Distributed and Privacy-Preserving Data Analytics

BSI SIF

Davide Frey
WIDE Team – Inria Rennes

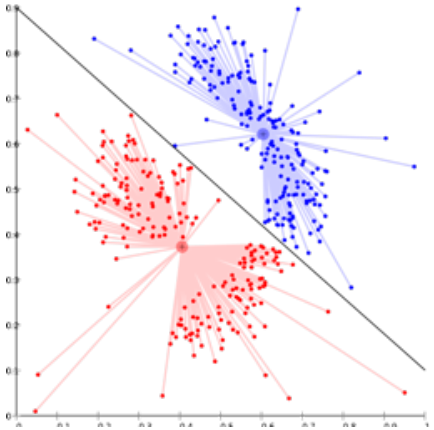
Outline

- Cloud Architectures for Distributed Data Analytics
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

Outline

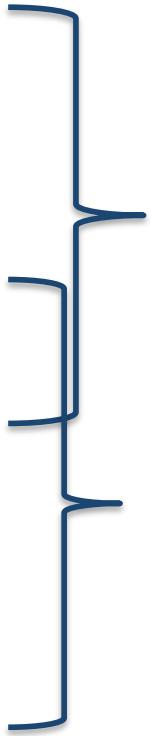
- Cloud Architectures for Distributed Data Analytics
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

Decentralized Data Analytics



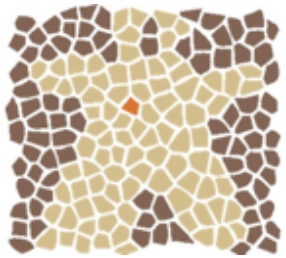
Why Distribute Computation

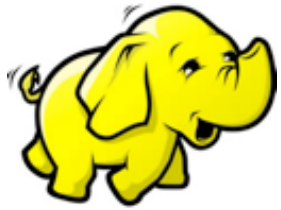
- Speed/Parallelization
- Scale
- Privacy / Cost /Energy



- Parallelize for Performance
- Decentralize for Simplicity

Are we Already Done?





GraphLab

Outline

- Cloud Architectures for Distributed Data Analytics
  - Map Reduce / Hadoop
  - Data Parallelism vs Model Parallelism
  - Google's Parameter Server
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

Outline

- Cloud Architectures for Distributed Data Analytics
  - Map Reduce / Hadoop
  - Data Parallelism vs Model Parallelism
  - Google's Parameter Server
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

MapReduce Example: G-Means

G-Means as a collection of map-reduce jobs

Algorithm 5 TestFewClusters Mapper

Input: *point* (text)
Output: *vectorid* (int) ⇒ A^{*2} (double)

procedure SETUP
  Build vectors from center pairs
  Read centers from previous iteration
end procedure

procedure MAP(*key*, *point*)
  Find nearest *center*
  Find corresponding *vector*
  Compute *projection* of *point* on *vector*
  Add *projection* to *list vectorid*
end procedure

procedure CLOSE
  for Each *list* **do**
    Read projections to build a *vector*
    Normalize *vector* (mean 0 , stddev 1)
    Compute $A^{*2} = adtest(vector)$
    Emit($vectorid \Rightarrow A^{*2}$)
  end for
end procedure

Algorithm 3 TestClusters Mapper

Input: *point* (text)
Output: *vectorid* (int) ⇒ *projection* (double)

procedure SETUP
  Build vectors from center pairs
  Read centers from previous iteration
end procedure

procedure MAP(*key*,
  Find nearest *cente*
  Find correspondin
  Compute *projectio*
  Emit(*vectorid*, *proj*
end procedure

Algorithm 2 KMeansAndFindNewCenters Mapper

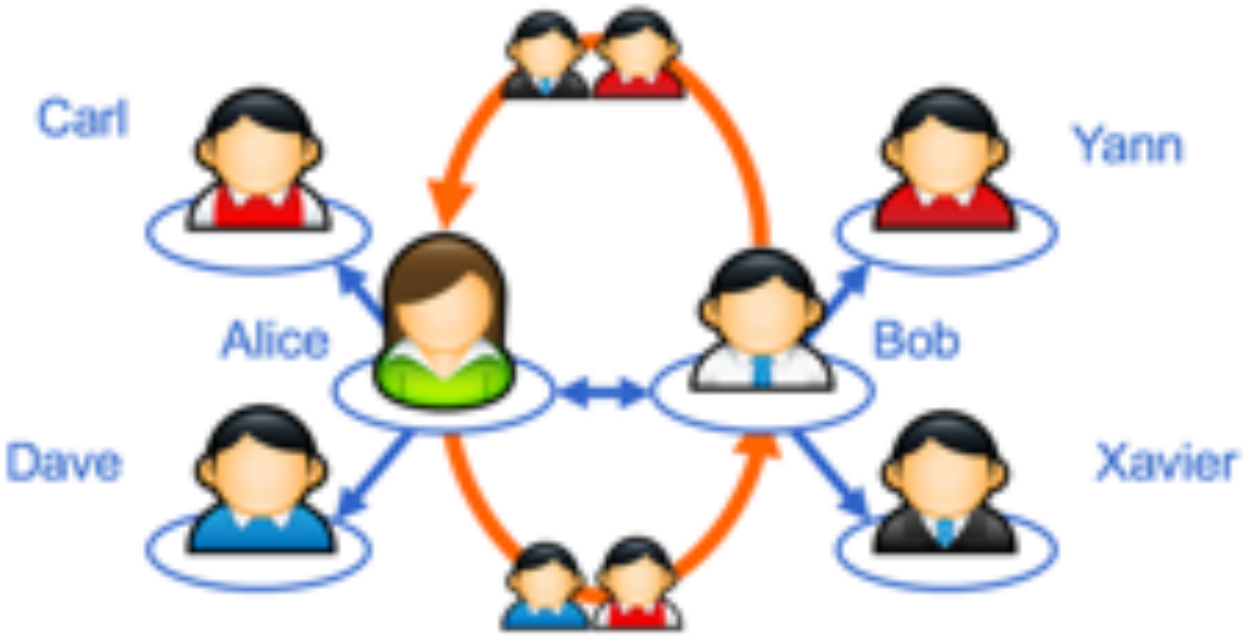
Input: point (text)
Output:
centerid (long) ⇒ coordinates (float[]), 1 (int)
centerid + *OFFSET* (long) ⇒ coordinates (float[]), 1 (int)

procedure MAP(*key*, *point*)
  Find nearest *center*
  Emit(*centerid*, *point*)
  Emit(*centerid* + *OFFSET*, *point*)
end procedure

[Deb14a] Thibault Debatty, Pietro Michiardi, Olivier Thonnard, Wim Mees. Determining the k in k-means with MapReduce. In Proc. of BeyondMR 2014.

Scalable KNN computation

Exploit greedy solutions



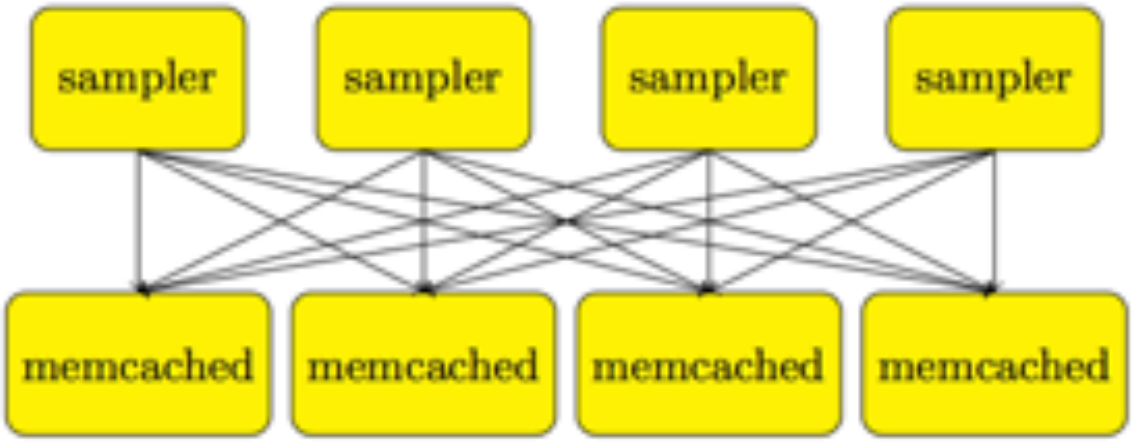
[Don11] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In Proceedings of the 20th international conference on World wide web (WWW '11). ACM, New York, NY, USA, 577-586.

Outline

- Cloud Architectures for Distributed Data Analytics
  - Map Reduce / Hadoop
  - Data Parallelism vs Model Parallelism
  - Google's Parameter Server
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

Data Parallelism: Parameter Servers

- Workers share common model
- Treat different portions on the data
- (Independently) update parameters

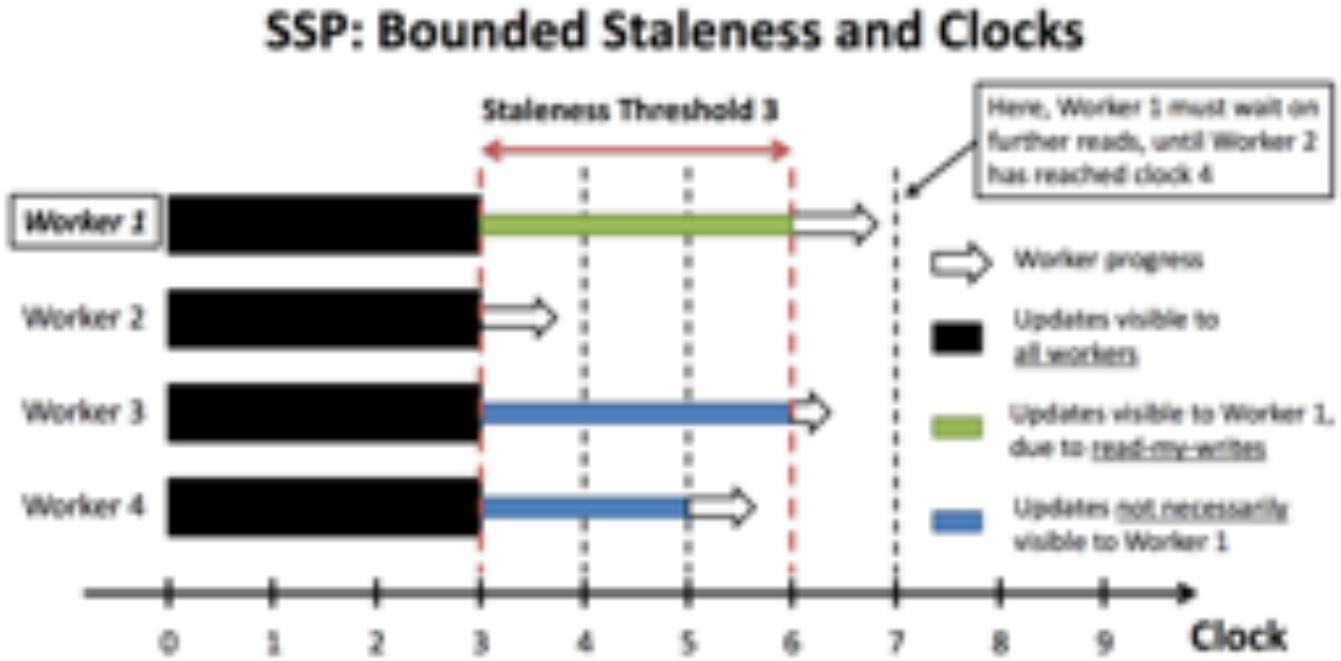


[Smo10] Alexander Smola and Shravan Narayanamurthy. 2010. An architecture for parallel topic models. Proc. VLDB Endow. 3, 1-2 (September 2010), 703-710.

Data Parallelism: Parameter Servers

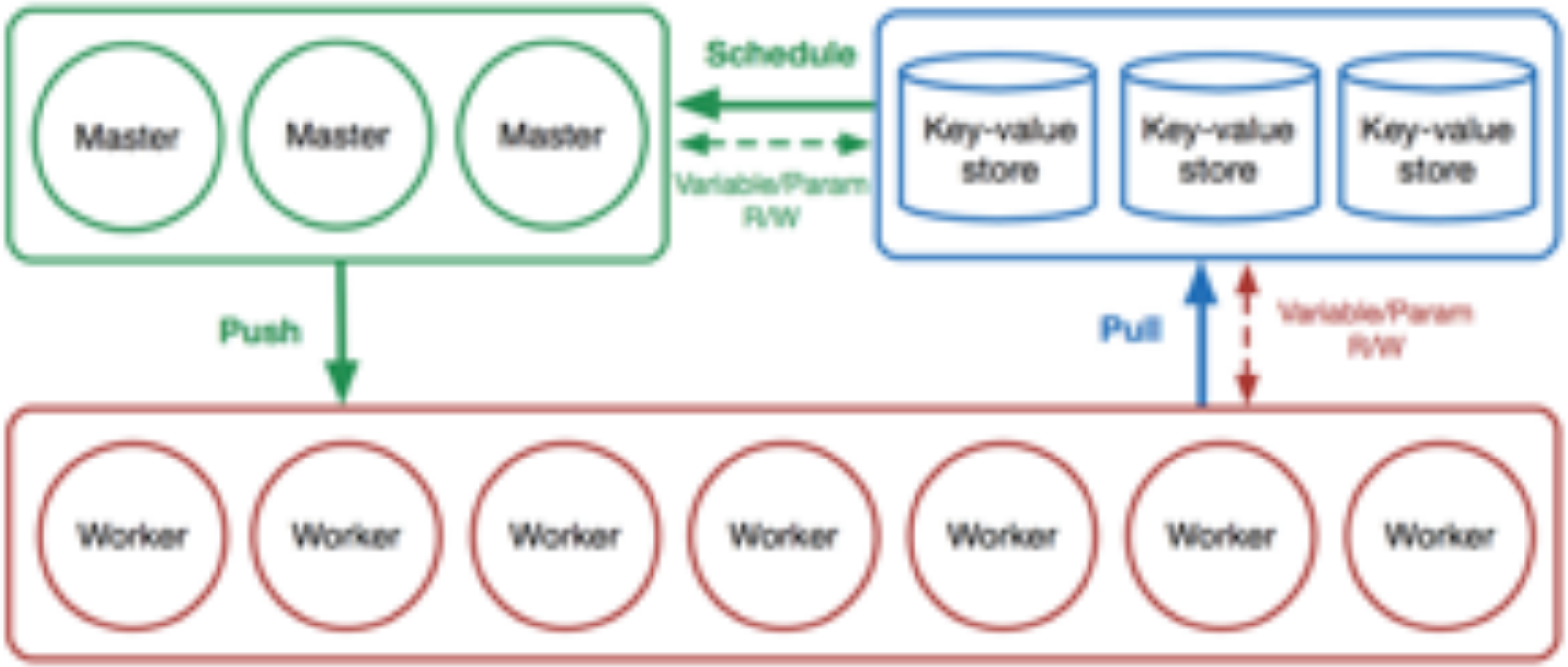
Stale Synchronous Parallel model

- Commutative associative parameter updates: $\theta \leftarrow \theta + \delta$



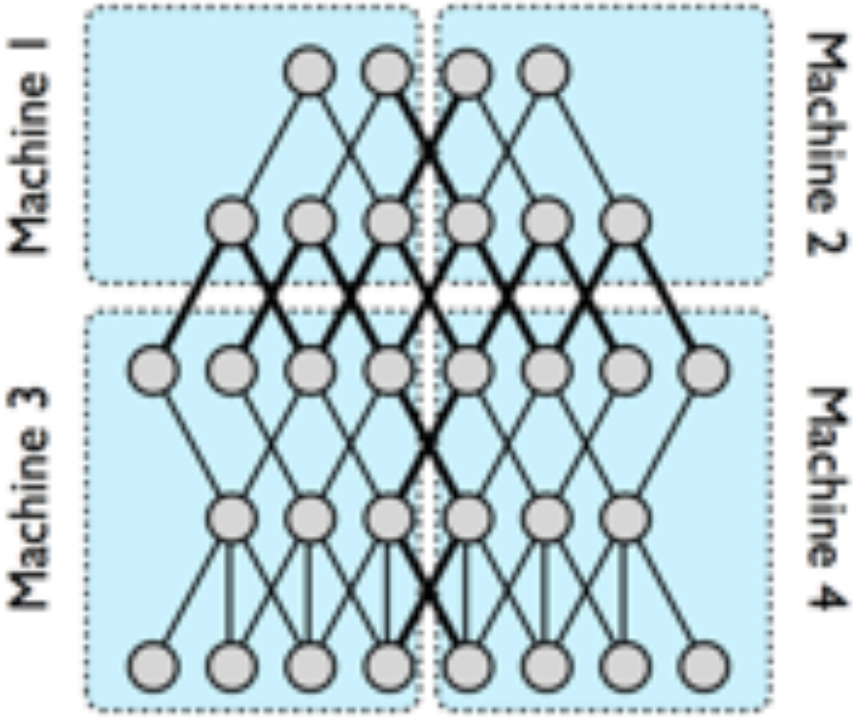
[ho14] Q. Ho, J. Cipar, H. Cui, J. Kim, S. Lee, P. B. Gibbons, G. Gibson, G. R. Ganger, and E. P. Xing. More effective distributed ML via a stale synchronous parallel parameter server. In NIPS, 2013.

Model Parallelism: STRADS



[Lee14] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A. Gibson, and Eric P. Xing. On Model Parallelization and Scheduling Strategies for Distributed Machine Learning. Neural Information Processing Systems, 2014 (NIPS 2014)

Model Parallelism: Google DistBelief



Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le and Andrew Y. Ng "Large Scale Distributed Deep Networks". Advances in Neural Information Processing Systems 2012.

Outline

- Cloud Architectures for Distributed Data Analytics
  - Map Reduce / Hadoop
  - Data Parallelism vs Model Parallelism
  - Google's Parameter Server
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

Scaling Distributed ML with the Parameter Server

- Lots of data -> 1TB to 1PB
- Large models -> large number of parameters 10^9 to 10^{12}
- Challenges even for distributed systems
  - Workers need to access all parameters
  - Many algorithms are sequential
  - Machines can be unreliable
- Slides based on paper (images extracted from paper)

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation* (OSDI'14). USENIX Association, Berkeley, CA, USA, 583-598.

Unreliable Machines

≈ #machine × time	# of jobs	failure rate
100 hours	13,187	7.8%
1,000 hours	1,366	13.7%
10,000 hours	77	24.7%

Led google to design new parameter server

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation* (OSDI'14). USENIX Association, Berkeley, CA, USA, 583-598.

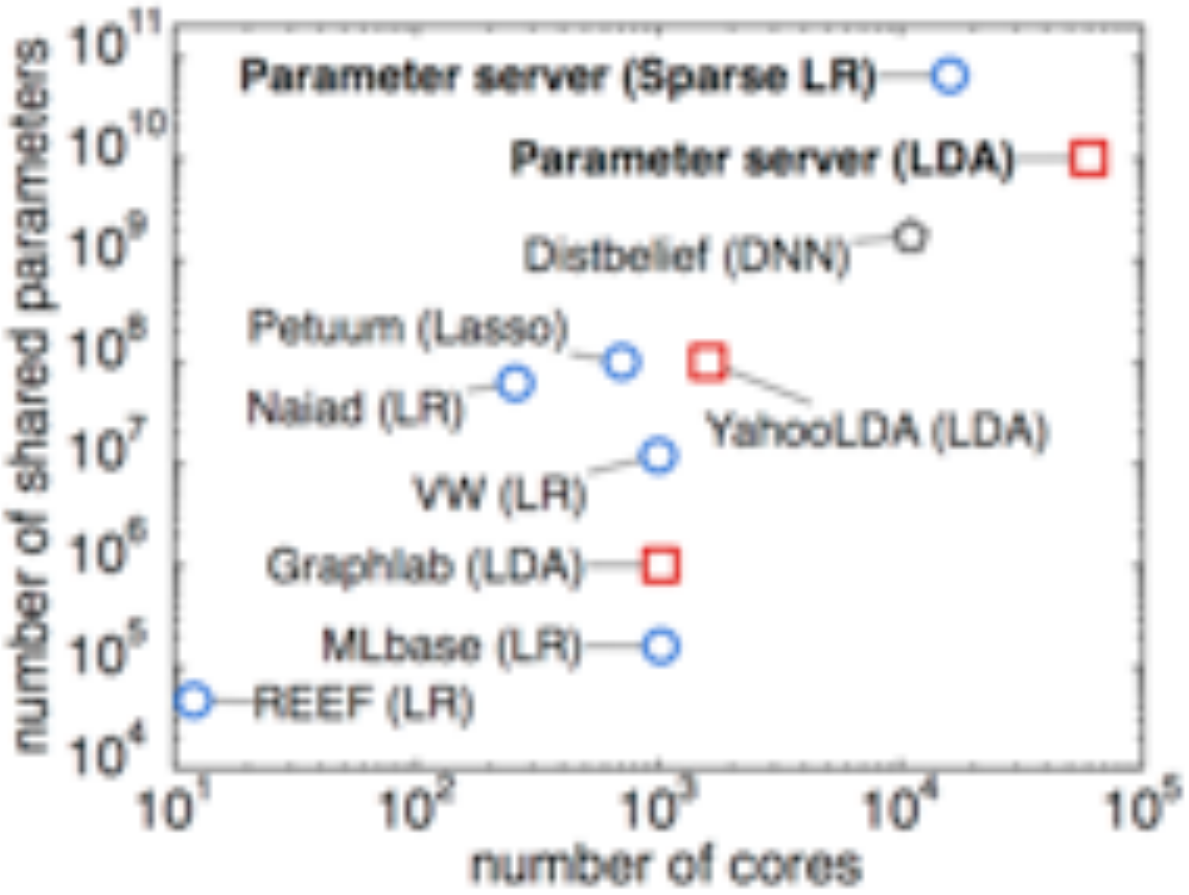
Requirements

- Efficient communication
- Flexible consistency models
- Elastic scalabilty
- Fault Tolerance and Durability
- Ease of Use

Architecture

- Multiple parameter server nodes
  - Each node maintaining subset of parameters
- Multiple workers
  - Each workers needs a subset of parameters
- Key,Value for each parameter -> Too costly
- Exploit logical structure of parameters:
  - Vectors
  - Matrices

Comparison with other frameworks



Big Models and Big Data

Relationship between large models and large data

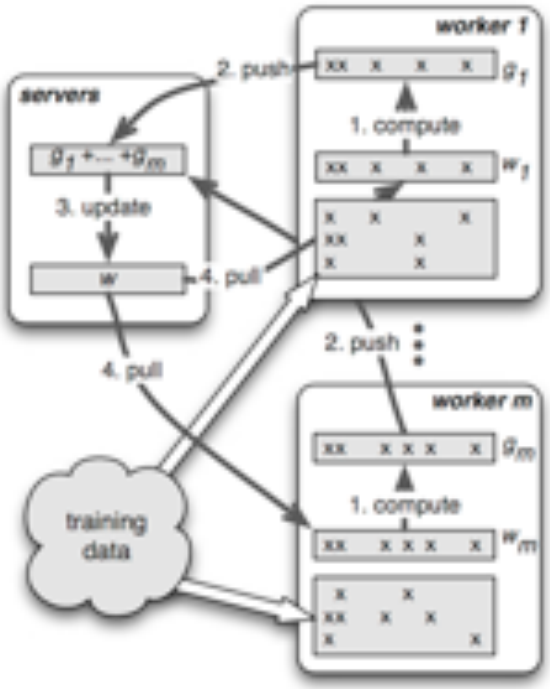
- Large models with little training data will overfit
- Small models with lots of training data will be ineffective

$$F(w) = \sum_{i=1}^{n} \ell(x_i, y_i, w) + \Omega(w).$$

loss

regularizer

Distributed Subgradient Descent



Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

1: issue LoadData() to all workers
2: **for** iteration $t = 0, \ldots, T$ **do**
3: issue WorkerIterate(t) to all workers.
4: **end for**

Worker $r = 1, \ldots, m$:

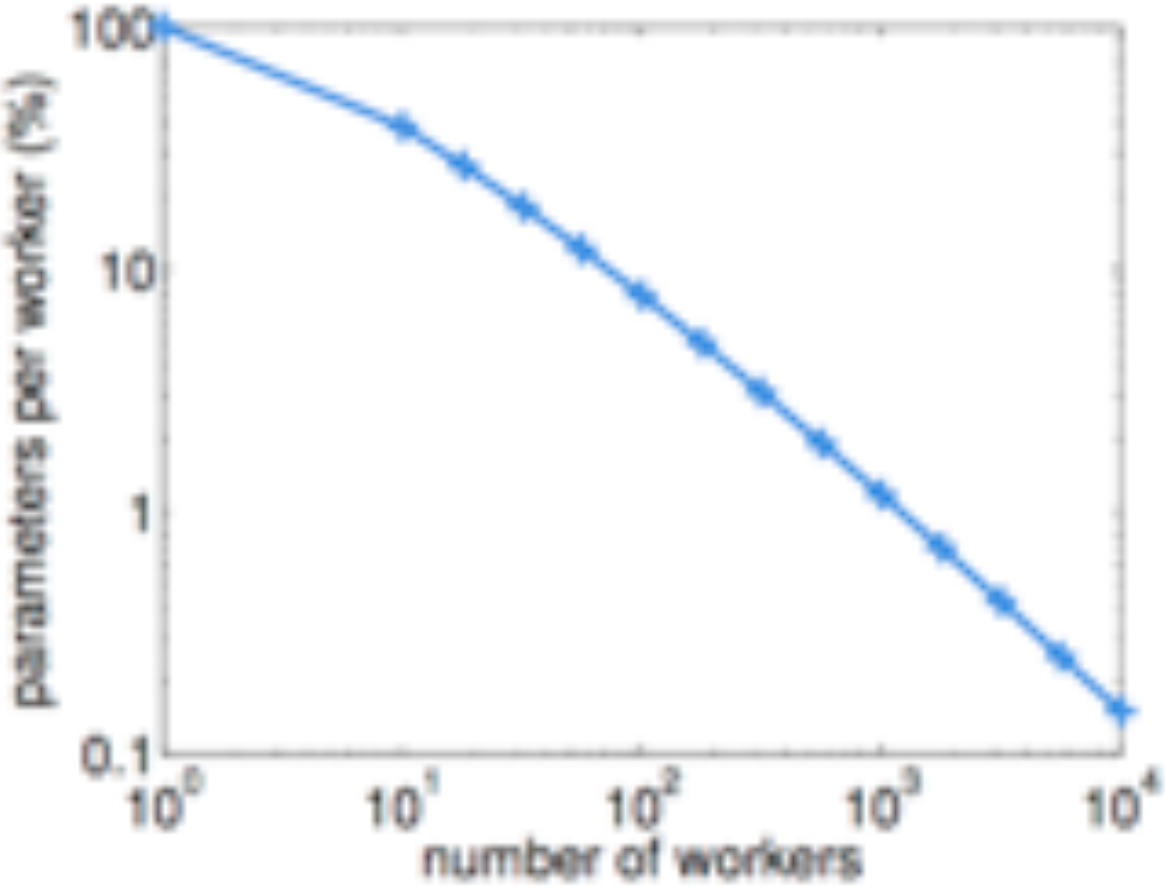
1: **function** LoadData()
2: load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
3: pull the working set $w_r^{(0)}$ from servers
4: **end function**
5: **function** WorkerIterate(t)
6: gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
7: push $g_r^{(t)}$ to servers
8: pull $w_r^{(t+1)}$ from servers
9: **end function**

Servers:

1: **function** ServerIterate(t)
2: aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
3: $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)}) \right)$
4: **end function**

Most expensive step
Workers only need w_i that are referenced in x_i

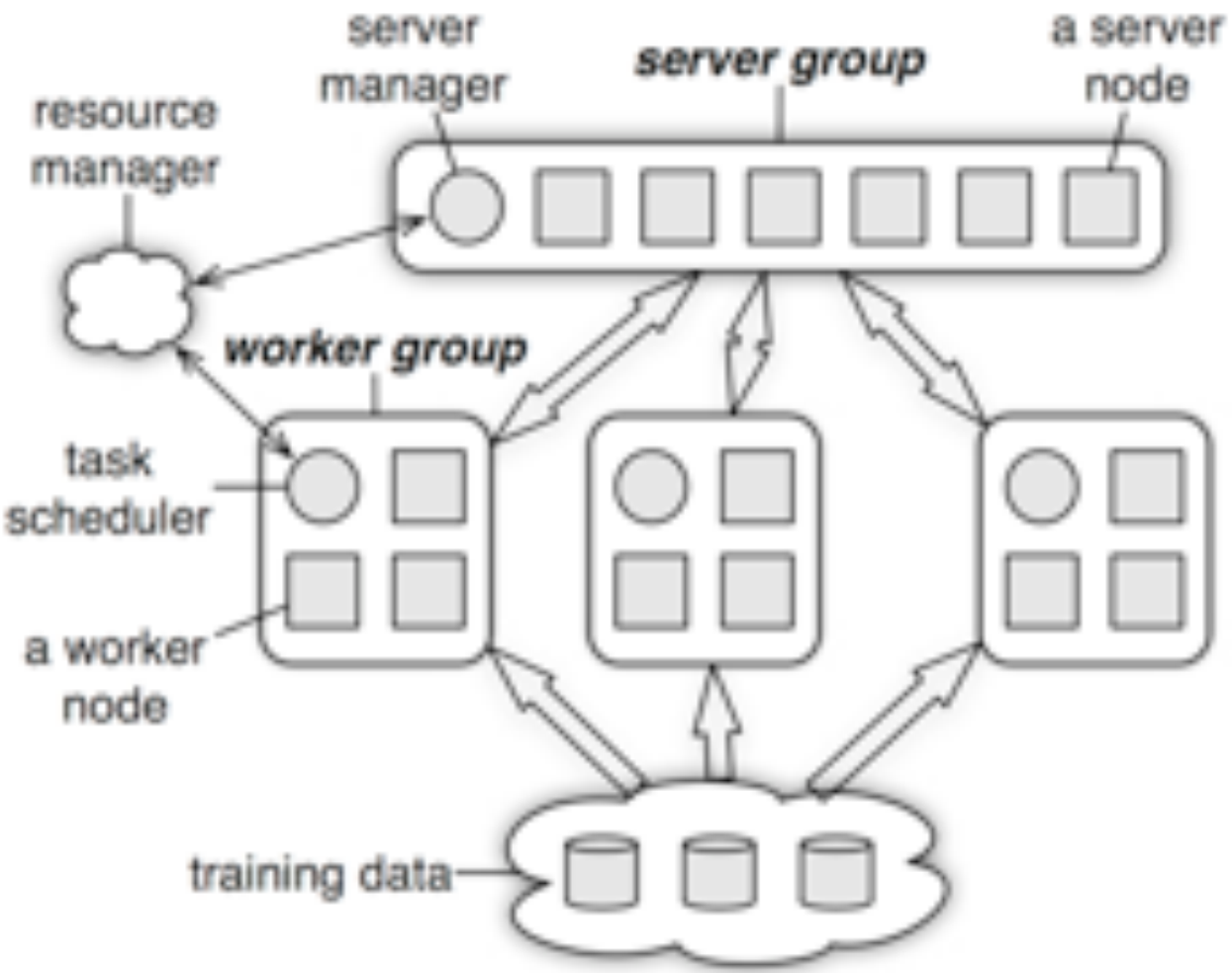
Workers vs Parameters



Unsupervised Example

- Generate topics from a list of documents
  - Without external topic list
- Latent Dirichlet Allocation (LDA)
  - Key step
    - Not a gradient
    - Measure of how well a document can be explained by model

Architecture



High-Level Operation

- Each worker group runs one application
- Server offers parameter namespaces
- Same model (namespace) may have multiple applications
  - Multiple applications solving the same image recognition
  - One application updating model and another using it

What is a Shared Parameter

- (Key, value) like in a KVS
- But server supports optimization by seeing parameters as sparse vectors
  - Vector addition
  - Multiplication Xw
  - Norm
  - others

Supporting Optimizations

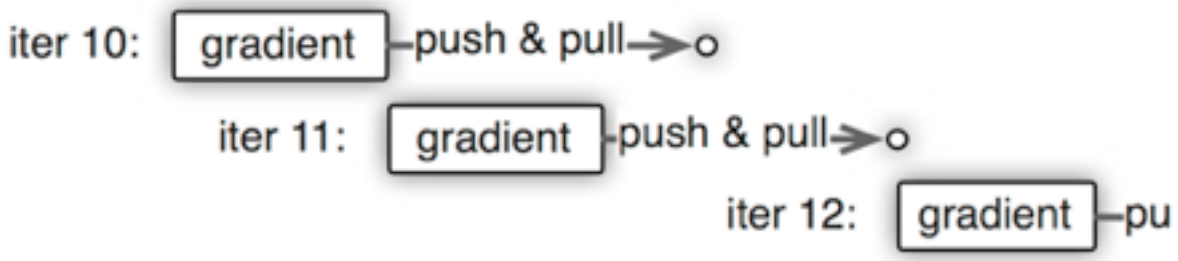
- Assume keys are ordered
  - Non-existing keys associated with 0 values
  - Exploit linear algebra libraries
- Data exchange
  - Push: worker pushes gradient
  - Pull: worker pulls weight vector
  - Optimization: range-based push and pull
    - W.push (R,dest)
    - W.pull (R,dest)

User defined functions

- Server nodes can run user-defined functions
  - E.g. to update parameters or regularizer
  - Information at server is more up-to-date than at workers
  - Only use workers for computationally intensive repetitive operations

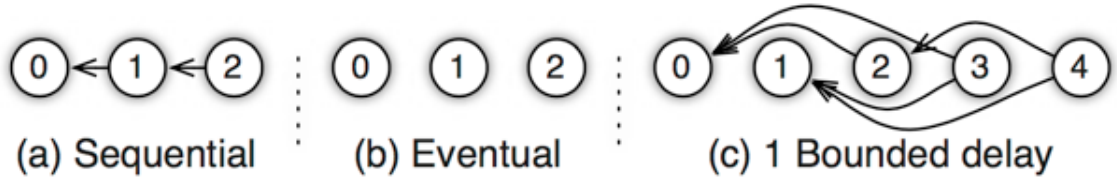
Asynchronous Tasks

- Remote procedure call issues a task
  - But RPC does not wait for task completion
- Callee replies with another RPC
- Tasks in parallel by default, but can be serialized



Consistency

- Independent tasks may lead to inconsistencies
  - Some algorithms may not care
  - Others may converge more slowly
- Programmer can specify consistency model



- Programmer may specify specific consistency criteria for particular (k,v) pairs.
  - E.g. only push entries that have changed a lot since last update

Implementation details

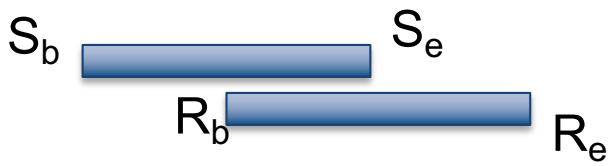
- Consistent Hashing
- Chain Replication
- Optimize range-based communication
  - Compression
    - Data
    - Range-based vector clocks

Range-based Vector Clocks

- Naïve vector clock requires O(nm) space for n noded and m parameters
- When a node pushes all parameters in a range they'll have the same timestamp
- Initially only one vector clock per node
  - Split when sending out an update across a range
  - Complexity now O(mk), k being the number of ranges

Algorithm 2 Set vector clock to t for range $\mathcal{R}$ and node i

1: **for** $\mathcal{S} \in \{\mathcal{S}_i : \mathcal{S}_i \cap \mathcal{R} \neq \emptyset, i = 1, \ldots, n\}$ **do**
2: **if** $\mathcal{S} \subseteq \mathcal{R}$ **then** $\text{vc}_i(\mathcal{S}) \leftarrow t$ **else**
3: $a \leftarrow \max(\mathcal{S}^b, \mathcal{R}^b)$ and $b \leftarrow \min(\mathcal{S}^e, \mathcal{R}^e)$
4: split range $\mathcal{S}$ into $[\mathcal{S}^b, a), [a, b), [b, \mathcal{S}^e)$
5: $\text{vc}_i([a, b)) \leftarrow t$
6: **end if**
7: **end for**

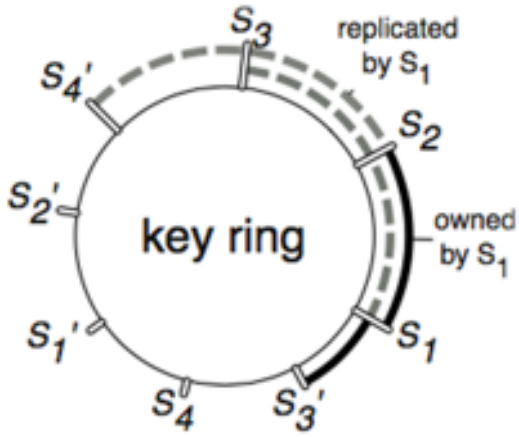


Communication between nodes

- Nodes exchange messages
  - Point-to-point
  - Multicast -> to a group of nodes
- Message
  - Set of (K,V) pairs in a Range with vector clock
  - If missing key in a range,
    - timestamp and unchanged value
  - Compression and hashing for unchanged values

Consistent Hashing

- DHT-like
- With multiple virtual servers for load balancing (see dynamo)
- 1-hop DHT
  - Servers handle ring management
  - All nodes cache key assignments
- Each server stores replica of k neighbors
- Replicate after aggregation (e.g. summing gradients)



Server Addition and Removal

- Server manager assigns new node a key range, splitting another or using that of a dead node
- Node fetches range as well as data it should replicate from neighbors
- Server managers broadcasts new assignments
  - Recipients may update their own ranges

Worker Addition and Removal

- New worker
  - Task manager assigns it a range of data
  - Worker loads training data from file system or workers
  - Worker pulls parameters from server
  - Task scheduler broadcasts change
- Worker removal
  - Algorithm designer may choose how to operate
    - Ignore loss of training data
    - Replace with another worker
    - Reassign data

Sparse Logistic Regression

Algorithm 3 Delayed Block Proximal Gradient [31]

Scheduler:

1: Partition features into b ranges $\mathcal{R}_1, \ldots, \mathcal{R}_b$
2: **for** $t = 0$ **to** T **do**
3: Pick random range $\mathcal{R}_{i_t}$ and issue task to workers
4: **end for**

Worker r at iteration t

1: Wait until all iterations before $t - \tau$ are finished
2: Compute first-order gradient $g_r^{(t)}$ and diagonal second-order gradient $u_r^{(t)}$ on range $\mathcal{R}_{i_t}$
3: Push $g_r^{(t)}$ and $u_r^{(t)}$ to servers with the KKT filter
4: Pull $w_r^{(t+1)}$ from servers

Servers at iteration t

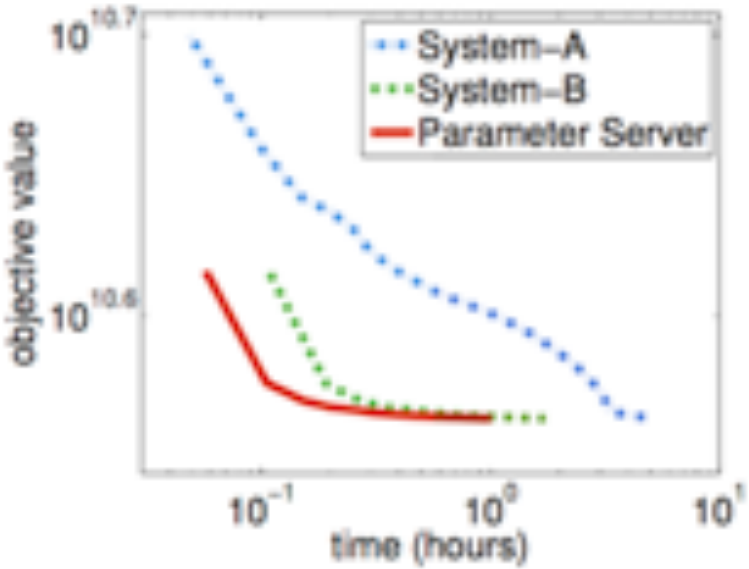
1: Aggregate gradients to obtain $g^{(t)}$ and $u^{(t)}$
2: Solve the proximal operator

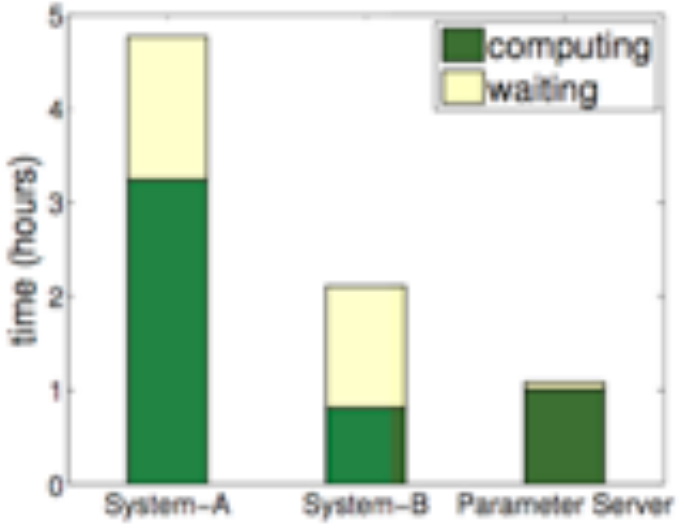
$$w^{(t+1)} \leftarrow \operatorname*{argmin}_w \Omega(w) + \frac{1}{2\eta}\|w^{(t)} - \eta g^{(t)} + w\|_H^2,$$

where $H = \operatorname{diag}(\lambda^{(t)})$ and $\|x\|_H^2 = x^T H x$

Comparison

Comparison with two state of the art systems





More results in the paper

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation* (OSDI'14). USENIX Association, Berkeley, CA, USA, 583-598.

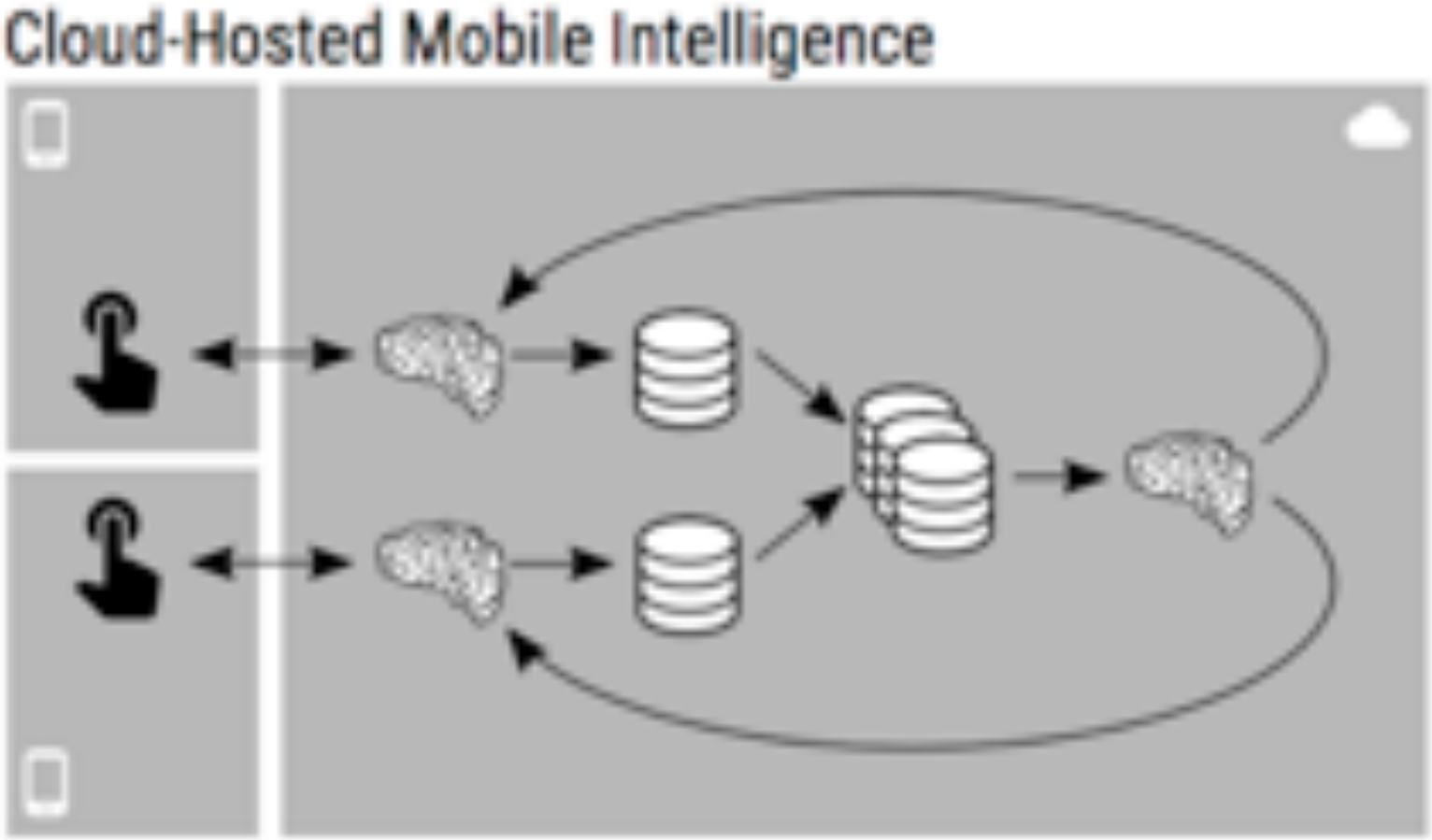
Outline

- Cloud Architectures for Distributed Data Analytics
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

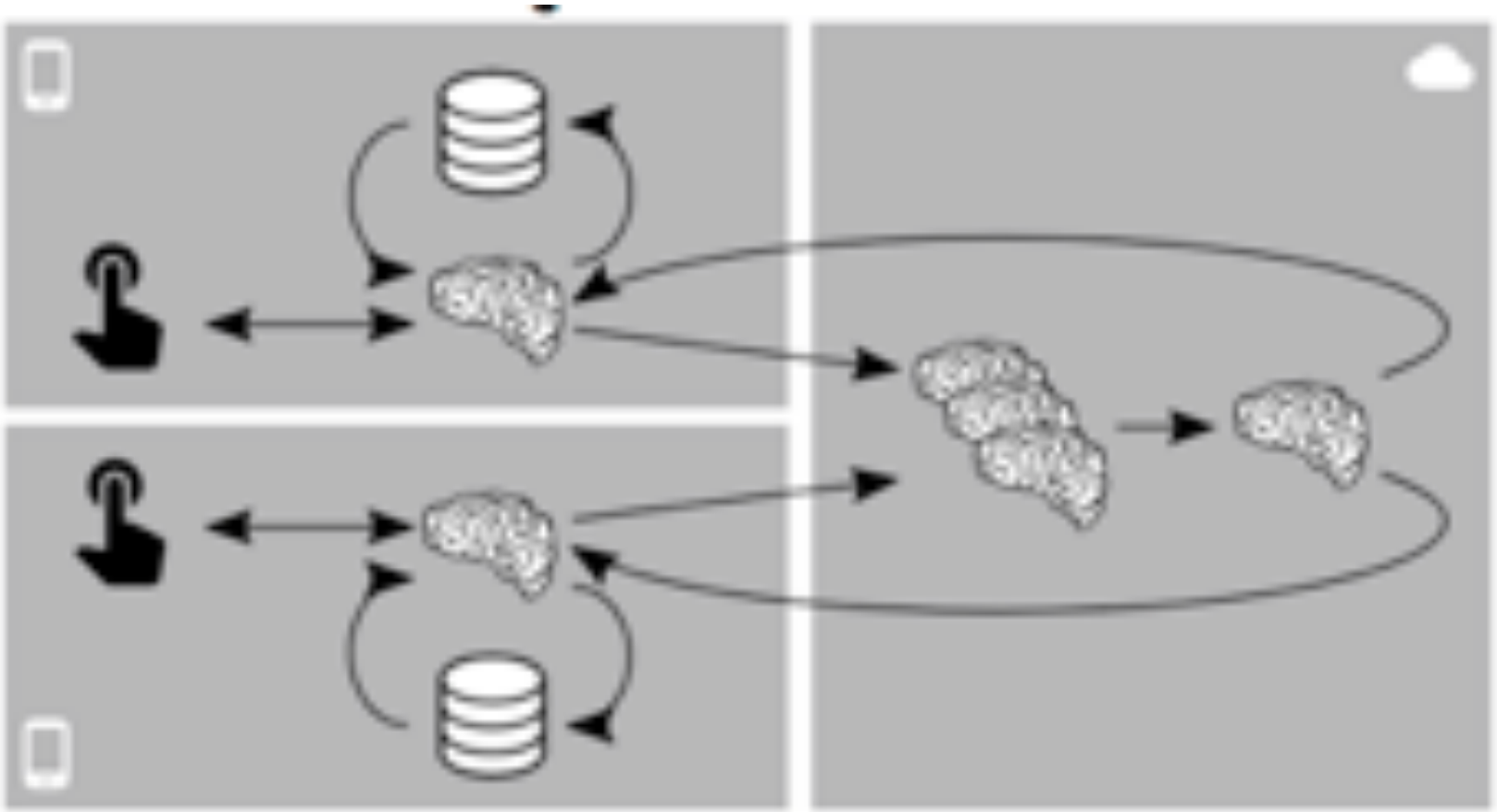
Federated Learning

- Name coined by Google
  - https://research.googleblog.com/2017/04/federated-learning-collaborative.html
- Goes beyond “model on device” solutions
- Brings training to the device
- Motivation
  - Privacy
  - Better efficiency
  - Faster response times (in model updates)

Cloud-based solution



Federated Learning



Federated Learning Motivation

- Training on real world data
- Real-world data often privacy sensitive
- Data often comes from user interaction
- Example
  - Predicting words on Android Gboard
  - Currently being tested
    - https://blog.google/products/search/gboard-now-on-android/

Challenges

- Non-IID data (IID=independent identically distributed)
- Unbalanced Data
- Massive Distribution
- Limited Communication
  - Upload bandwidth typically <1MB/s
  - Often clients participate only on wi-fi
- Computation comparatively cheap

Optimization Target

Finite sum objective function

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where} \qquad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

Data partitioned over K clients

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w)$$

Exploiting Edge Computation

- SGD easy to distribute
  - Many examples in the following slides
  - One gradient computation per round of communication
- But this is inefficient given high communication cost
- Do several steps of gradient computation, updating local model and then averaging models

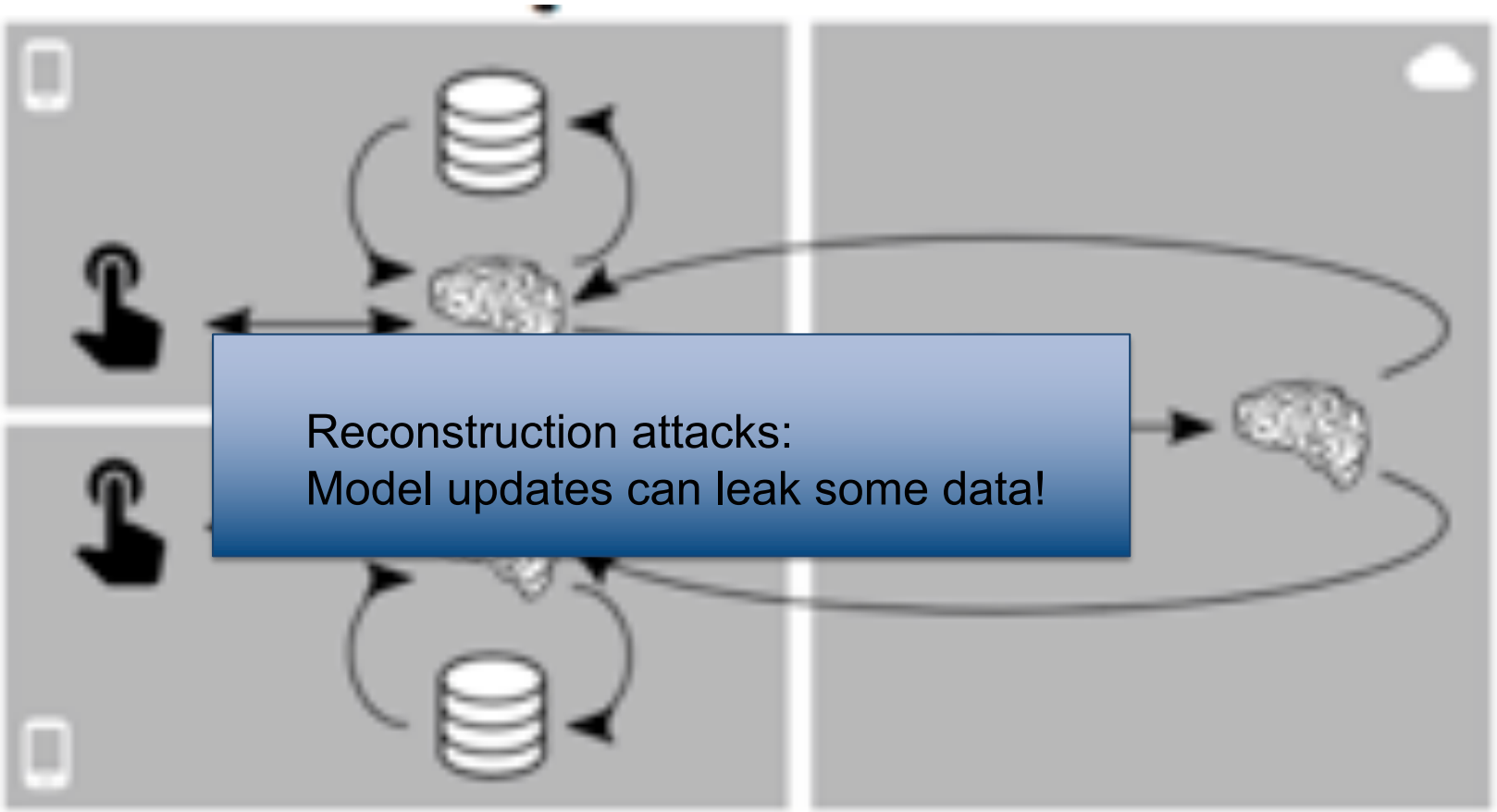
Outline

- Cloud Architectures for Distributed Data Analytics
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

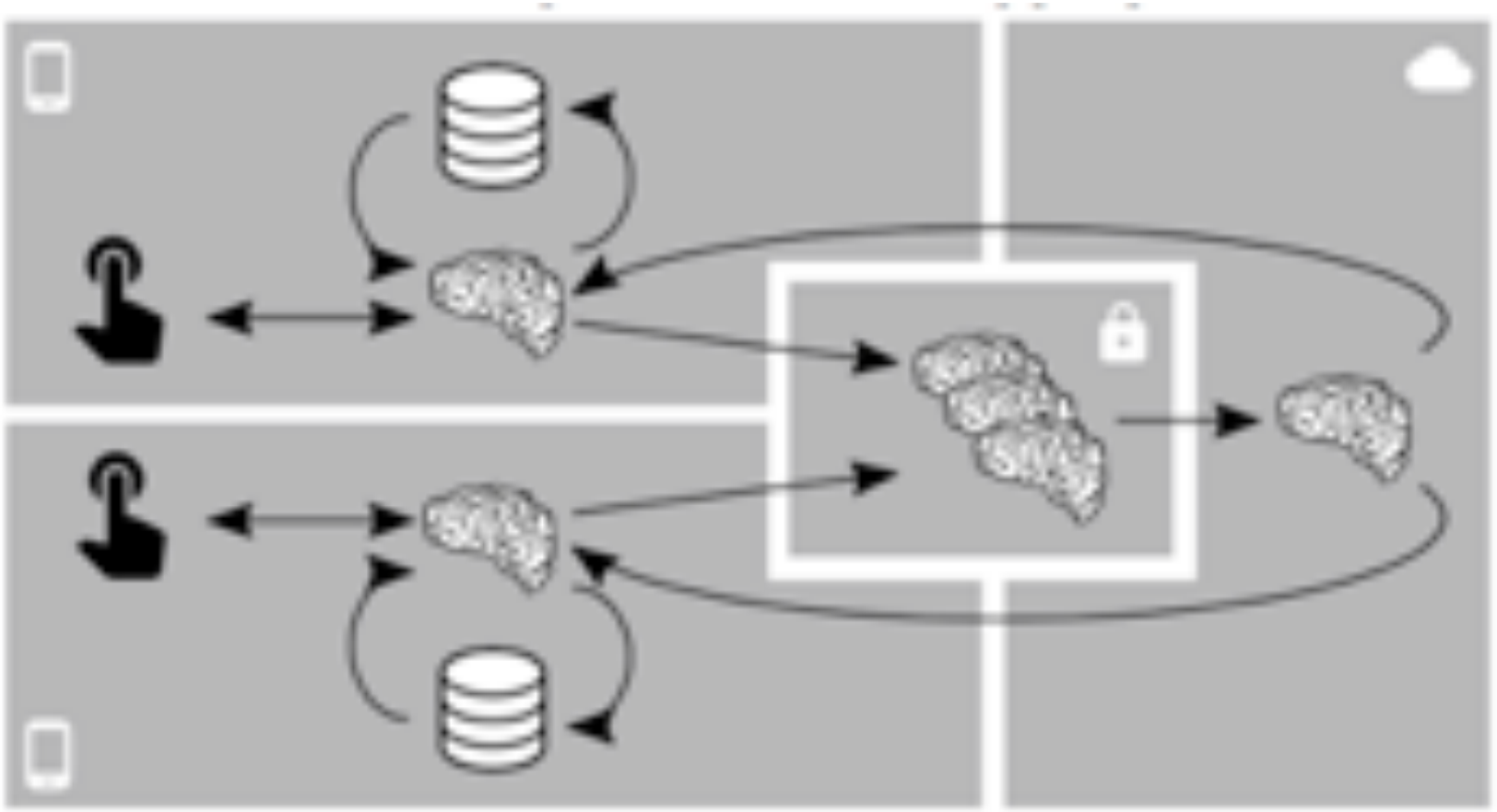
Improving Privacy in Federated Learning

- Credits
  - Slides based on
  - Bonawitz, Keith, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel,Daniel Ramage, Aaron Segal and Karn Seth. “Practical Secure Aggregation for Privacy Preserving MachineLearning.” IACR Cryptology ePrint Archive 2017 (2017): 281
  - Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel,Daniel Ramage, Aaron Segal, Karn Seth:Practical Secure Aggregation for Federated Learning on User-HeldData. CoRRabs/1611.04482 (2016)
  - Images from papers (where not otherwise specified)

Federated Learning

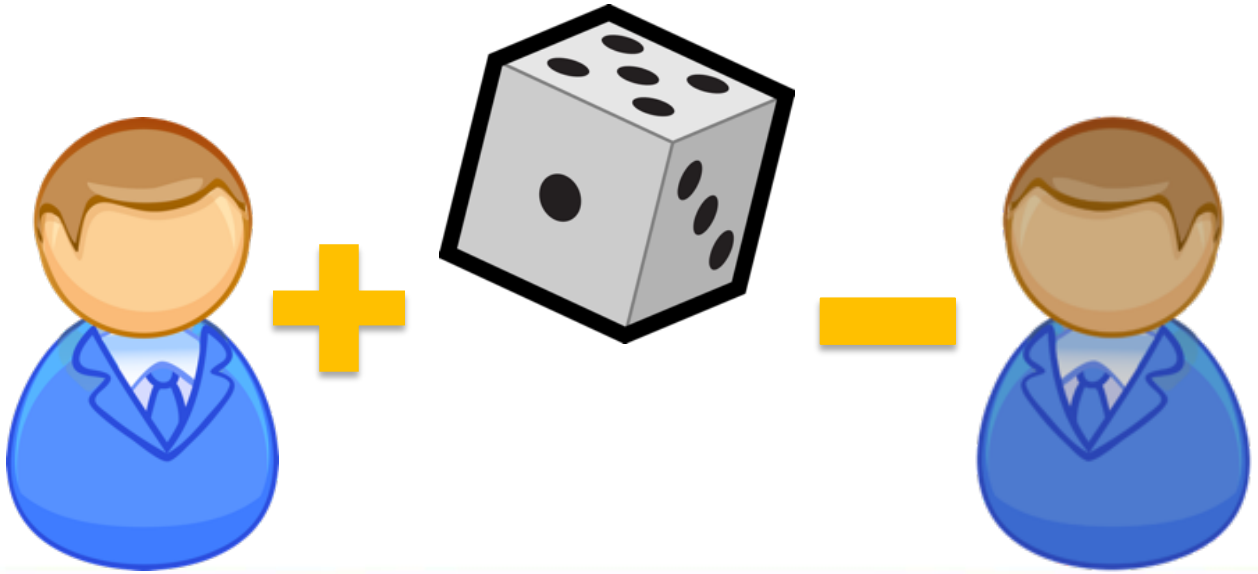


FL with Secure Aggregation



Main idea

- Pair of users agree on random value
  - One user adds it
  - Other user subtracts it



Managing dropped-out users

- Share DH secret with all users using threshold secret sharing (need at least k to recover)
  - Fine to reveal a user's secret if she has dropped out, but what if she's slow?
- Double Masking

Double Masking

- Generate additional random value in addition to shared secret.
- Send share of random value to all other users

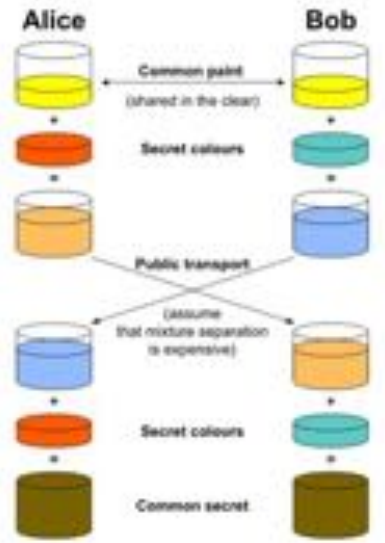
$$\boldsymbol{y}_u = \boldsymbol{x}_u + \boldsymbol{b}_u + \sum_{v \in \mathcal{U}: u<v} \boldsymbol{s}_{u,v} - \sum_{v \in \mathcal{U}: u>v} \boldsymbol{s}_{v,u} \pmod R$$

- Server may ask for share of b_u or for share of s_u,v
  - b for surviving users
  - s for dropped out users

Protocol Overview

Communication Efficiency

- Very costly to exchange all values:
  - Agree on common seeds for PRG
- Diffie Hellman scheme



Outline

- Cloud Architectures for Distributed Data Analytics
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

Lightweight Privacy-Preserving Aggregation

- Joint work with
- Tristan Allard, George Giakkoupis, Julien Lepiller

Davide Frey
davide.frey@inria.fr
Inria Rennes

Private Decentralized Averaging

- Imagine a network of things with precious, valuable, but privacy sensitive data.
- How do you aggregate such data without hampering privacy?

Some solutions exist

- Centralized Aggregator
  - Trusted Third Party
  - TEE-based
- Decentralized Aggregation with homomorphic encryption

  [Allard 2015]

  Need special hardware, trust, or heavy encryption operations

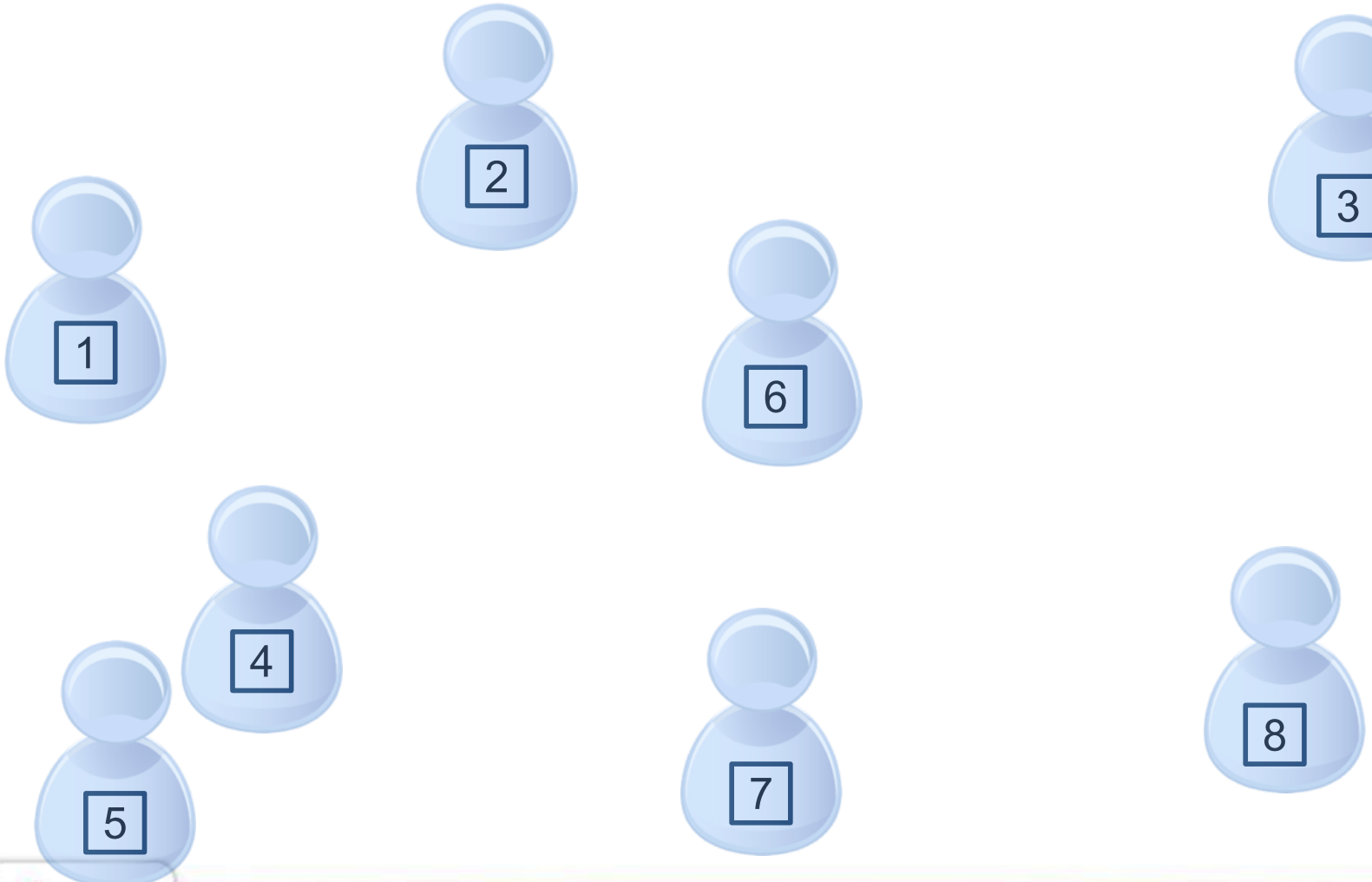
We Target Networks of Small Devices

Measurements with Damgard-Jurik cryptosystem on a Raspberrry Pi. Times in seconds.

Key size	Encryption	Addition	Decryption
1024	1	2.10^{-3}	3
2048	1.10^{1}	7.10^{-3}	2.10^{1}
4096	8.10^{1}	3.10^{-2}	2.10^{2}

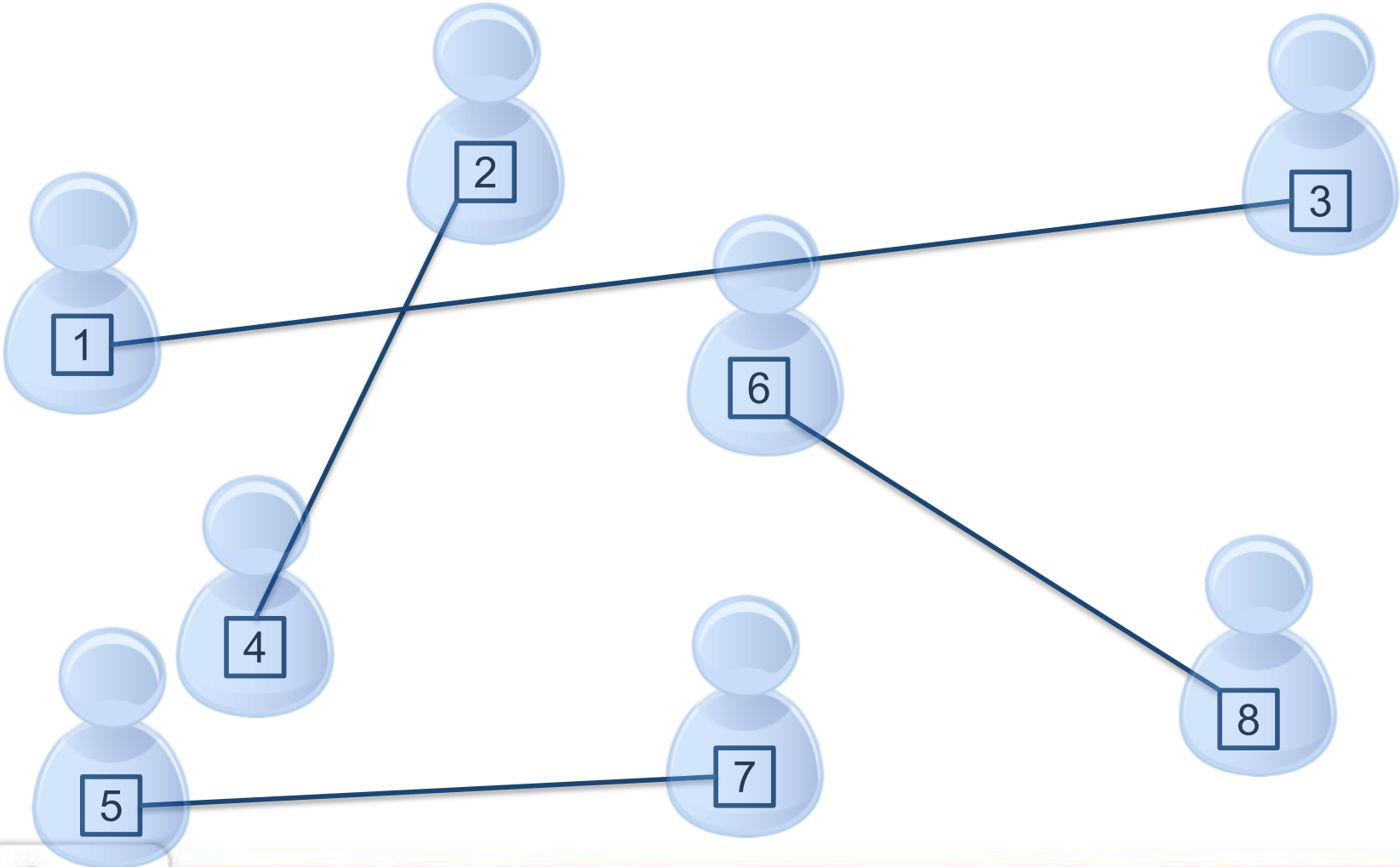
Starting Point

Gossip-based averaging [Jelasity 2004]



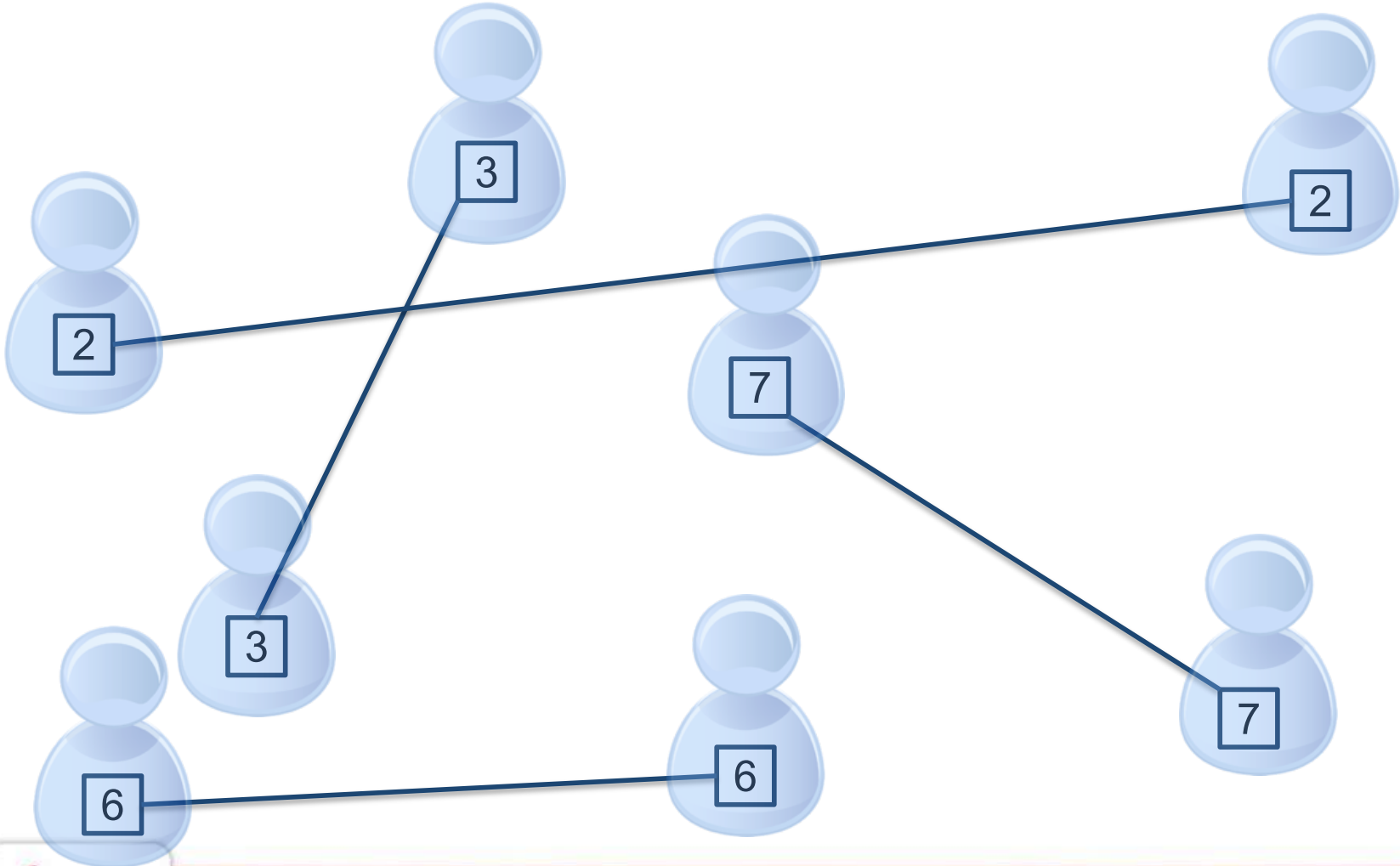
Starting Point

Gossip-based averaging [Jelasity 2004]



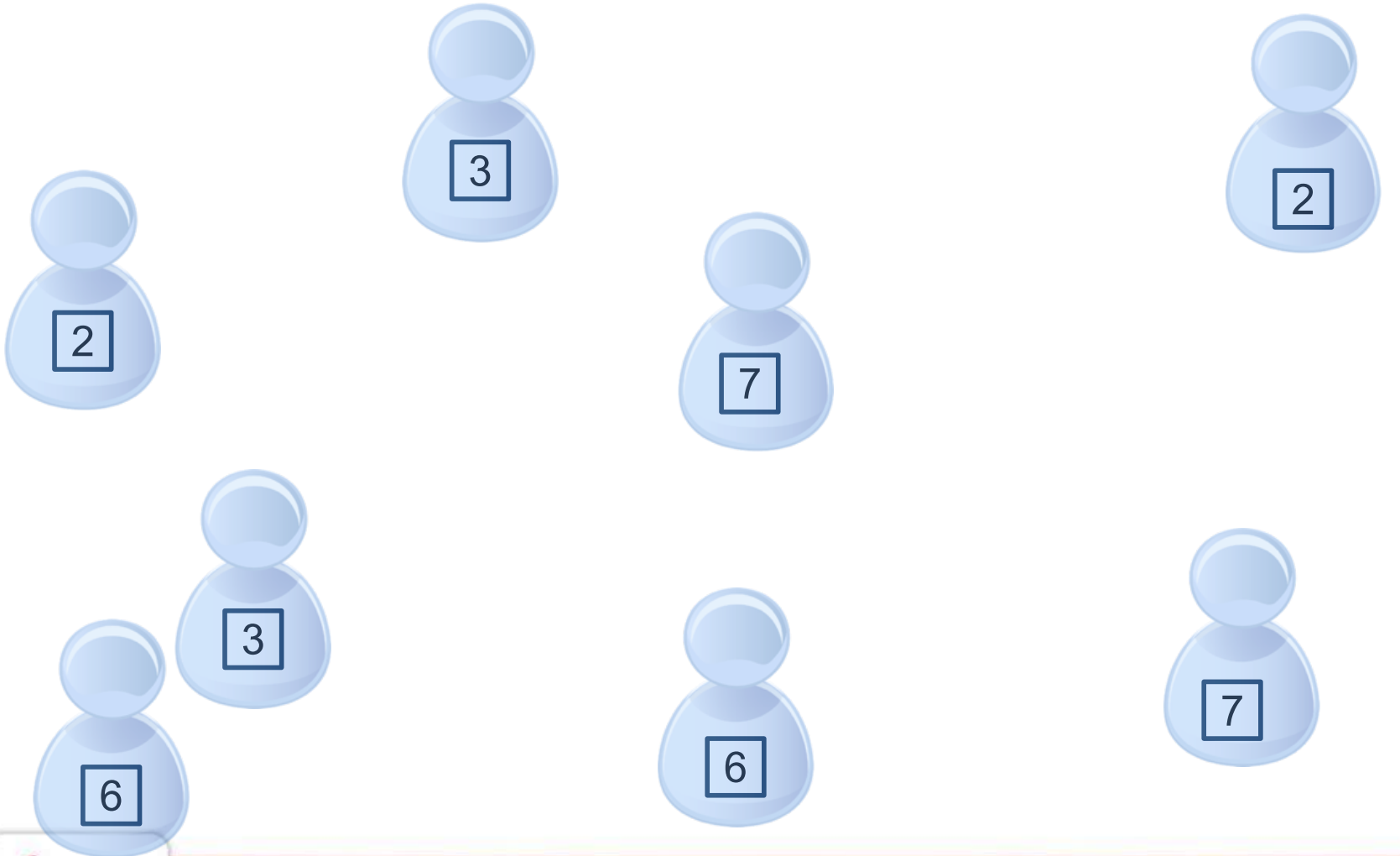
Starting Point

Gossip-based averaging [Jelasity 2004]



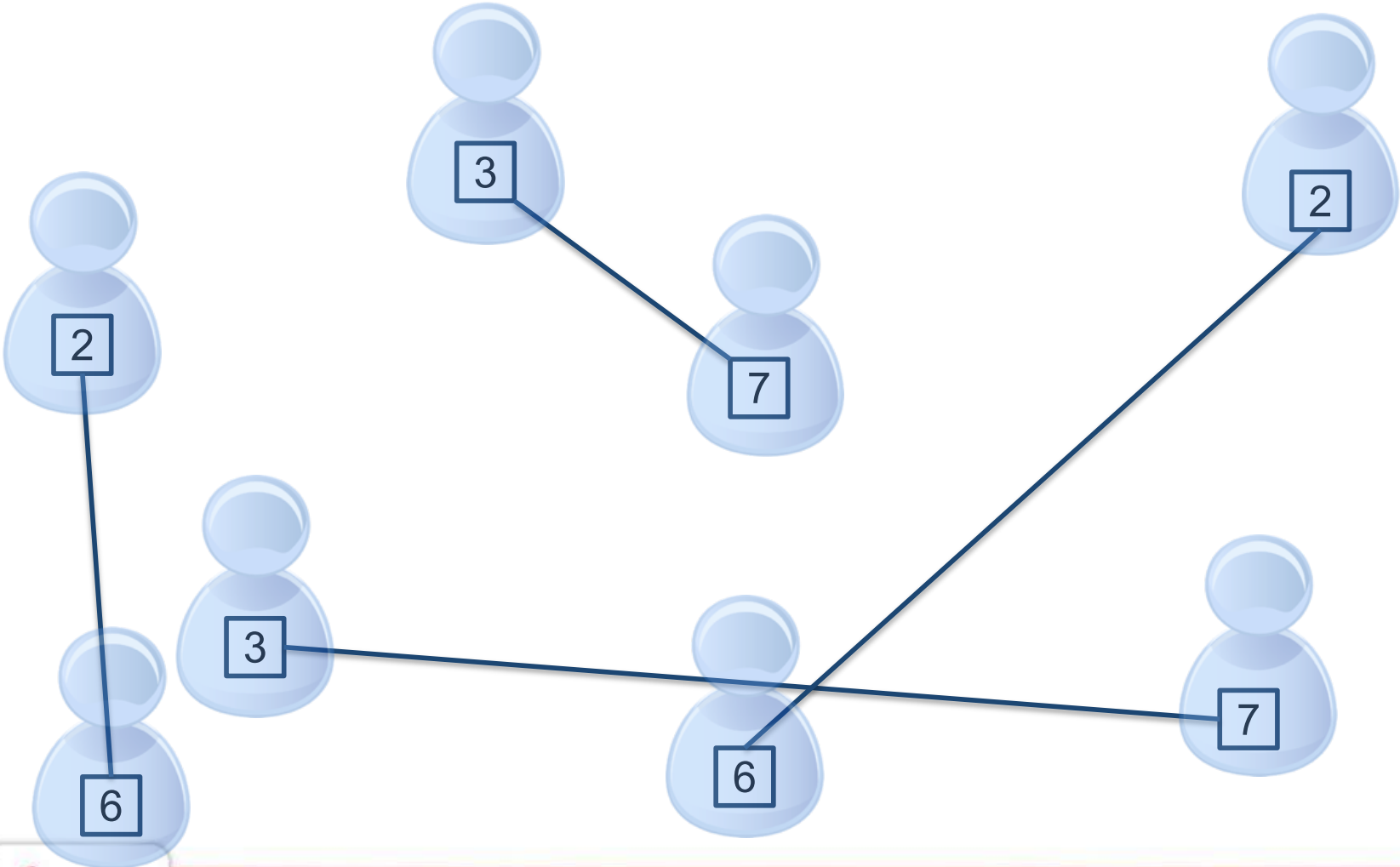
Starting Point

Gossip-based averaging [Jelasity 2004]



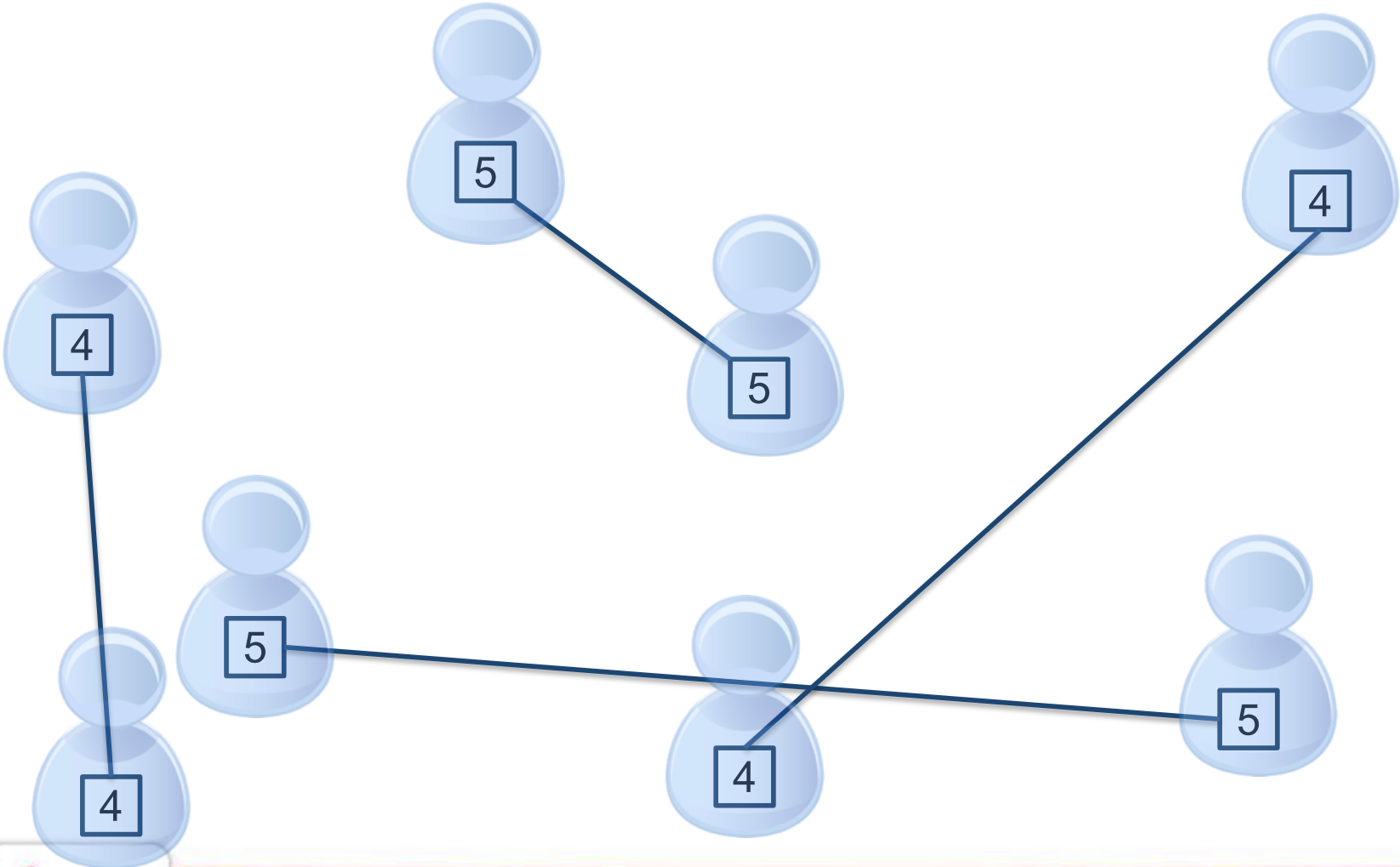
Starting Point

Gossip-based averaging [Jelasity 2004]



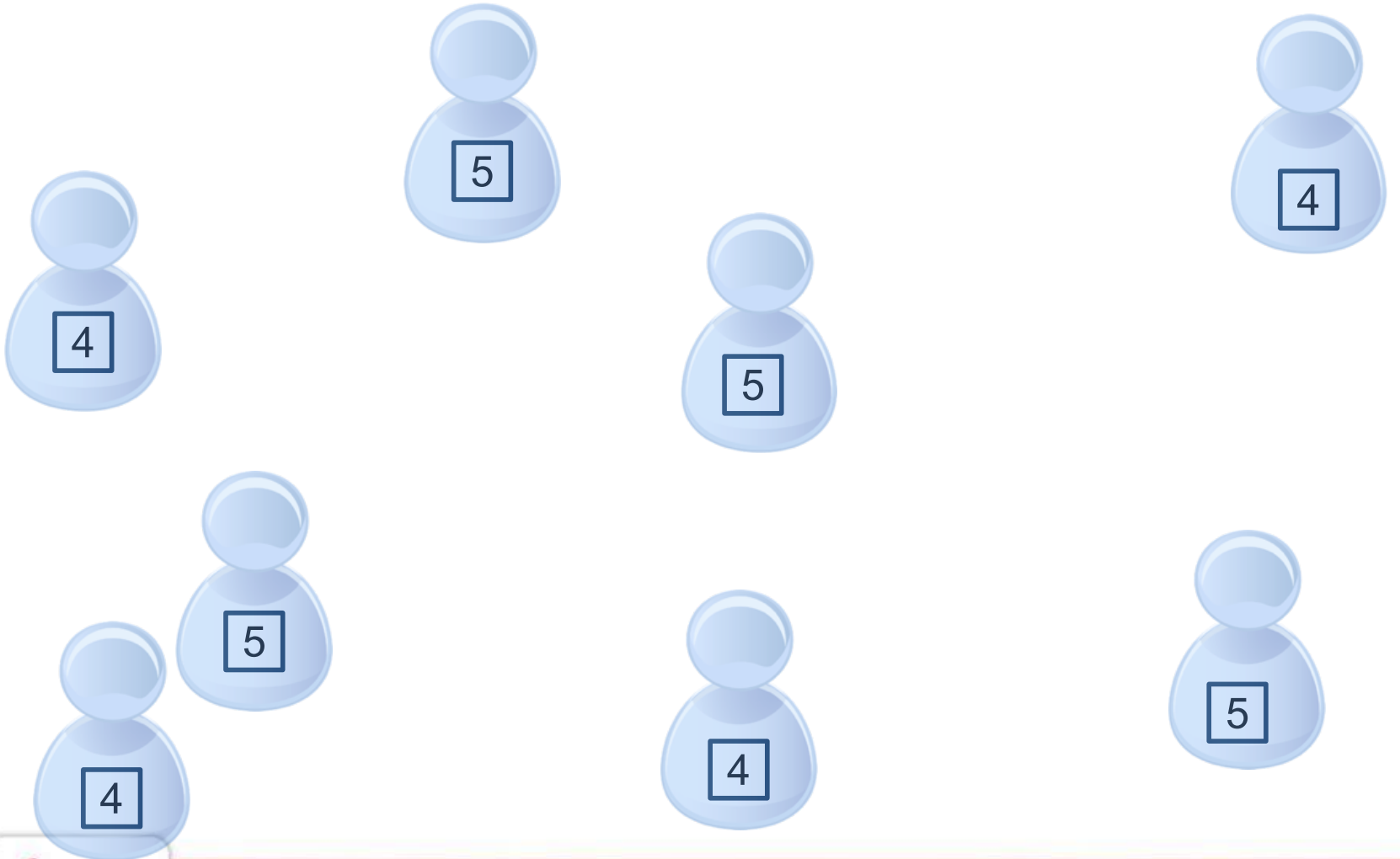
Starting Point

Gossip-based averaging [Jelasity 2004]



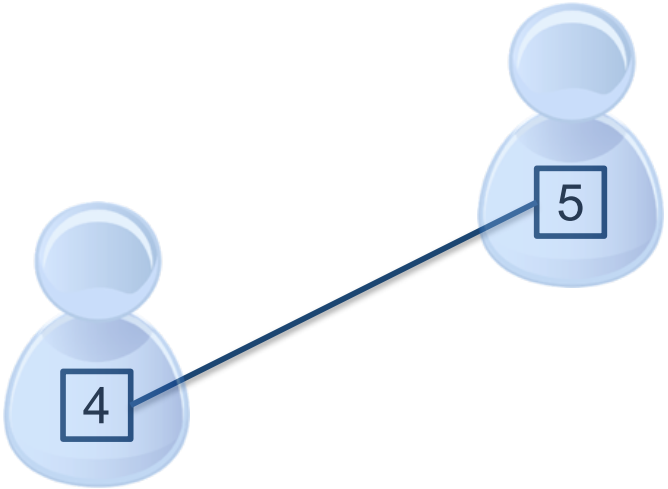
Starting Point

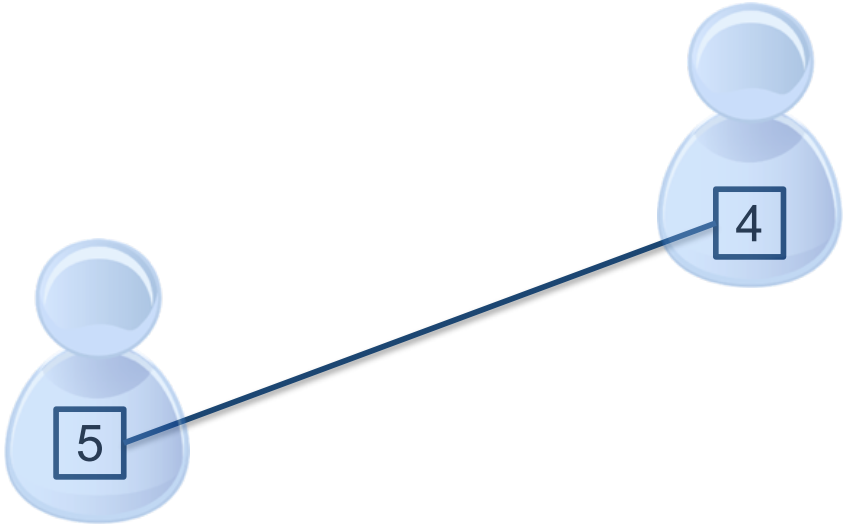
Gossip-based averaging [Jelasity 2004]

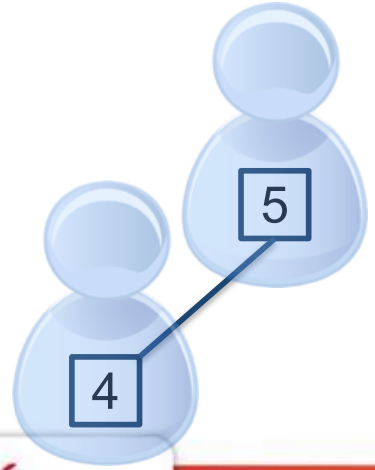


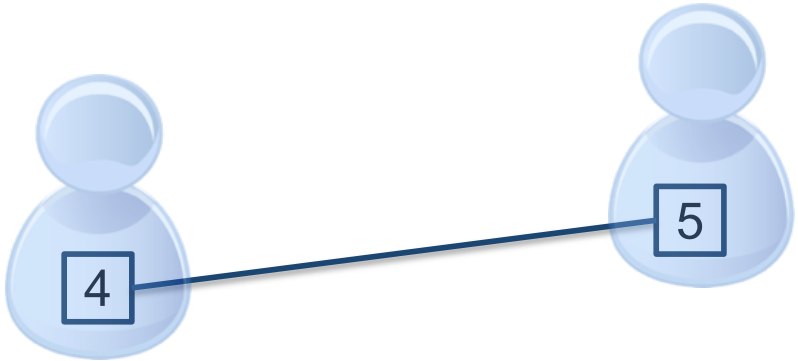
Starting Point

Gossip-based averaging [Jelasity 2004]



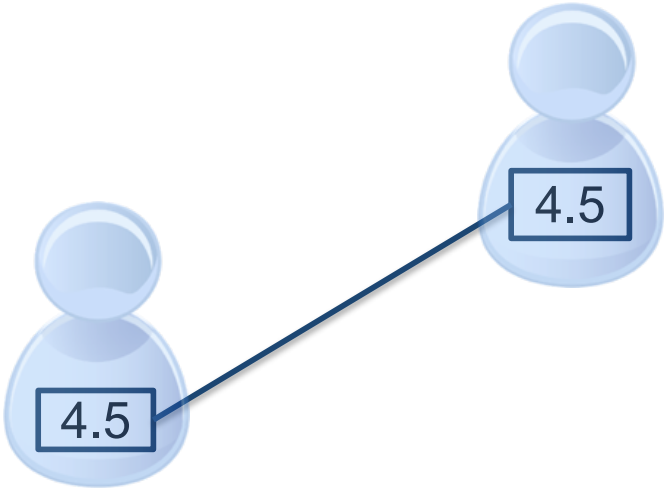


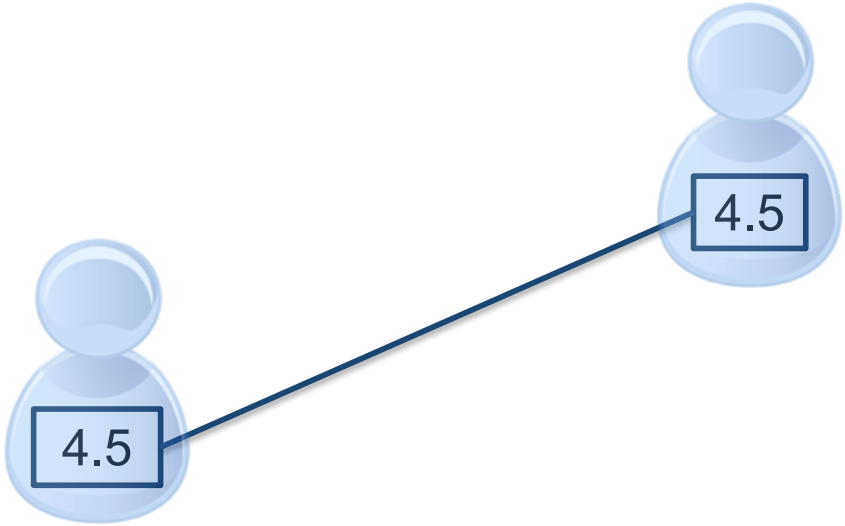


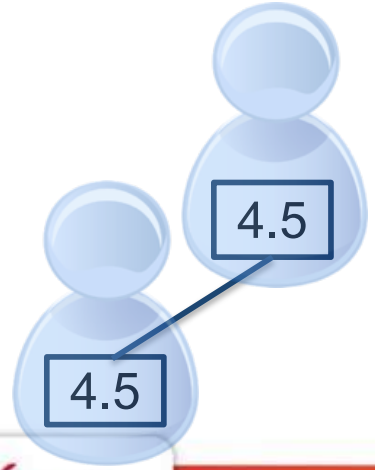


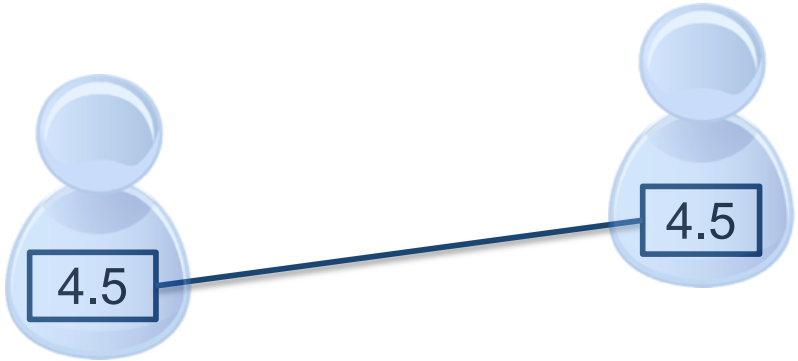
Starting Point

Gossip-based averaging [Jelasity 2004]



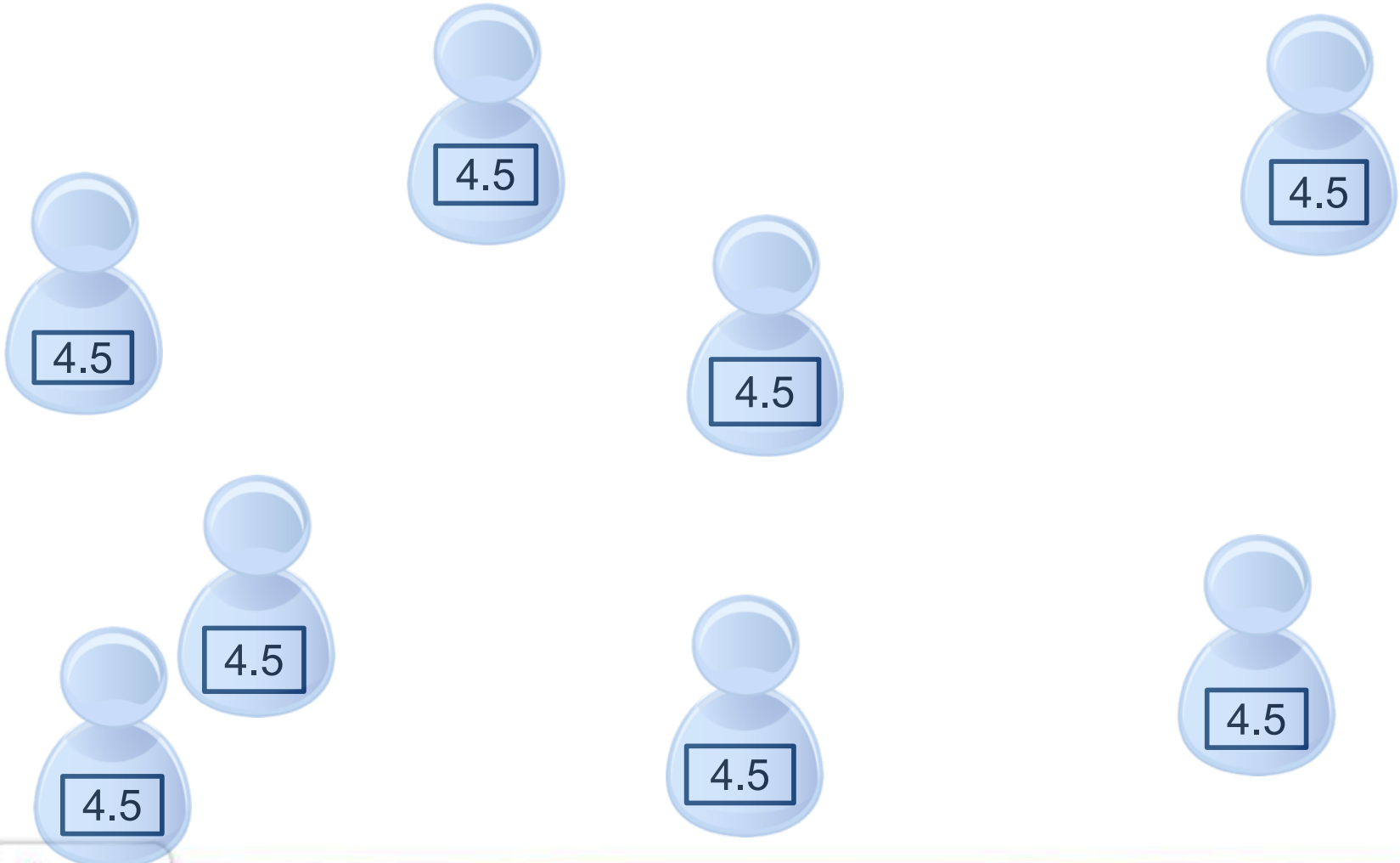






Starting Point

Gossip-based averaging [Jelasity 2004]



Attacker Model

Node Attacker

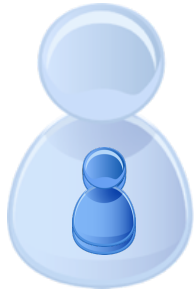
- Participates in the protocol
- Honest but curious
- Controls multiple nodes

Edge/Eavesdropping Attacker

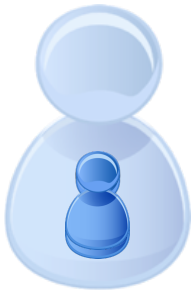
- Observes communication from the outside
- Honest but curious (won't alter messages)
- Models ISP or the NSA

Addressing Node Attackers

- Each node emulates a number of virtual nodes
- Randomized values with average equal to node's value



Inspired by Secret Sharing [Shamir79]



Creating the Shares

- Noise distribution
  - Uniform over a finite interval
  - Non uniform over unbounded interval
    - Gaussian
    - Laplace
- Noise average
  - Zero sum → exact average
  - Non-zero sum → DP

Addressing Edge Attackers

- Eavesdropping ISP can see all communication in and out of a node (or multiple nodes)
  - Shares offer no protection

Addressing Edge Attackers: Observation

Values become less and less private as protocol advances

Use encryption only in the first rounds

Preliminary Evaluation

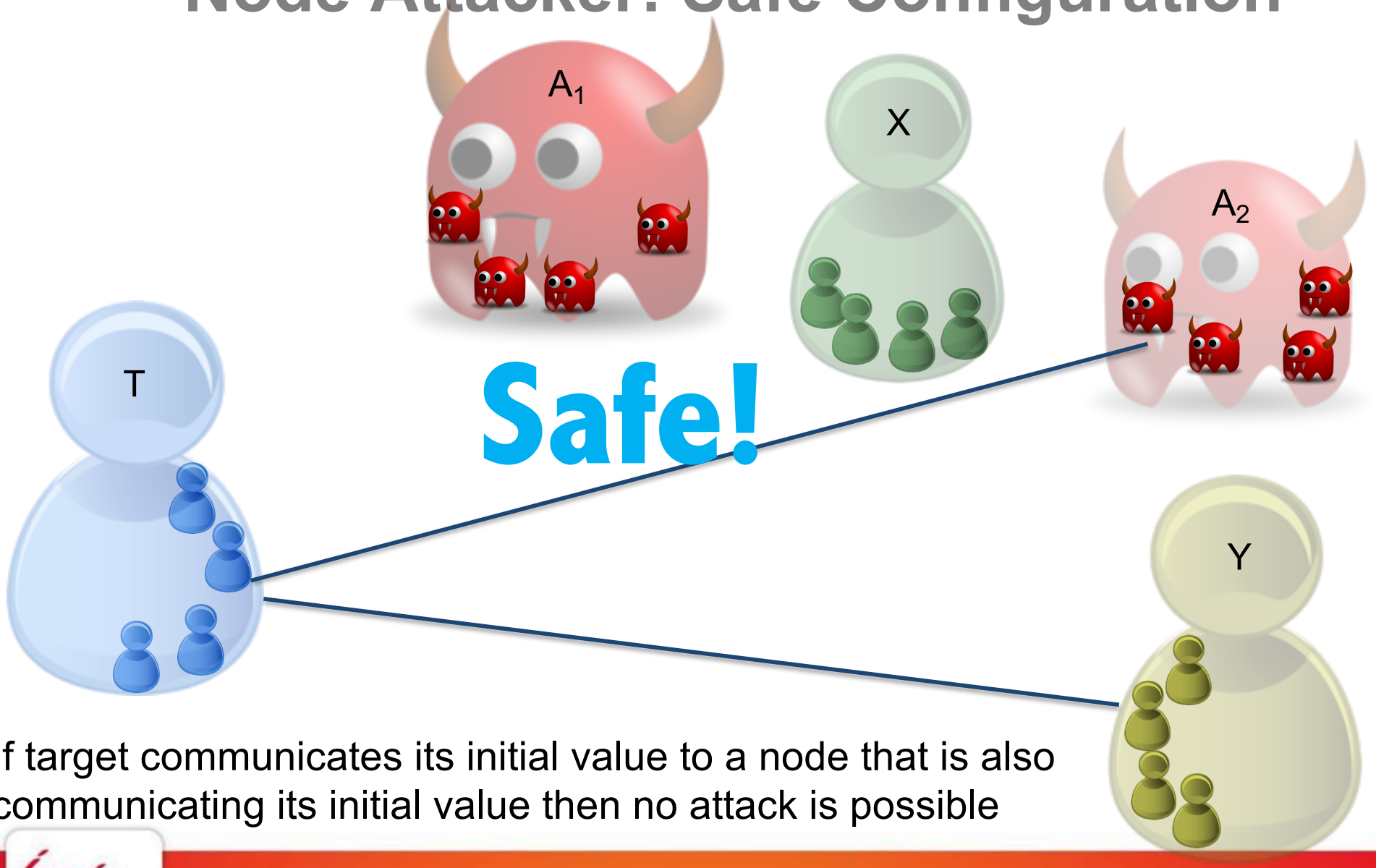
Model

- *n* network nodes
- Node Attacker
  - Controls c colluding attackers
  - Evaluate probability of learning initial value
- Edge Attacker
  - Models an eavesdropping service provider
  - Evaluate leakage at the end of encrypted rounds

Node Attacker: Strawman Attack

Probability of directly contacting a colluder at the first round: $\frac{c}{n}$

Node Attacker: Safe Configuration

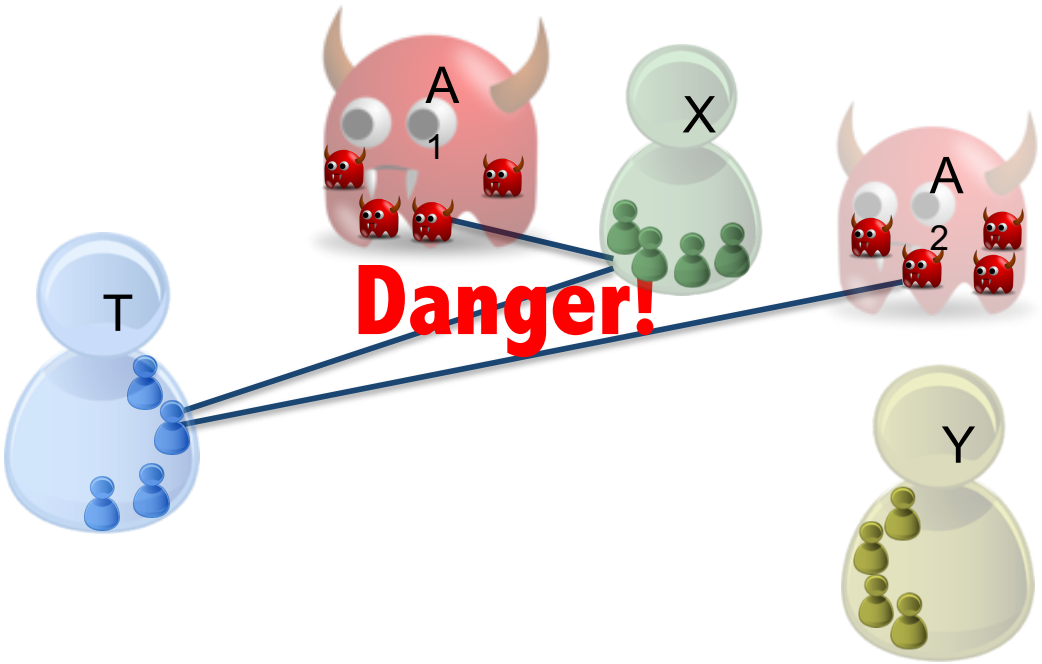


If target communicates its initial value to a node that is also communicating its initial value then no attack is possible

Node Attacker: Indirect Attack

Colluders can also learn information indirectly X talks to A1, then talks to T, then T talks to A2. A1 and A2 can reconstruct T's value.

Indirect Attack: Analysis



Colluders can also learn information indirectly X talks to A1, then talks to T, then T talks to A2. A1 and A2 can reconstruct T's value.

$$P(i,j) = \frac{i^j e^{-i}}{j!}$$

Probability that a node has already exchanged *j* values after *i* rounds

$$P_{\text{Xall}}(i) = 1 - \sum_{j=0}^{s-1} P(i,j)$$

Probability that a node has already exchanged all its initial share values

Node Attacker: a Rough Upper Bound

$\frac{c}{n}$ ..... Probability that T contacts an attacker

$P_{\text{Xall}}(i) = 1 - \sum_{j=0}^{s-1} P(i,j)$ ..... Probability of a node's having exchanged all initial values

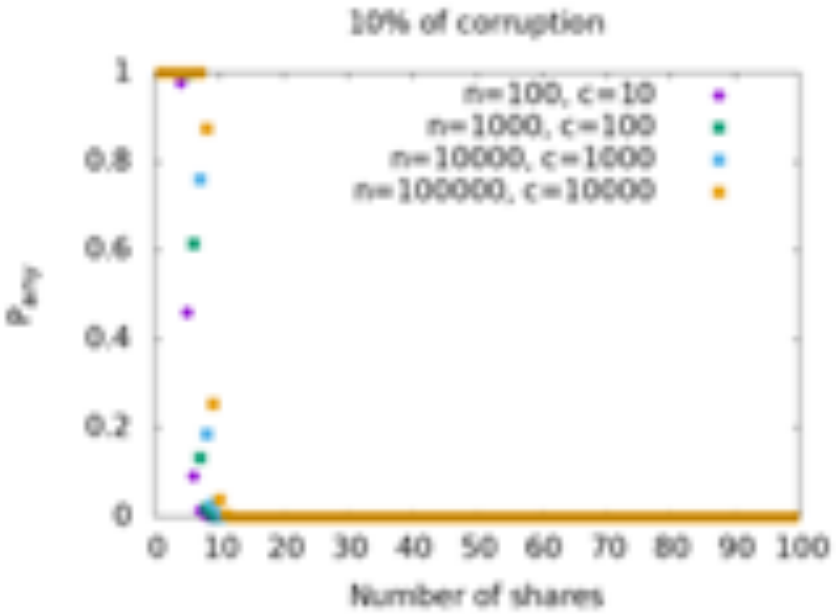
$\frac{n-c}{n} \cdot P_{\text{Xall}}(i)$ ..... Probability of contacting one such node

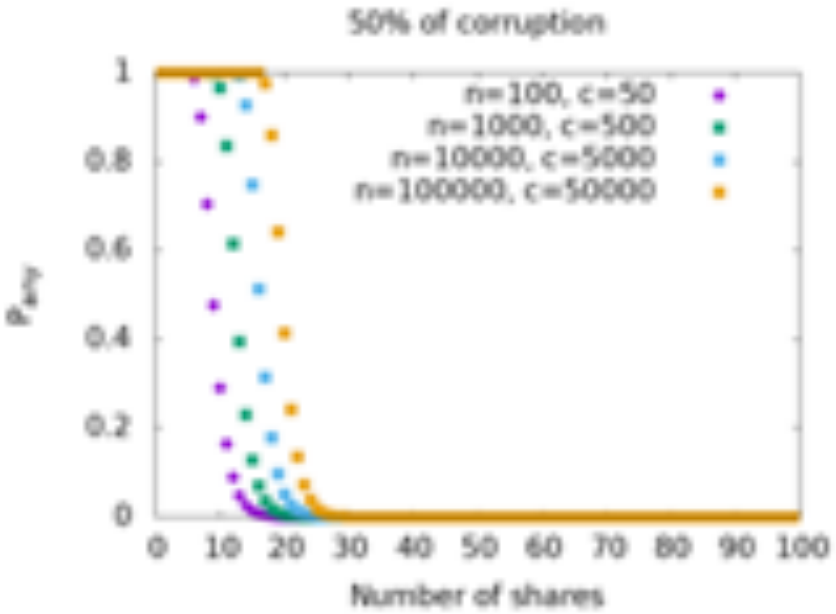
$P_{\text{leak}} = \frac{c}{n} + \frac{n-c}{n} \cdot P_{\text{Xall}}(i)$ ..... Probability of contacting one such node

$P_{\text{learn}} \leq \prod_{i=1}^{s} \left(\frac{c}{n} + \frac{n-c}{n} \cdot P_{\text{Xall}}(i) \right)$ ..... Probability T's leaking all of its shares

$P_{\text{any}} \leq 1 - (1 - P_{\text{learn}})^{n-c}$ ..... Probability that a node learns about all the shares of at least one node

Node Attacker: Results



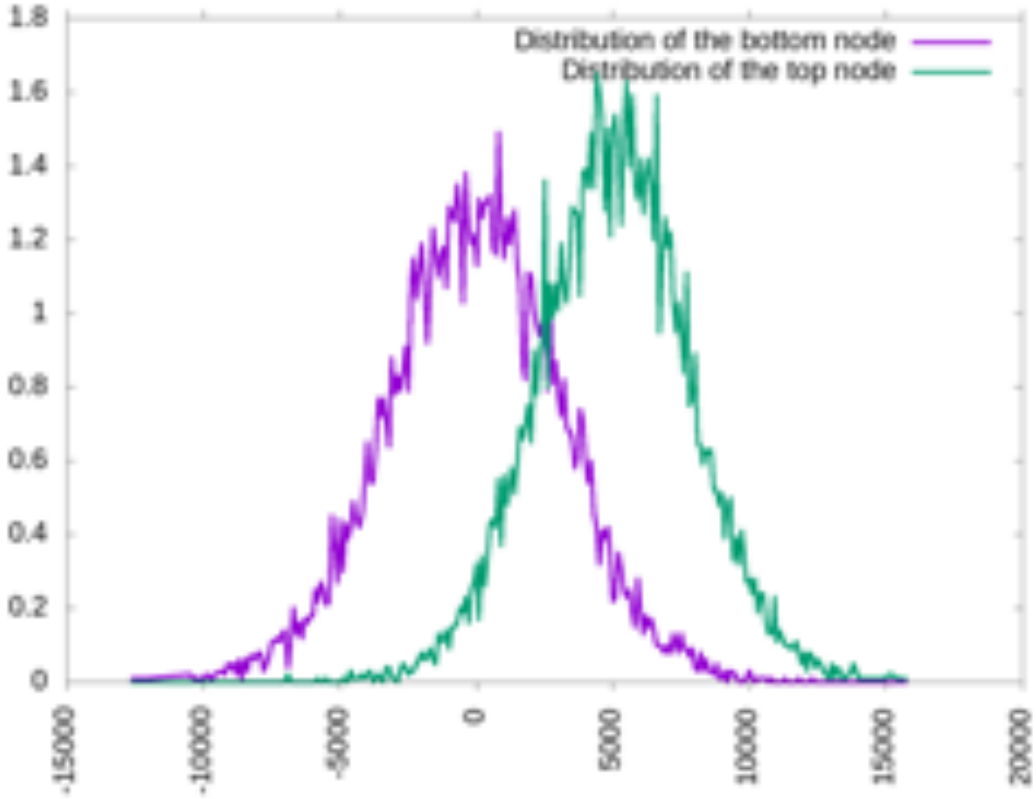


Preliminary Evaluation: Edge Attacker

- Worst-case scenario
  - 1 node with value 10k
  - 9999 nodes with value 0
- Evaluate distribution of possible values for the two types
- Two configurations
  - Less Noise & More Encryption
  - More Noise & Less Encryption

Edge Attacker Results

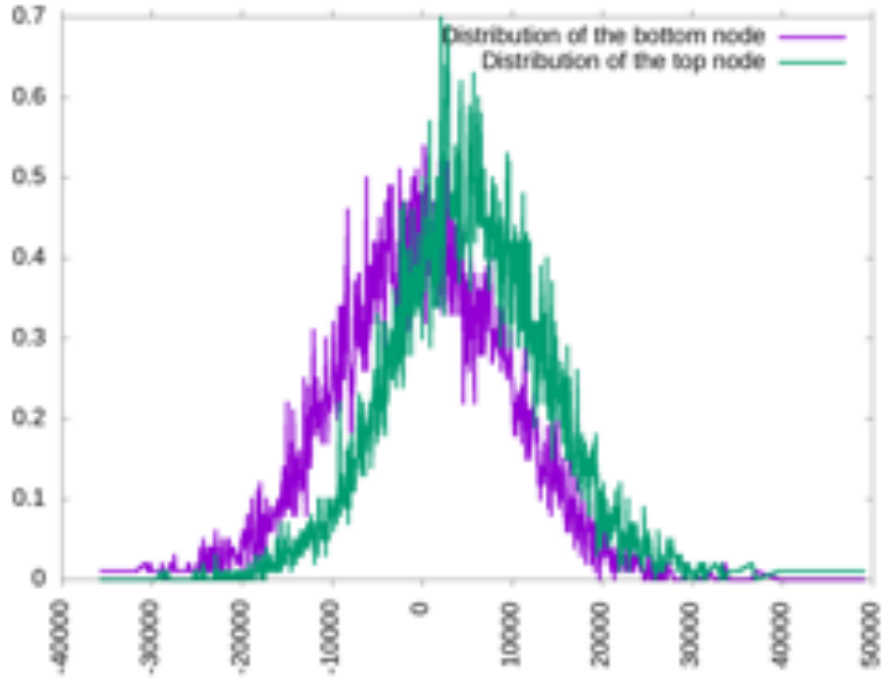
Baseline configuration



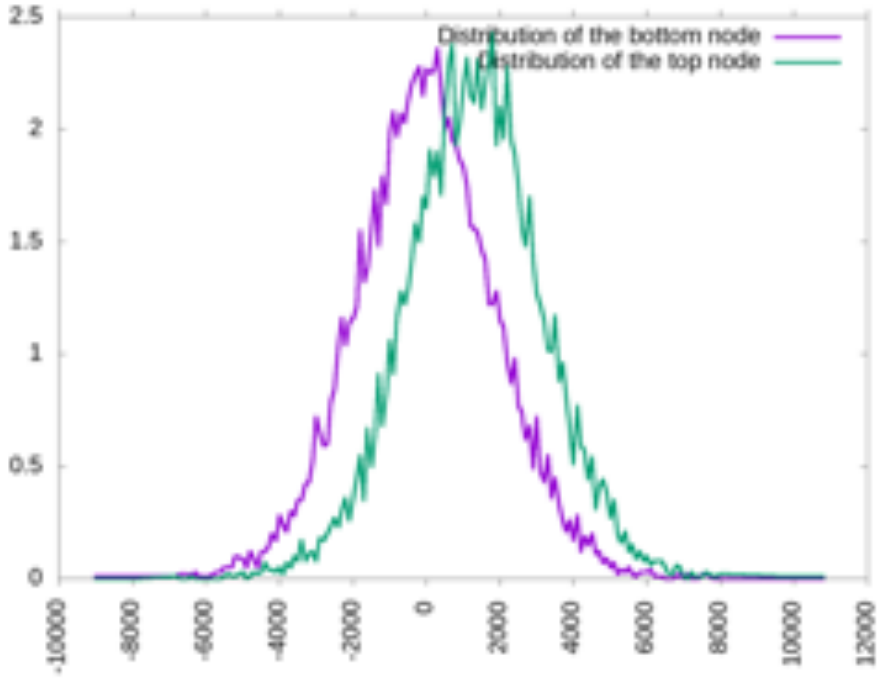
1 Encrypted round, σ = 10k

Edge Attacker Results

Different ways to achieve good trade-off



1 Encrypted round, σ = 30k



3 Encrypted round, σ = 10k

Privacy preserving Random-Walk-Based Gradient Descent

- Credits
  - Slides based on
    - Gábor Danner, Márk Jelasity: Fully Distributed Privacy Preserving Mini-batch Gradient Descent Learning. DAIS 2015: 30-44
    - Róbert Ormándi, István Hegedüs, Márk Jelasity: Gossip learning with linear models on fully distributed data. Concurrency and Computation: Practice and Experience 25(4): 556-571 (2013)
  - Images and algorithms from papers

Privacy preserving mini-batch gradient descent

- Network of nodes without server
- Each node has all information about each data item
- Each node may have as little as one training example
- Adversary wantes to learn private data of other nodes
- Corrupted nodes picked a-priori

Stochastic Gradient Descent

- Iterate over training examples in random order
- For each example
  - Compute gradient of error function (loss)
  - Update parameters
- To accelerate learning
  - Mini batches
  - Compute many gradients
  - Update as a result of batch

Gossip Learning

- Models perform random walks on the network

Algorithm 1 Gossip Learning Scheme

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send modelCache.freshest() to p
6: end loop
7: procedure ONRECEIVEMODEL(m)
8:     modelCache.add(createModel(m, lastModel))
9:     lastModel ← m
10: end procedure
```

createModel: create a updated model based on local information

Outline

- Cloud Architectures for Distributed Data Analytics
- Edge Architectures for Scalability and Privacy Preservation
- Peer-to-Peer architectures for Private Analytics

To Take Away and Conclude

- Distributed and Privacy Preserving ML and data analytics
- Very hot topics!
- Lots of papers being published
- Lots of opportunities