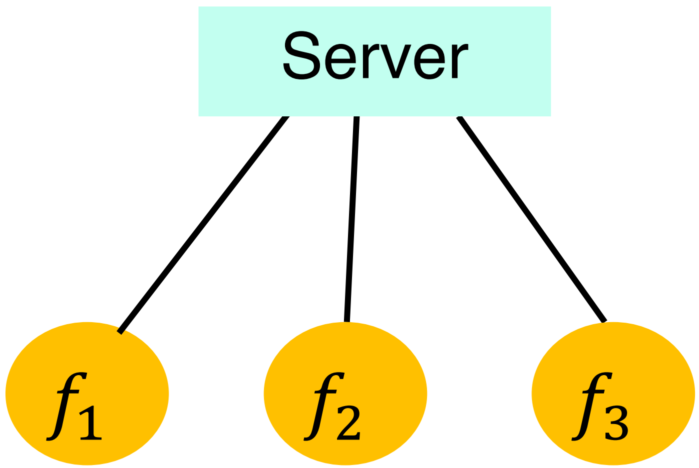
Fault-tolerance

Distributed Optimization



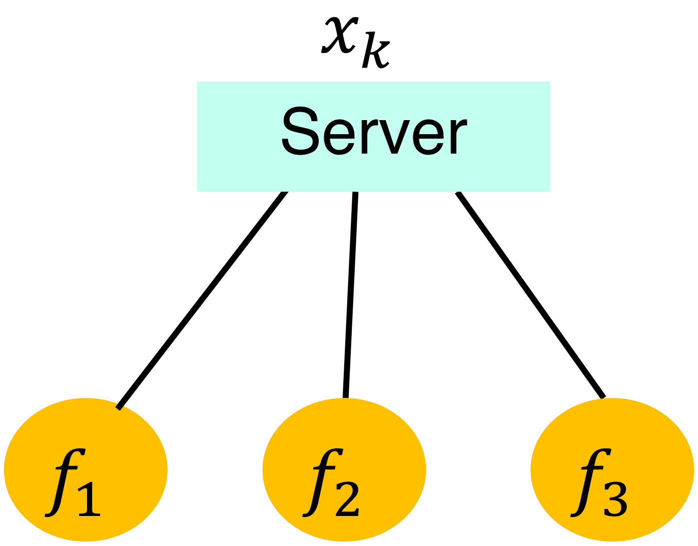


Client-Server Architecture



Client-Server Architecture

- Server maintains estimate x_k
- Client i knows $f_i(x)$

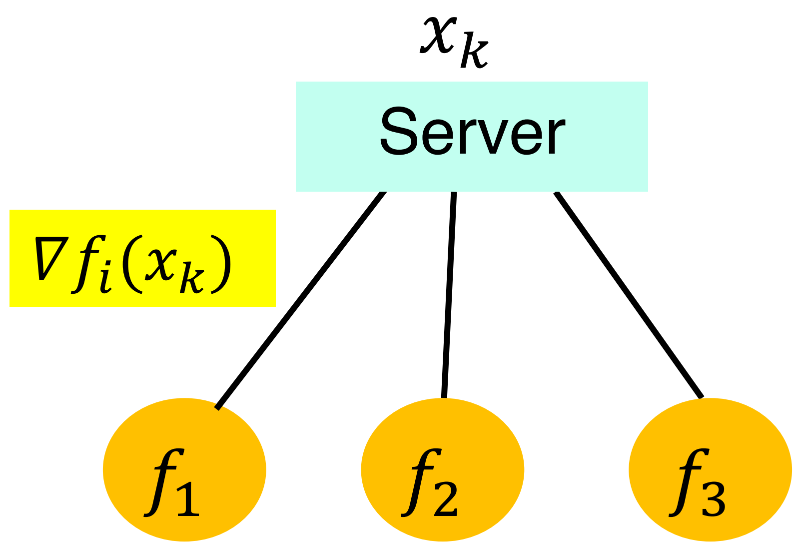


Client-Server Architecture

- Server maintains estimate x_k
- Client i knows $f_i(x)$

In iteration k+1

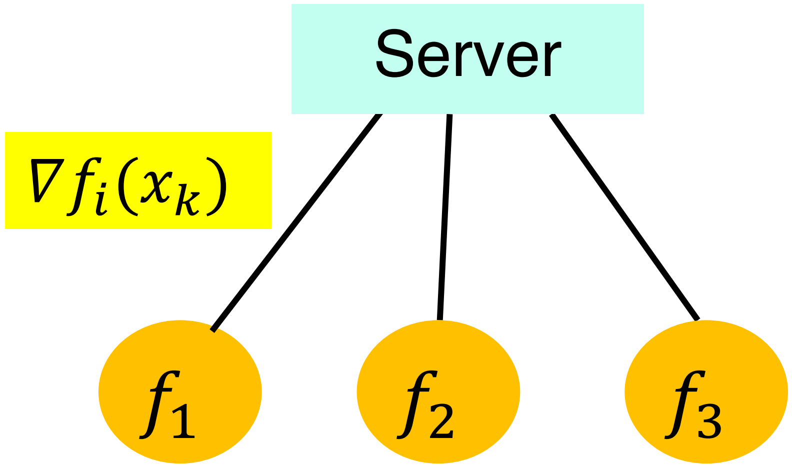
- Client i
  - Download x_k from server
  - Upload gradient $\nabla f_i(x_k)$



Client-Server Architecture

- Server maintains estimate x_k
- Client i knows $f_i(x)$

In iteration k+1



- Client i
  - Download x_k from server
  - Upload gradient $\nabla f_i(x_k)$

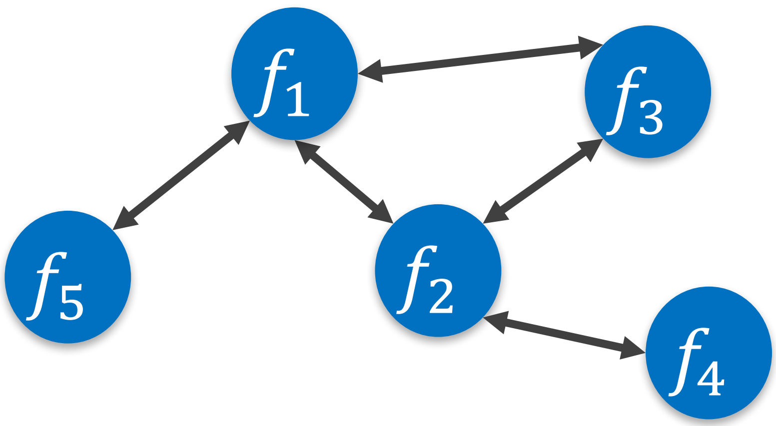
- Server

$$x_{k+1} \leftarrow x_k - \alpha_k \sum_i \nabla f_i(x_k)$$

Variations

- Stochastic
- Asynchronous
- …

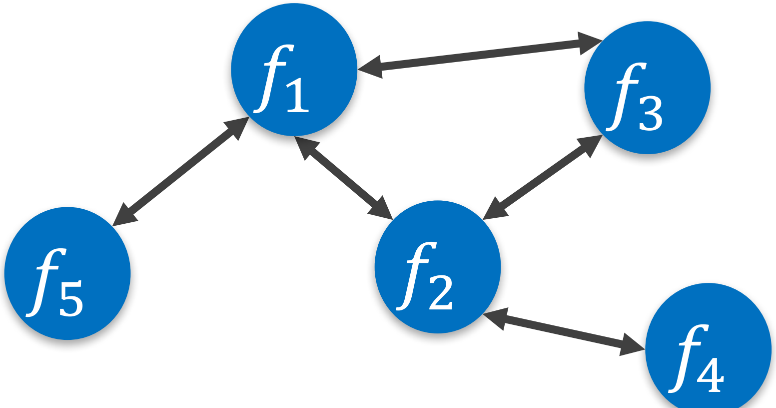
Peer-to-Peer Architecture



Peer-to-Peer Architecture

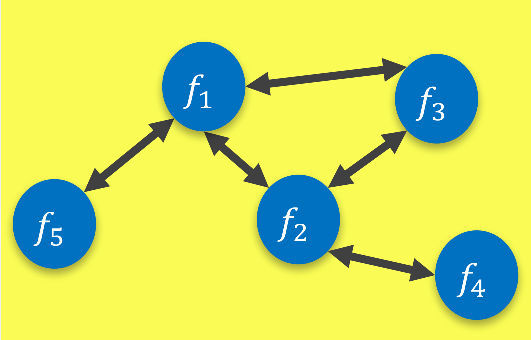
- Each agent maintains local estimate x
- Consensus step with neighbors
- Apply own gradient to own estimate

$$x_{k+1} \leftarrow x_k - \alpha_k \nabla f_i(x_k)$$

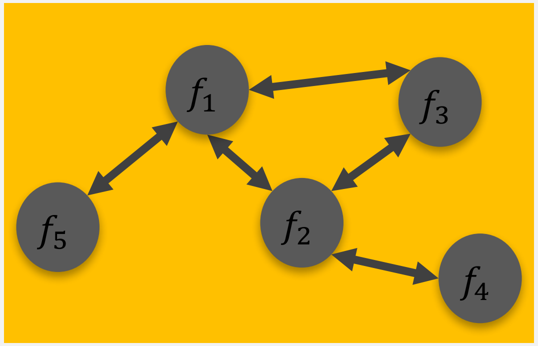


Outline

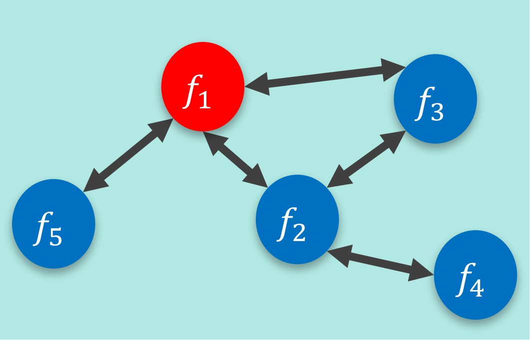
$$\underset{i}{argmin} \sum f_i(x)$$



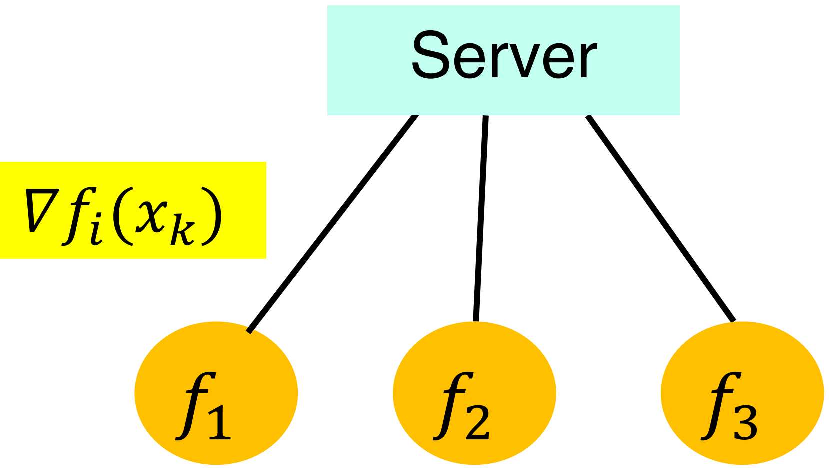
Distributed Optimization

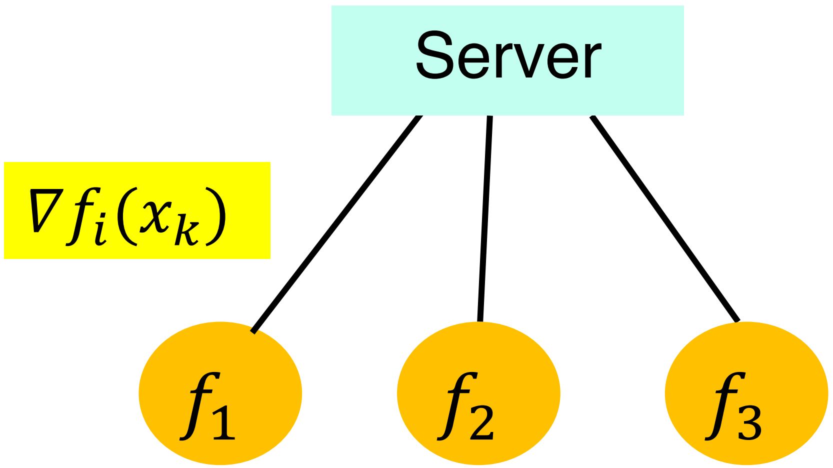


Privacy

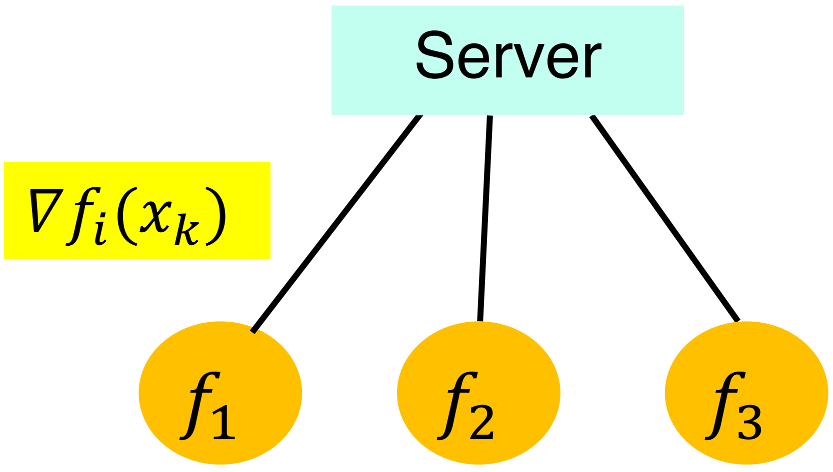


Fault-tolerance


Server
$\nabla f_i(x_k)$
f_1
f_2
f_3



Server observes gradients → privacy compromised



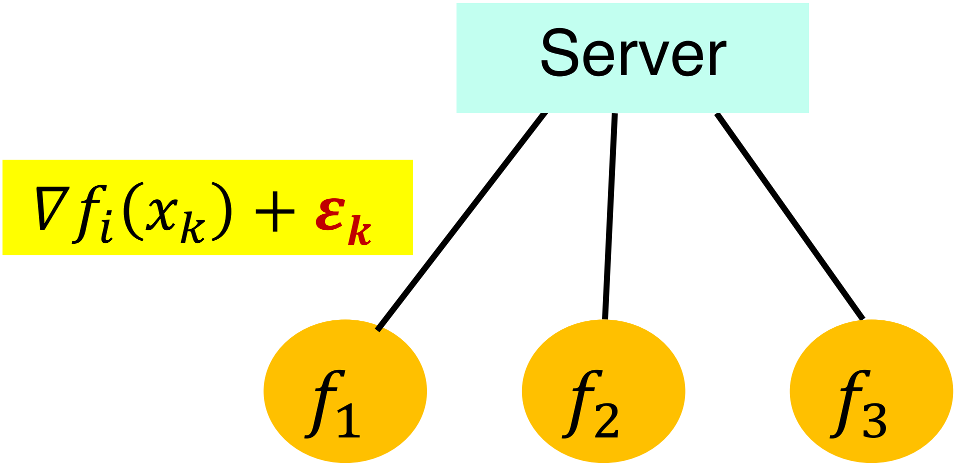
Server observes gradients ➔ privacy compromised

Achieve privacy and yet collaboratively optimize

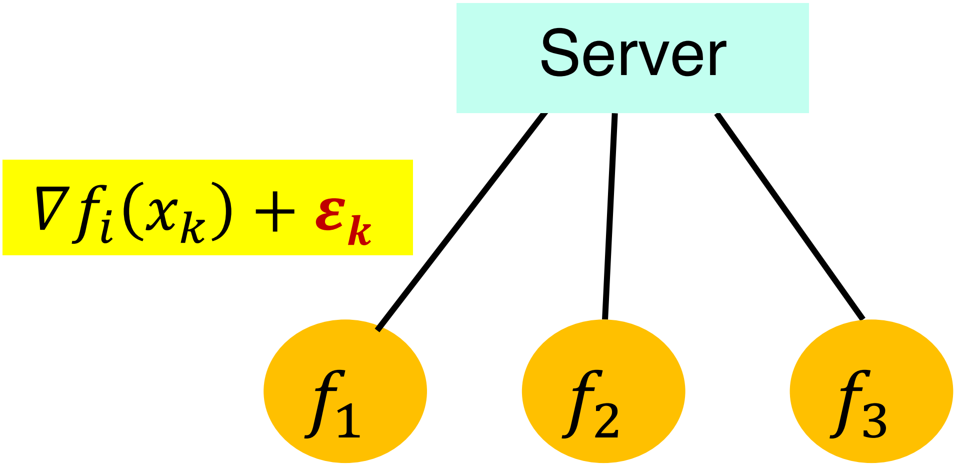
Related Work

- Cryptographic methods (homomorphic encryption)
- Function transformation
- Differential privacy

Differential Privacy



Differential Privacy



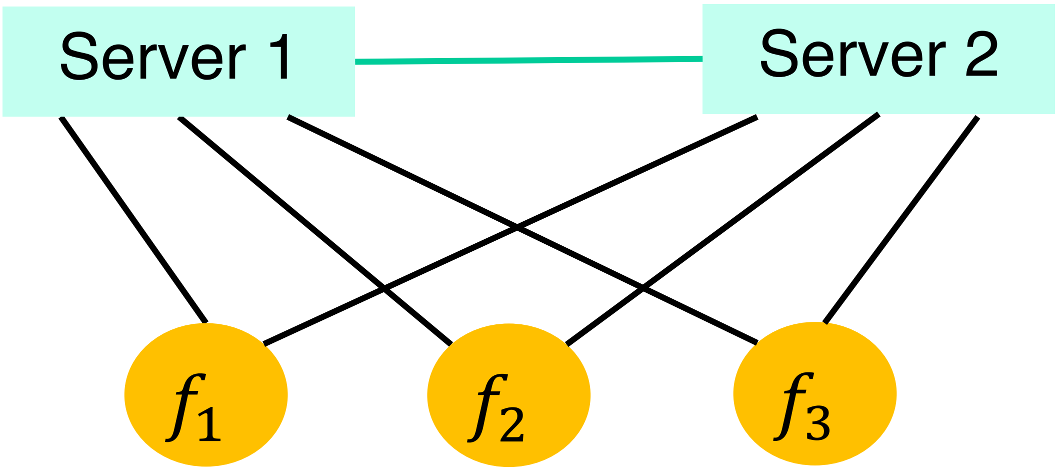
Trade-off privacy with accuracy

Proposed Approach

- Motivated by secret sharing

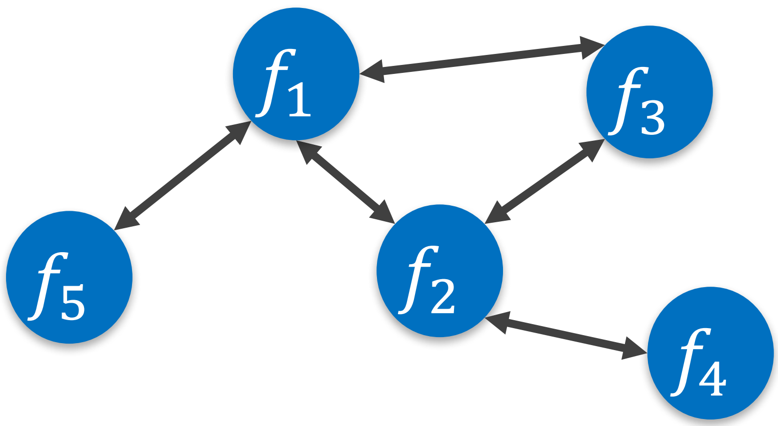
- Exploit diversity … Multiple servers / neighbors

Proposed Approach



Privacy if subset of servers adversarial

Proposed Approach



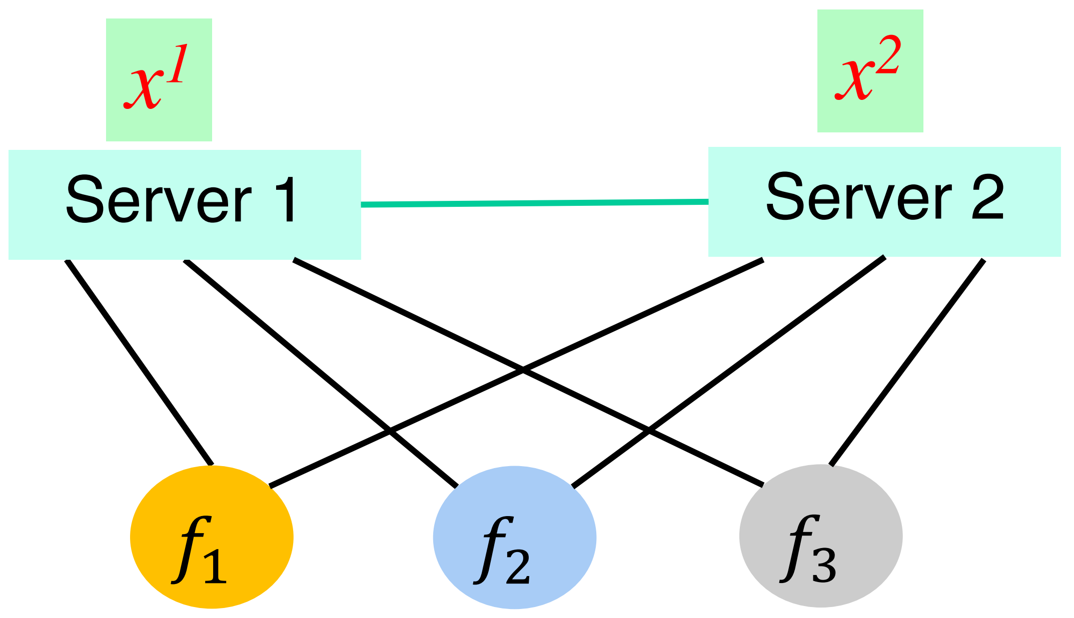
Privacy if subset of neighbors adversarial

Proposed Approach

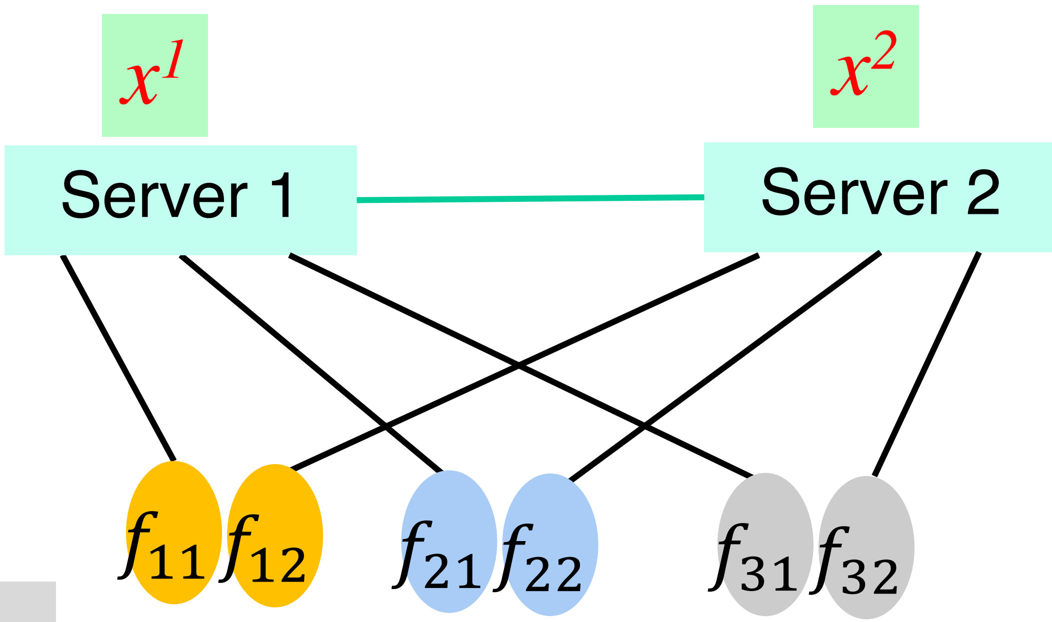
- Structured noise that

  “cancels” over servers/neighbors

Intuition

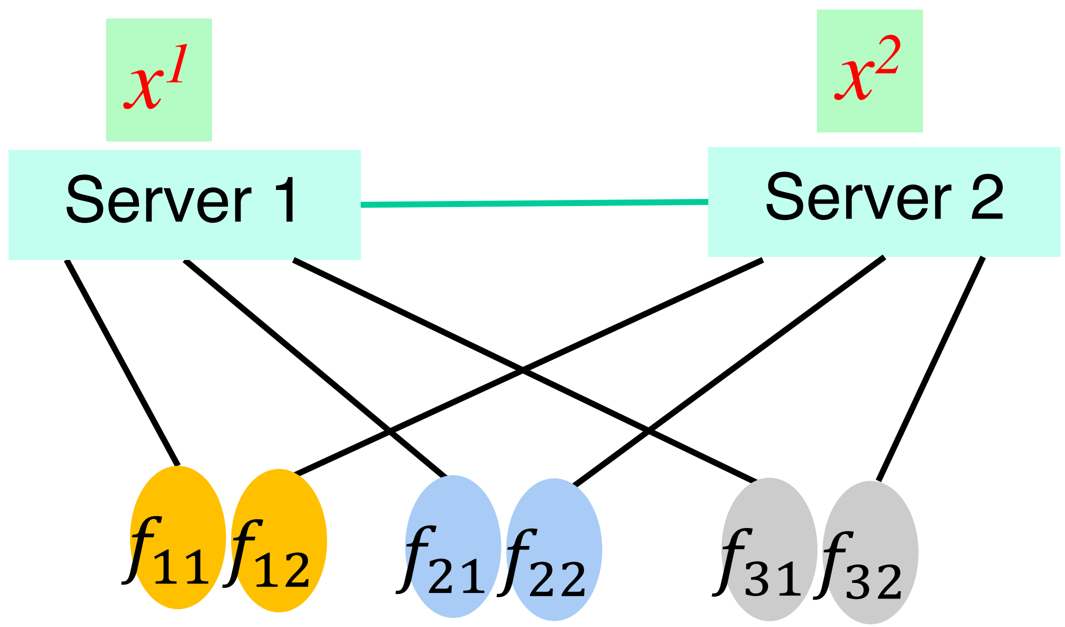


Intuition



Each client simulates multiple clients

Intuition



$$f_{11}(x) + f_{12}(x) = f_1(x)$$

$f_{ij}(x)$ not necessarily convex

Algorithm

- Each server maintains an estimate

In each iteration

- Client i
  - Download estimates from corresponding server
  - Upload gradient of f_i

- Each server updates estimate using received gradients

Algorithm

- Each server maintains an estimate

In each iteration

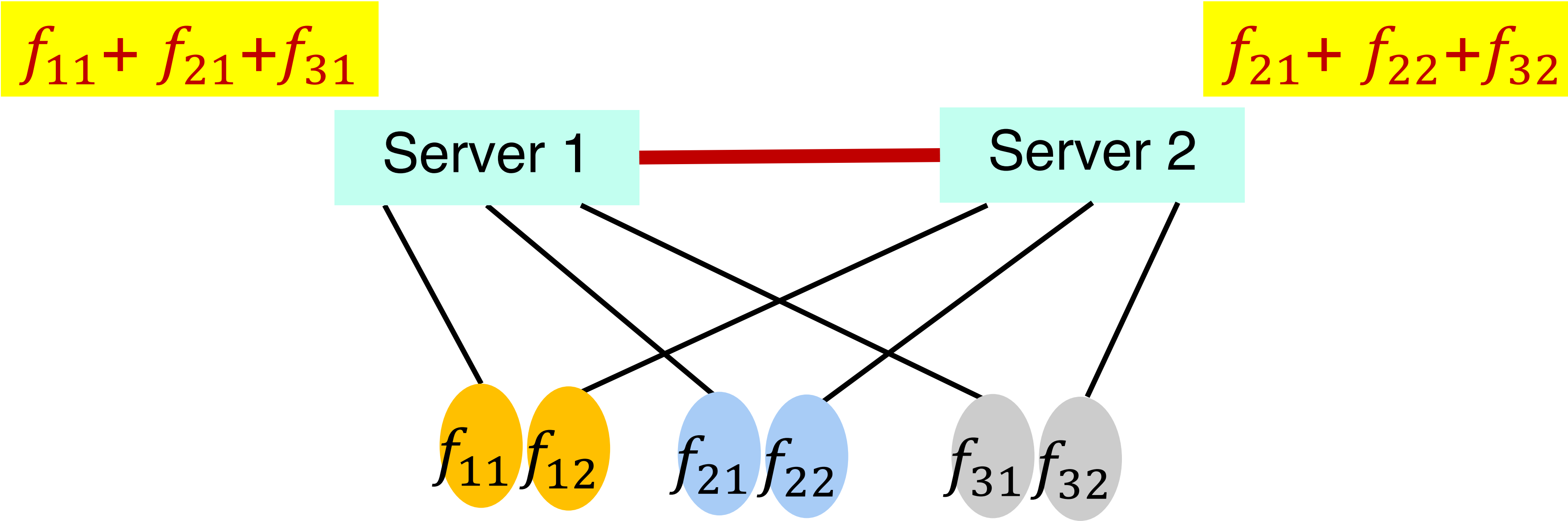
- Client i
  - Download estimates from corresponding server
  - Upload gradient of f_i
- Each server updates estimate using received gradients
- Servers periodically exchange estimates to perform a consensus step

Claim

- Under suitable assumptions, servers eventually reach consensus in

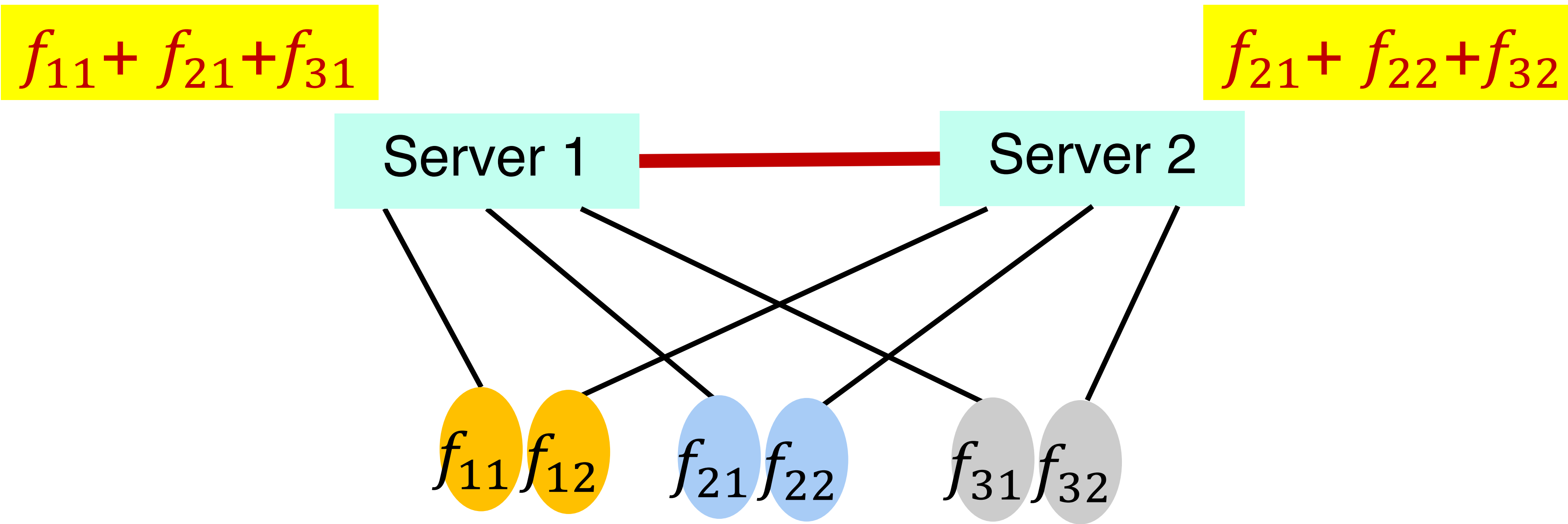
$$argmin \sum_{i} f_i(x)$$

Privacy



$f_{11}+f_{21}+f_{31}$
$f_{21}+f_{22}+f_{32}$
Server 1
Server 2
f_{11} f_{12} f_{21} f_{22} f_{31} f_{32}

Privacy



- Server 1 may learn f_{11}, f_{21}, f_{31}, $f_{21}+f_{22}+f_{32}$
- Not sufficient to learn f_i

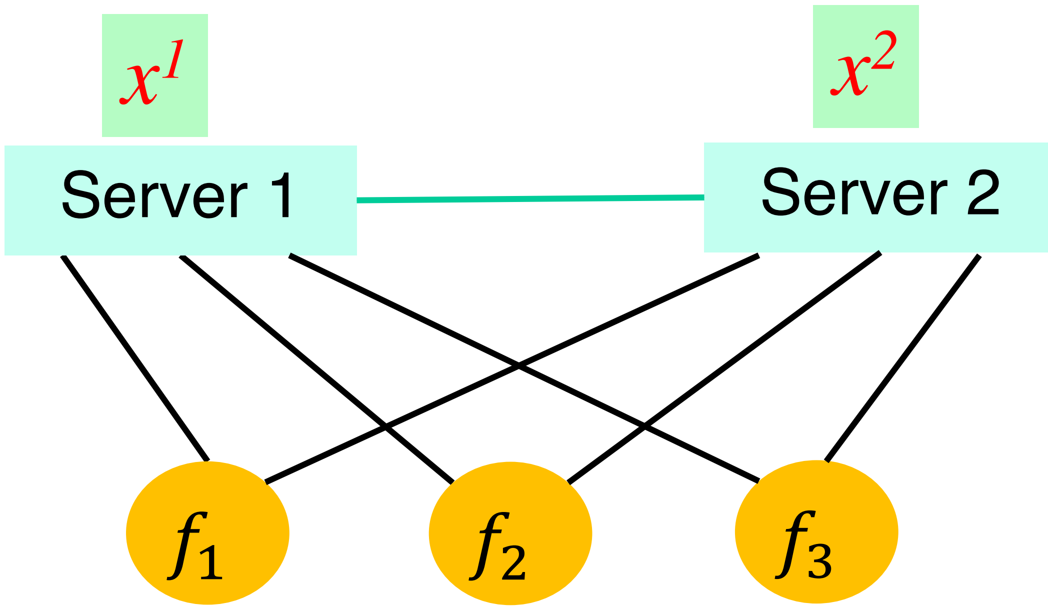
$$f_{11}(x) + f_{12}(x) = f_{1}(x)$$

- *Function splitting* not necessarily practical
- Structured randomization as an alternative

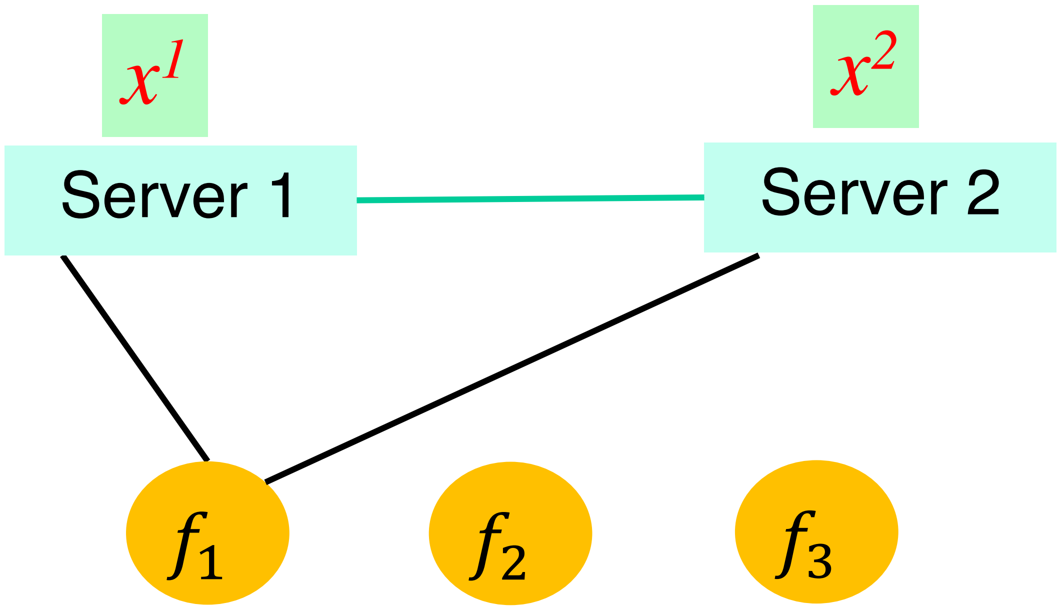
Structured Randomization

- Multiplicative or additive noise in gradients
- Noise *cancels* over servers

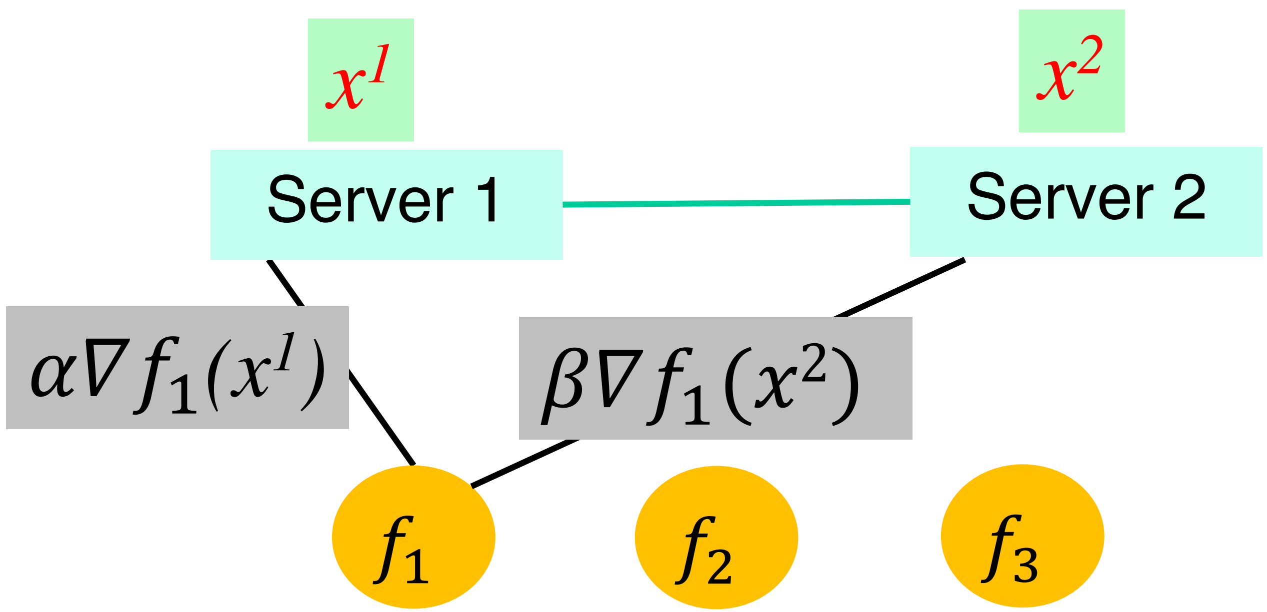
Multiplicative Noise



Multiplicative Noise

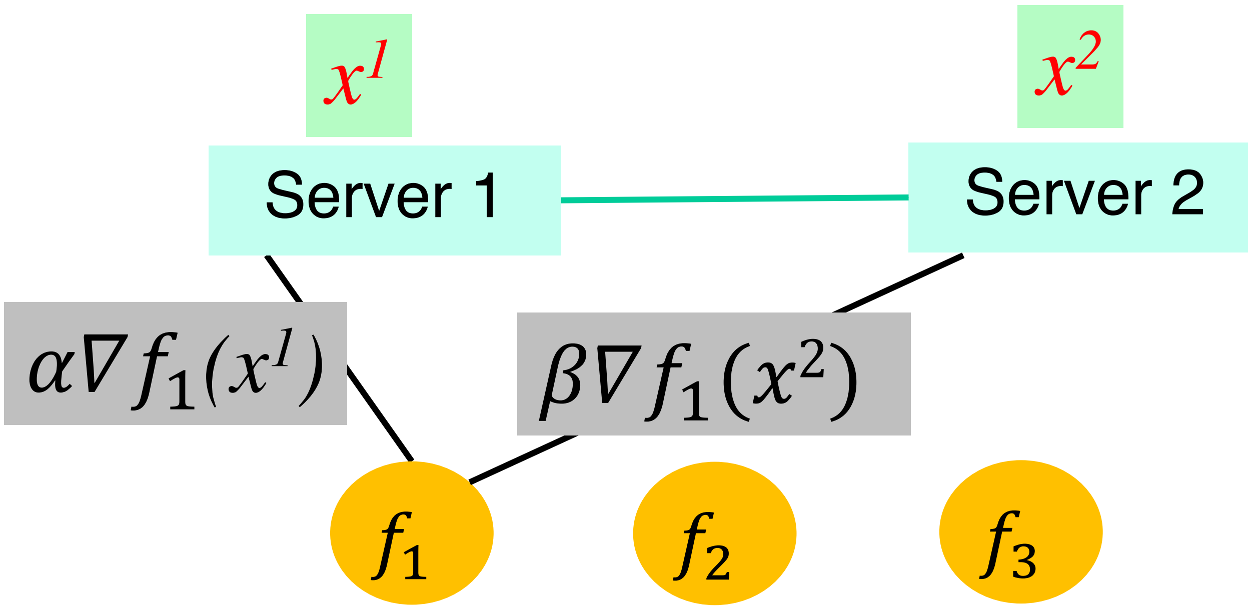


Multiplicative Noise



$\alpha + \beta = 1$

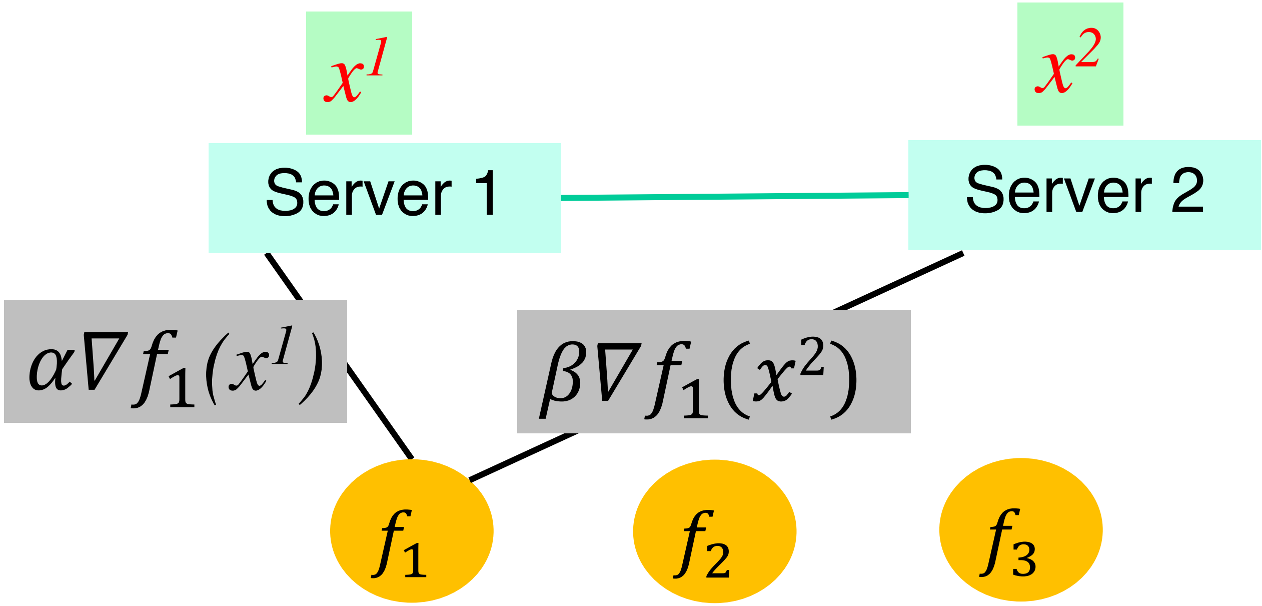
Multiplicative Noise



$\alpha + \beta = 1$

Suffices for this invariant to hold over a larger number of iterations

Multiplicative Noise



$\alpha + \beta = 1$

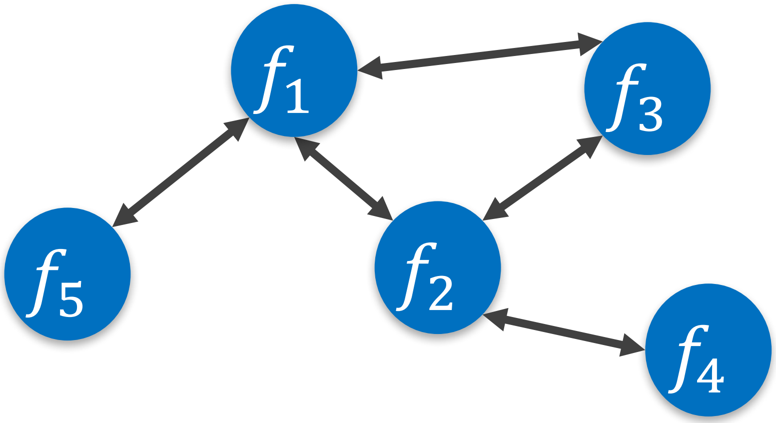
Noise from client *i* to server *j* *not* zero-mean

Claim

- Under suitable assumptions, servers eventually reach consensus in

$$argmin \sum_i f_i(x)$$

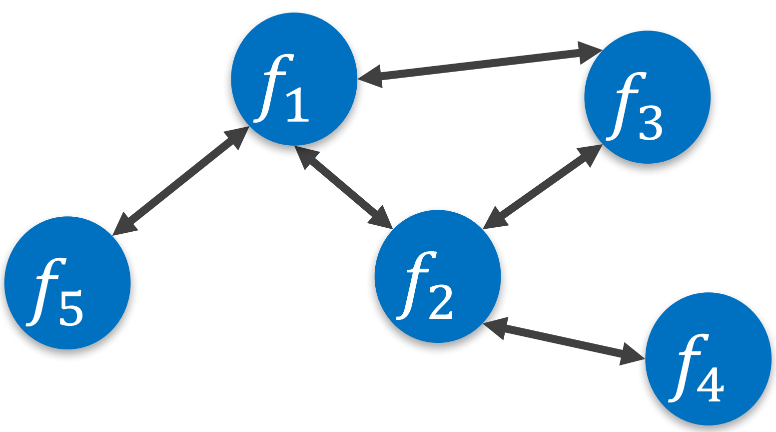
Peer-to-Peer Architecture



Reminder …

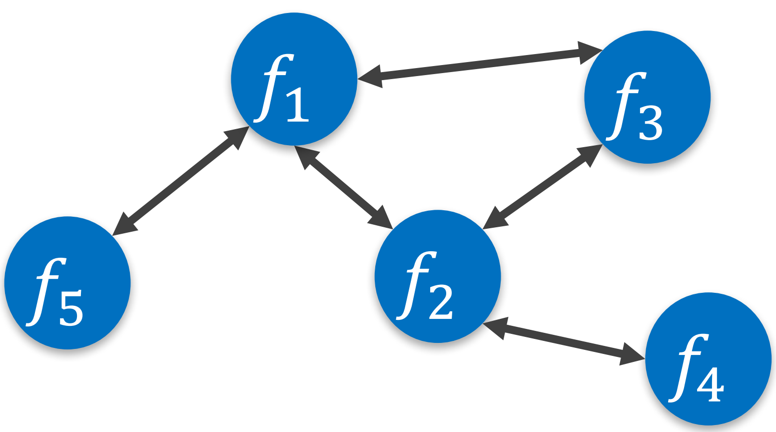
- Each agent maintains local estimate x
- Consensus step with neighbors
- Apply own gradient to own estimate

$$x_{k+1} \leftarrow x_k - \alpha_k \nabla f_i(x_k)$$



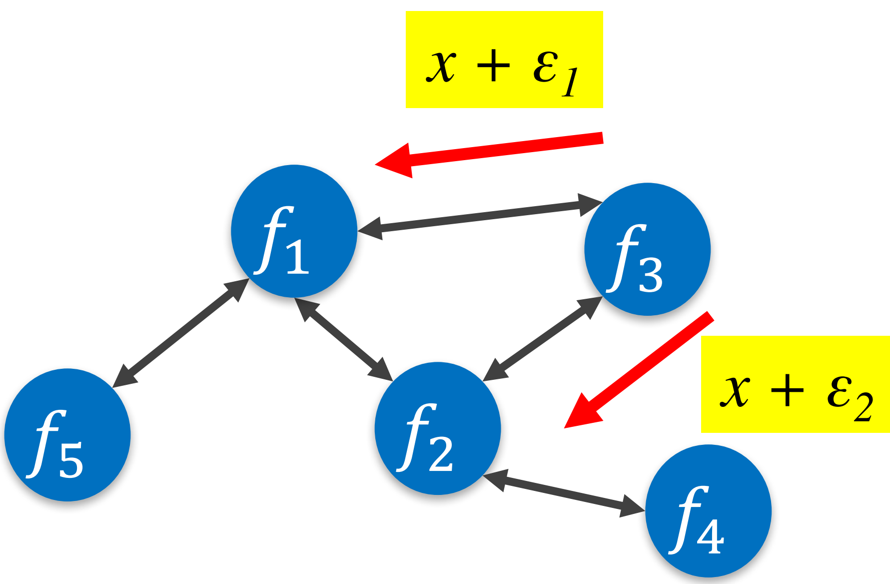
Proposed Approach

- Each agent shares noisy estimate with neighbors
  - Scheme 1 – Noise cancels over neighbors
  - Scheme 2 – Noise cancels network-wide



Proposed Approach

- Each agent shares noisy estimate with neighbors
  - Scheme 1 – Noise cancels over neighbors
  - Scheme 2 – Noise cancels network-wide



$\varepsilon_1 + \varepsilon_2 = 0$ (over iterations)

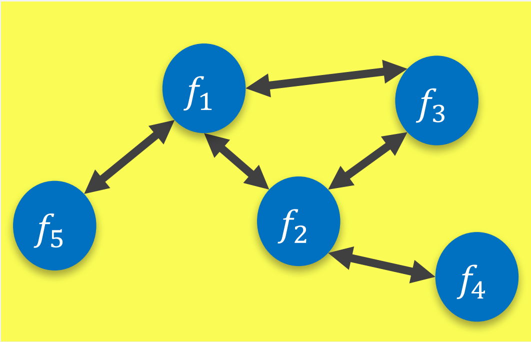
Peer-to-Peer Architecture

- Poster today

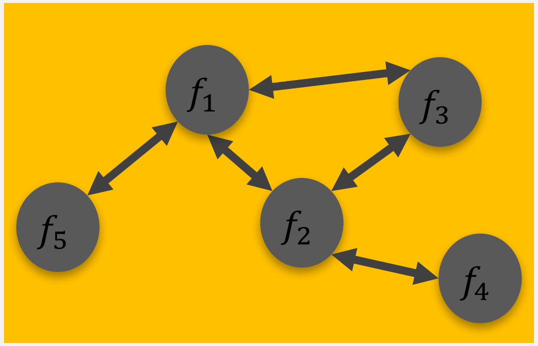
Shripad Gade

Outline

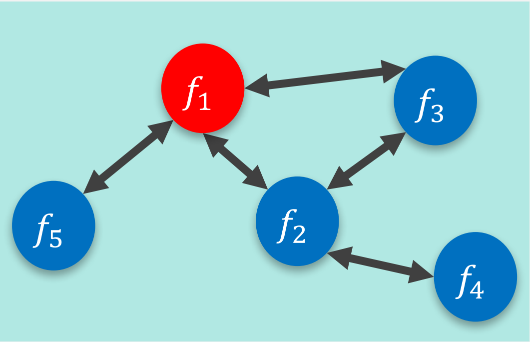
$$argmin \sum_{i} f_i(x)$$



Distributed Optimization



Privacy



Fault-tolerance

Fault-Tolerance

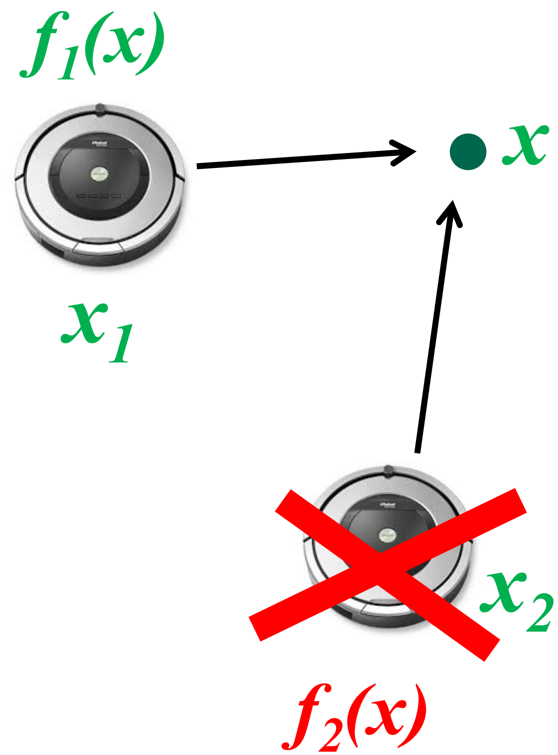
- Some agents may be faulty
- Need to produce "correct" output despite the faults

Byzantine Fault Model

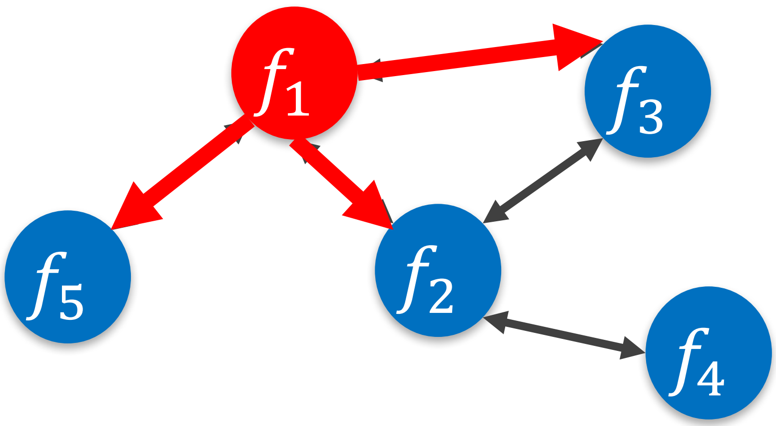
- No constraint on misbehavior of a faulty agent
- May send bogus messages
- Faulty agents can collude

Peer-to-Peer Architecture

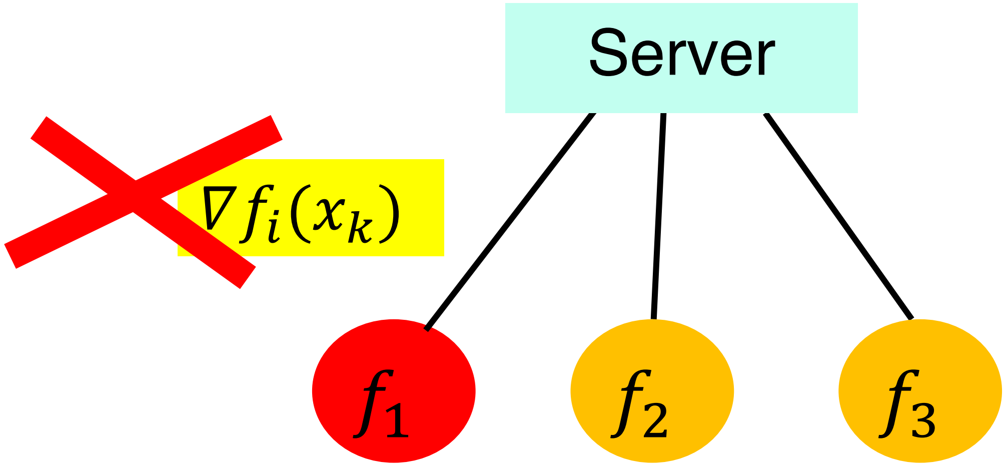
- $f_i(x)$ = cost for robot i to go to location x
- Faulty agent may choose arbitrary cost function



Peer-to-Peer Architecture



Client-Server Architecture



Fault-Tolerant Optimization

- The original problem is not meaningful

$$argmin \sum_i f_i(x)$$

Fault-Tolerant Optimization

- The original problem is not meaningful

$$argmin \sum_{i} f_i(x)$$

- Optimize cost over only non-faulty agents

$$argmin \sum_{i\ good} f_i(x)$$

Fault-Tolerant Optimization

- The original problem is not meaningful

$$argmin \sum_{i} f_i(x)$$

- Optimize cost over only non-faulty agents

Impossible!

$$argmin \sum_{i\ good} f_i(x)$$

Fault-Tolerant Optimization

- Optimize weighted cost over only non-faulty agents

$$\operatorname{argmin} \sum_{i\ good} f_i(x)\, \boldsymbol{\alpha}_i$$

- With $\boldsymbol{\alpha}_i$ as close to 1/ good as possible

Fault-Tolerant Optimization

- Optimize weighted cost over only non-faulty agents

$$\text{argmin} \sum_{i \; good} f_i(x) \, \boldsymbol{\alpha}_i$$

With t Byzantine faulty agents:

t weights may be 0

Fault-Tolerant Optimization

- Optimize weighted cost over only non-faulty agents

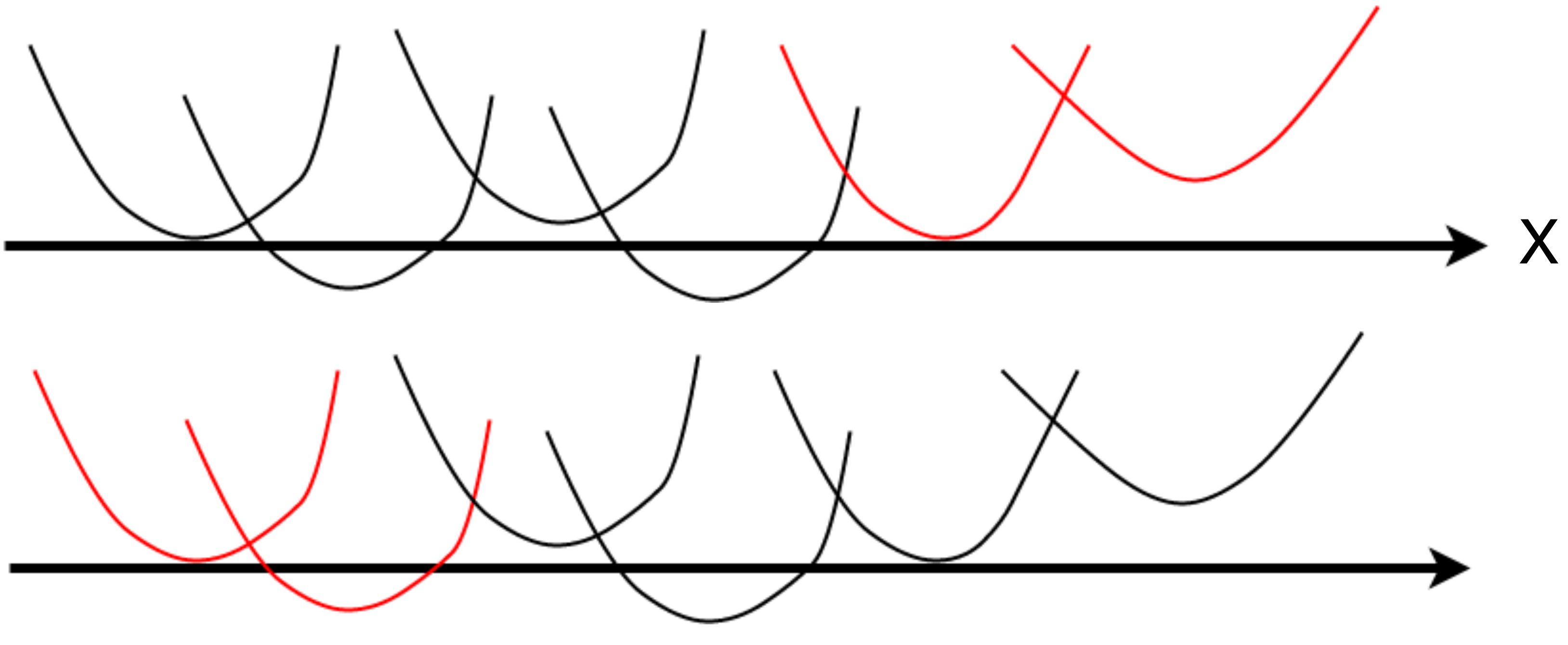
$$\underset{i\ good}{argmin \sum} f_i(x)\,\boldsymbol{\alpha}_i$$

t Byzantine agents, n total agents

At least n-2t weights guaranteed to be > 1/2(n-t)

Centralized Algorithm

- Of the n agents, any t may be faulty
- How to filter cost functions of faulty agents?



Centralized Algorithm: Scalar argument *x*

Define a virtual function G(x) whose gradient is obtained as follows

Centralized Algorithm: Scalar argument *x*

Define a virtual function G(*x*) whose gradient is obtained as follows

At a given *x*

- Sort the gradients of the n local cost functions

Centralized Algorithm: Scalar argument x

Define a virtual function G(x) whose gradient is obtained as follows

At a given x

- Sort the gradients of the n local cost functions
- Discard smallest t and largest t gradients

Centralized Algorithm: Scalar argument *x*

Define a virtual function G(*x*) whose gradient is obtained as follows

At a given *x*

- Sort the gradients of the n local cost functions
- Discard smallest t and largest t gradients
- Mean of remaining gradients = Gradient of G at *x*

Centralized Algorithm: Scalar argument *x*

Define a virtual function G(*x*) whose gradient is obtained as follows

At a given *x*

- Sort the gradients of the n local cost functions
- Discard smallest t and largest t gradients
- Mean of remaining gradients = Gradient of G at *x*

Virtual function G(*x*) is convex

Centralized Algorithm: Scalar argument *x*

Define a virtual function G(*x*) whose gradient is obtained as follows

At a given *x*

- Sort the gradients of the n local cost functions
- Discard smallest t and largest t gradients
- Mean of remaining gradients = Gradient of G at *x*

Virtual function G(*x*) is convex → Can optimize easily

Peer-to-Peer Fault-Tolerant Optimization

- Gradient filtering similar to centralized algorithm

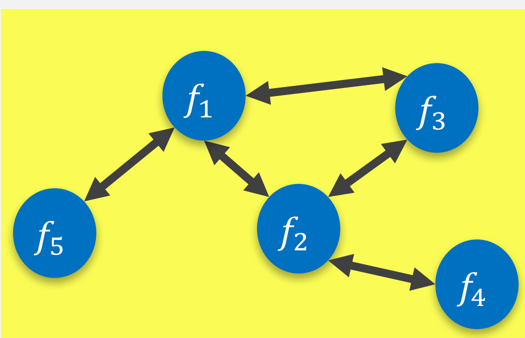
  … require “rich enough” connectivity
  … correlation between functions helps

- Vector case harder

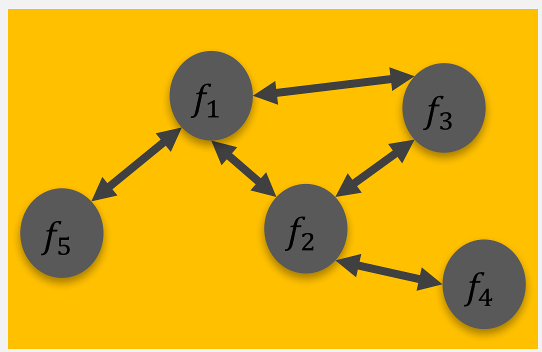
  … redundancy between functions helps

Summary

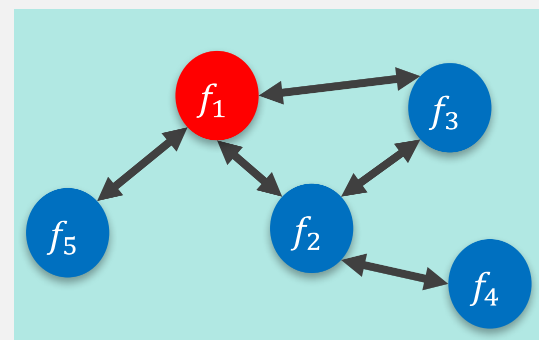
$$argmin \sum_{i} f_i(x)$$



Distributed Optimization



Privacy



Fault-tolerance

Thanks!

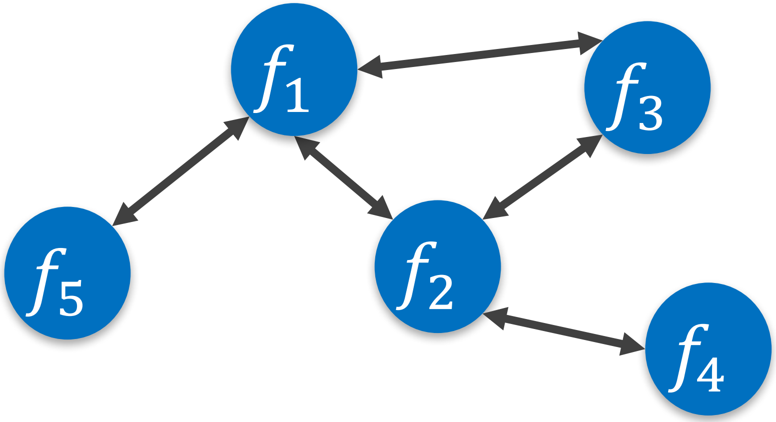
disc.ece.illinois.edu

Distributed Peer-to-Peer Optimization

- Each agent maintains local estimate x

In each iteration

- Compute weighted average with neighbors' estimates



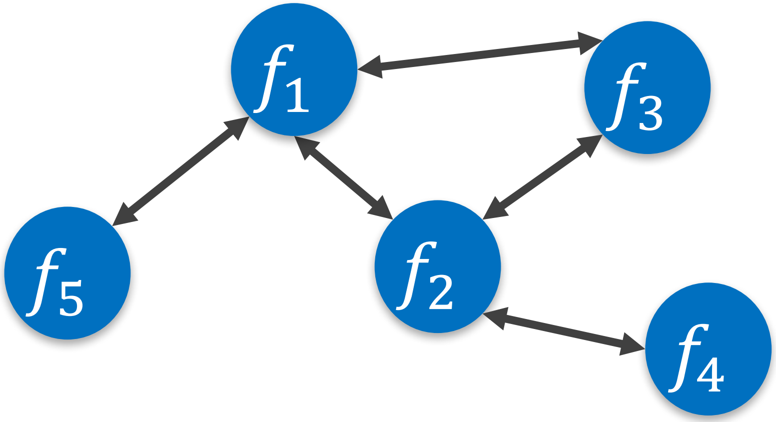
Distributed Peer-to-Peer Optimization

- Each agent maintains local estimate x

In each iteration

- Compute weighted average with neighbors' estimates
- Apply own gradient to own estimate

$$x_{k+1} \leftarrow x_k - \alpha_k \nabla f_i(x_k)$$



Distributed Peer-to-Peer Optimization

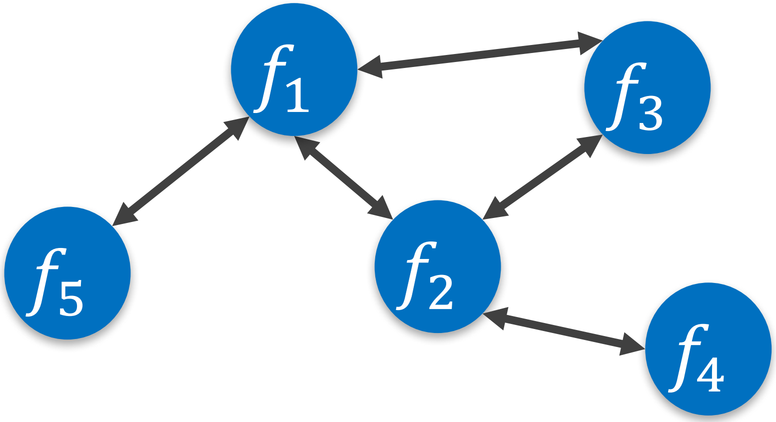
- Each agent maintains local estimate x

In each iteration

- Compute weighted average with neighbors' estimates
- Apply own gradient to own estimate

$$x_{k+1} \leftarrow x_k - \alpha_k \nabla f_i(x_k)$$

- Local estimates converge to $argmin \sum_i f_i(x)$



RSS – Locally Balanced

Perturbations

- Add to zero (locally per node)
- Bounded ($\leq \Delta$)

Algorithm

- Node j selects $\mathrm{d}_{\mathrm{k}}^{\mathrm{j,i}}$ such that $\sum_{\mathrm{i}} \mathrm{d}_{\mathrm{k}}^{\mathrm{j,i}} = 0$ and $\left|d_k^{j,i}\right| \leq \Delta$
- Share $\mathrm{w}_{\mathrm{k}}^{\mathrm{j,i}} = \mathrm{x}_{\mathrm{k}}^{\mathrm{j}} + \mathrm{d}_{\mathrm{k}}^{\mathrm{j,i}}$ with node i
- Consensus and (Stochastic) Gradient Descent

RSS – Network Balanced

Perturbations

- Add to zero (over network)
- Bounded ($\leq \Delta$)

Algorithm

- Node j computes perturbation d_k^j
  - sends $s^{j,i}$ to i
  - add received $s^{i,j}$ and subtract sent $s^{j,i}$ $\Rightarrow d_k^j = \sum \text{rcvd} - \sum \text{sent}$
- Obfuscate state $w_k^j = x_k^j + d_k^j$ shared with neighbors
- Consensus and (Stochastic) Gradient Descent

Convergence

Let $\hat{x}^j{}_T = \sum^T \alpha_k \, x^j_k \,/ \sum^T \alpha_k$ and $\alpha_k = 1/\sqrt{k}$

$$f\left(\hat{x}^j{}_T\right) - f(x^*) \leq \mathcal{O}\left(\frac{\log(T)}{\sqrt{T}}\right) + \mathcal{O}\left(\frac{\Delta^2 \log(T)}{\sqrt{T}}\right)$$

- Asymptotic convergence of iterates to optimum
- Privacy-Convergence Trade-off

- Stochastic gradient updates work too

Function Sharing

- Let $f_i(x)$ be bounded degree polynomials

Algorithm

- Node j shares $s^{j,i}(x)$ with node i
- Node j obfuscates using $p_j(x) = \sum s^{i,j}(x) - \sum s^{j,i}(x)$
- Use $\hat{f}_j(x) = f_j(x) + p_j(x)$ and use distributed gradient descent

Function Sharing - Convergence

- Function Sharing iterates converge to correct optimum ($\sum \hat{f}_i(x) = f(x)$)

- Privacy:

If vertex connectivity of graph $\geq$ f then no group of f nodes can estimate true functions f_i (or any good subset)

- $p_j(x)$ is also similar to $f_j(x)$ then it can hide $f_i(x)$ well