elicsir
Elevating Italian Computer Science Innovation and Research

14.05.2026

Gossip Protocols on Distributed Systems

Order without authority

Savorgnan Enrico

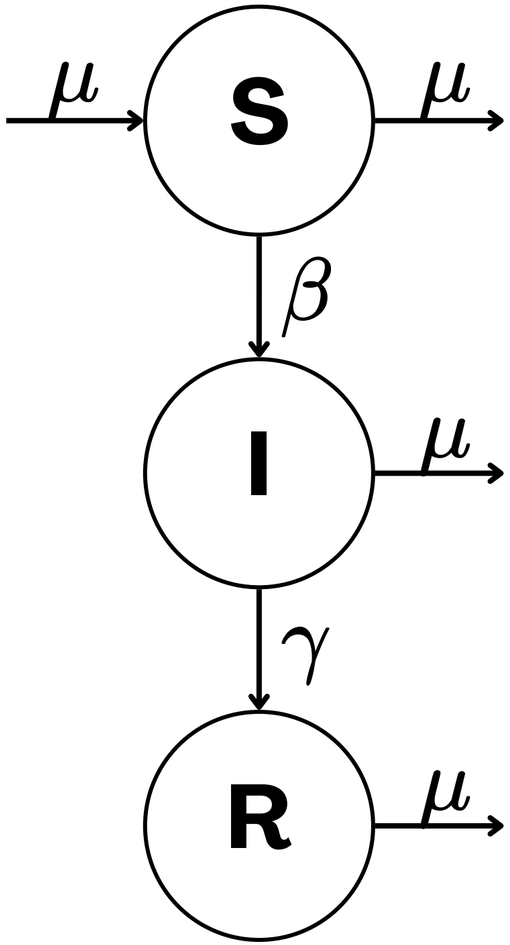
Overview

1. Epidemic & Gossip Models
2. Distributed Systems
3. Gossip Protocols
4. Applications

Why bother?

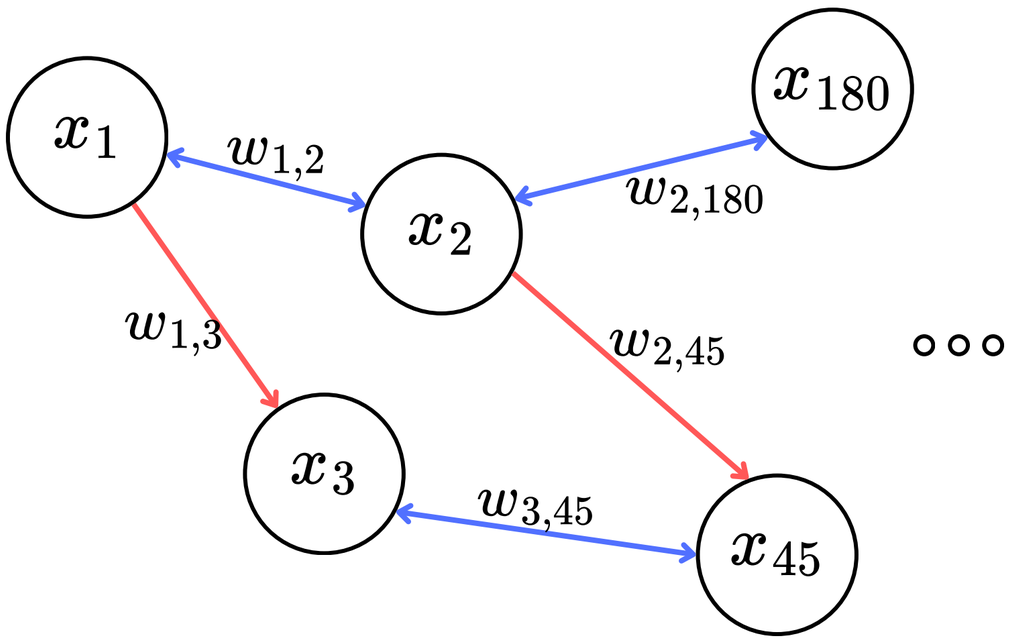
Scalable & Resilient approach where performance matters.

Epidemic & Gossip Models



$$\begin{cases} \dot{S} &= \mu(1-S) - \beta I S \\ \dot{I} &= \beta I S - (\mu+\gamma) I \\ \dot{R} &= \gamma I - \mu R \end{cases}$$

Distributed Systems



$\circ\circ\circ$

$x_i = \{\text{id}, \text{value}, \text{timestamp}\}$

Why: Ensure scalability, reliablity, privacy

Examples:

1. Blockchain
2. BitTorrent
3. Cloud Infrastructures(Amazon S3)

Gossip Protocols

What: Rules for sharing **information** $f(\text{value}_i)$

How:

1. **Decentralized** communication
2. Exchanges are **synchronized**
3. **No perfect** communication
4. **Concurrent** information

Gossip Protocols ~ Anti-Entropy

- **SI** Model-inspired
- **Continuous** updates
- **Robust**
- Convergence: $\mathcal{O}(\log n)$

A. Push Style

```
on timeout do
  Q ← random(P)
  send PUSH(P.value, P.time) to Q
  set timeout Δ

on receive PUSH(v, t) do
  if P.time < t then
    P.value ← v
    P.time ← t
```

Gossip Protocols ~ Anti-Entropy

B. Pull Style

```
on timeout do
  Q ← random(P)
  send PULL(P, P.time) to Q
  set timeout Δ

on receive PULL(Q, t) do
  if P.time > t then
    send REPLY(P.value, P.time) to Q

on receive REPLY(v, t) do
  if P.time < t then
    P.value ← v
    P.time ← t
```

C. Push-Pull Style

```
on timeout do
  Q ← random(P)
  send PUSHPULL(P,P.value, P.time) to Q
  set timeout Δ

on receive PUSHPULL(Q, v, t) do
  if P.time < t then
    P.value ← v
    P.time ← t
  else
    send REPLY(P, P.value, P.time) to Q

on receive REPLY(Q, v, t) do
  if P.time < t then
    P.value ← v
    P.time ← t
```

Gossip Protocols ~ Rumor-Mongering Approach

- **SIR** model-inspired
- **Rare** updates
- More **efficient**
- Usually **Push-style**
- Convergence: $\mathcal{O}(\log n)$

Stop Rule:

- **When**
  - ***Blind***: Evaluation at **each turn**
  - ***Feedback***: Evaluation after a **notice**
- **How**
  - ***Counter***: Stop **after k** evaluations
  - ***Coin***: Stop with **probability 1/k**

Distributed Aggregation of the Mean

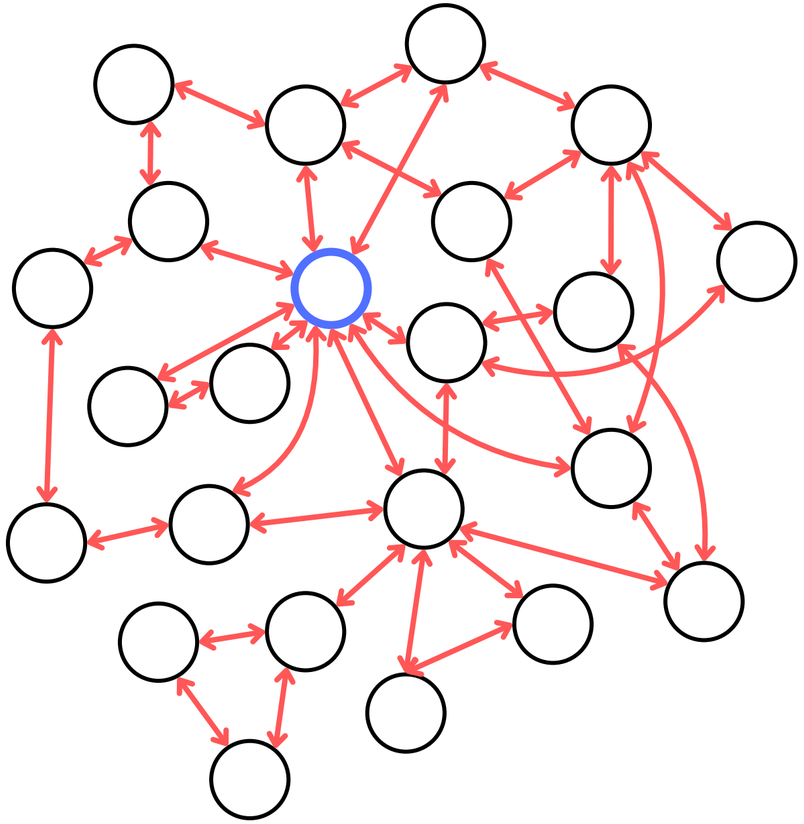
```
on timeout do
  Q ← random(getPeers(P))
  P.state ← P.value
  send PUSHPULL(P,P.state) to Q
  set timeout Δ

on receive PUSHPULL(Q, s) do
  send REPLY(P, P.state) to Q
  P.state ← (P.state + s) / 2

on receive REPLY(Q, v) do
  P.state ← (P.state + v) / 2
```

- Push-Pull style
- At each round:
  - **Average** of the states: **constant**
  - **Variance** of the states: **shrinks**
- **Overapproximation** in case of concurrent messages
- Performance depends on ***getPeers()***

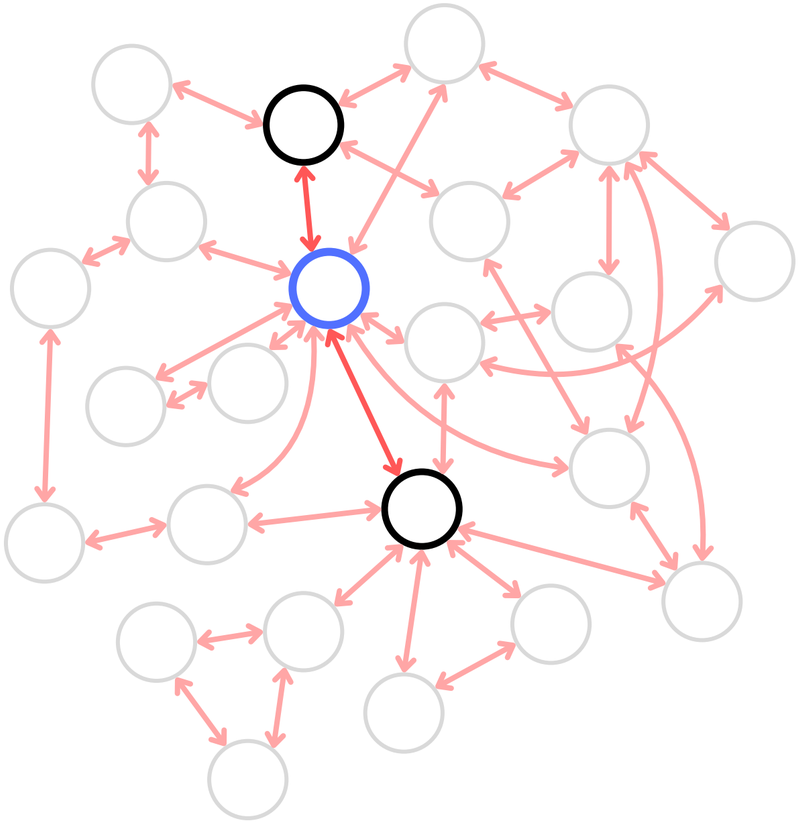
Peer Sampling problem: NewsCast



- Churning:
  - Nodes **enter** and **leave** the network

- **NewsCast**:
  - Use a **Peer View** of size ***k***
  - Peer View contains **descriptors**: *{P, P.timestamp}*
  - Message: *{P.peerView* U *P.descriptor}*

Peer Sampling problem: NewsCast



k = 2

- Churning:
  - Nodes **enter** and **leave** the network

- **NewsCast**:
  - Use a **Peer View** of size ***k***
  - Peer View contains **descriptors**: *{P, P.timestamp}*
  - Message: *{P.peerView U P.descriptor}*

```
on receive MESSAGE(Q.peerView) do
  P.peerView ← extractNewest(P.peerView U Q.peerView, k)
```

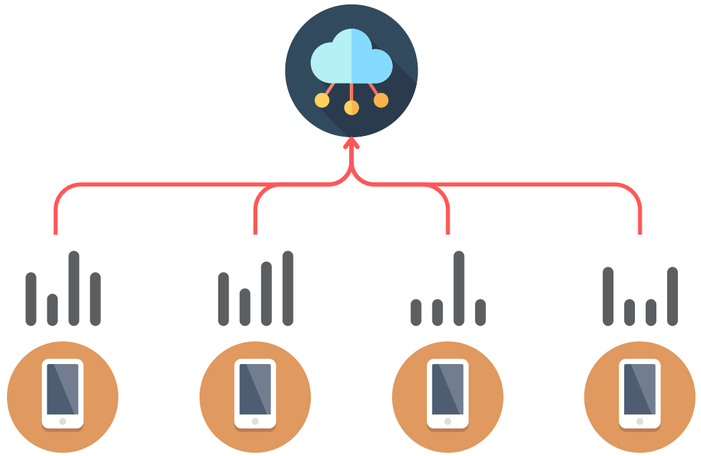
Gossip Learning

- Distributed Learning approach
  - Build ML models from **distributed data**
  - Each node trains **its model** from **its data**

- Alternative to **federated learning**

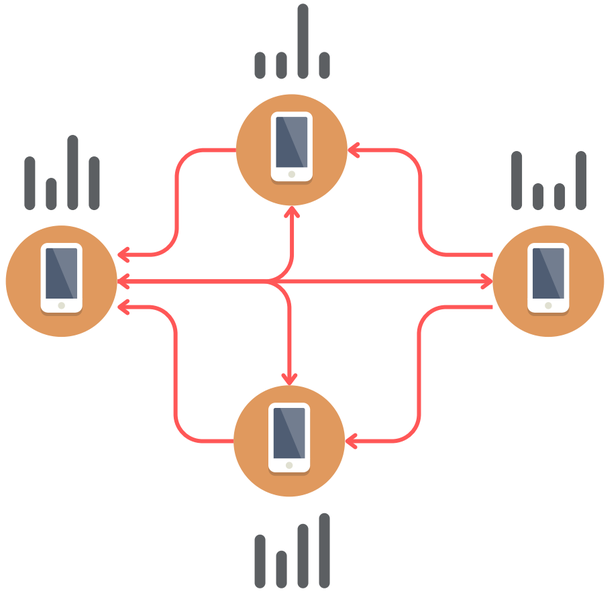
- Ok for **Supervised** Learning
- Use of **Stochastic Gradient Descend**

- Messages: *{P.modelWeights, P.timestamp}*



Federated Learning approach

Gossip Learning approach



Gossip Learning

```
initialize
  P.currentModel ← initModel()
  P.lastModel ← P.currentModel
  set eta, X, y as learingRate, P.data, P.labels

on timeout do
  Q ← random(getPeer(P))
  send MESSAGE(P.currentModel) to Q
  set timeout Δ

on receive MESSAGE(Q.model) do
  P.currentModel ← update(merge(P.lastmodel,
Q.model))

  P.lastModel ← Q.model
```

```
on update(model) do
 weights, timestamp ← model
 eta ← 1 / (learningRate * timestamp)

 weights ← (1 - eta * learningRate) * weights +
eta SGD(weights, P.data, P.labels)

 return (weights, timestamp + 1)

on merge(model1, model2) do
  weights1, timestamp1 ← model1
  weights2, timestamp2 ← model2

  newWeights ← (weights1 * timestamp1 + weights2
* timestamp2) / (timestamp1 + timestamp2)

  newTimestamp ← timestamp1 + timestamp2

  return (newWeights, newTimestamp)
```

Bibliography

1. P. Bellavista, L. Foschini, and A. Mora, *Decentralised learning in federated deployment environments: A system-level survey*. ACM Computing Surveys (CSUR), 54(1):1–38, 2021.
2. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, *Epidemic algorithms for replicated database maintenance*. Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, pages 1–12, 1987.
3. L. Ghiro, F. Restuccia, S. D'Oro, S. Basagni, T. Melodia, L. Maccari, and R. Lo Cigno, *A blockchain definition to clarify its role for the internet of things*. 2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet), pages 1–8. IEEE, 2021.
4. L. Giaretta and S. Girdzijauskas, *Gossip learning: Off the beaten path*. 2019 IEEE International Conference on Big Data (Big Data), pages 1117–1124. IEEE, 2019.
5. I. Hegedus, G. Danner, and M. Jelasity, *Decentralized learning works: An empirical comparison of gossip learning and federated learning*. Journal of Parallel and Distributed Computing, 148:109–124, 2021.
6. M. Jelasity, A. Montresor, and O. Babaoglu, *Gossip-based aggregation in large dynamic networks*. ACM Transactions on Computer Systems (TOCS), 23(3):219–252, 2005.
7. A. Montresor. *Gossip and epidemic protocols*.
8. R. Ormandi, I. Hegedus, and M. Jelasity, *Gossip learning with linear models on fully distributed data*. Concurrency and Computation: Practice and Experience, 25(4):556–571, 2013.
9. F. Turazza, M. Pietri, M. Picone, and M. Mamei, *Fedbgs: A blockchain approach to segment gossip learning in decentralized systems*. In 2025 IEEE 45th International Conference on Distributed Computing Systems Workshops (ICDCSW), pages 760–770. IEEE, 2025.

Thanks

Savorgnan Enrico

Distributed Aggregation of the Mean

Concurrent messages:

```
on timeout do
  Q ← random(getPeers(P))
  P.state ← P.value
  send PUSHPULL(P,P.state) to Q
  set timeout Δ

on receive PUSHPULL(Q, s) do
  d ← P.state - (P.state + s)/2
  send REPLY(P, d) to Q
  P.state ← P.state - d

on receive REPLY(Q, d) do
  P.state ← P.state + d
```

Distributed Aggregation of the Mean

Convergence

- Goal: $\mathbb{E}[\sigma^2(t+1)] = \rho\sigma^2(t) = \frac{1}{2\sqrt{e}}\sigma^2(t)$
- Assumptions: $\forall i, \mathbb{E}[x_i] = 0 \quad \wedge \quad \mathbb{V}[x_i] < \infty \quad \wedge \quad \forall i \neq j,\ \mathrm{corr}(x_i, x_j) = 0$
  getPeer() random "enough"
- Result 1: $\mathbb{E}[\sigma^2(t+1)] \approx \mathbb{E}[2^{-\mathbf{x}}]\mathbb{E}[\sigma^2(t)]$
- Idea 2: getPeer() creates a random permutation of the nodes,
  and pairs each of them to another one, picked at random.
- Result 2: $\mathbb{P}[x = x_j] = \mathbb{P}[x' = x_j - 1] = \frac{1}{(j-1)!}e^{-1} \qquad x' \sim \mathrm{Pois}(\lambda = 1)$