07.04.2025 Week 8 exercises: Gossip

Exercise 1:

What are the 3 ways of propagating information in a graph? What are the advantages and disadvantages of each one of them?

The 3 ways of propagating information in the graph are the following:

- **Pull** is bad if just a small amount of nodes are aware of the information, as in some rounds we may not distribute any information to the rest.
- **Push** is good when we have a few small number of nodes that know the information in a graph, as they will push it in every round to some others.
- **Both pull and push** is a very fast way to spread the information along the graph as it takes advantage of both approaches.

Exercise 2:

Recall from the lecture the skeleton for gossip-based peer sampling service. Three system parameters: c, H and S play an important role in the dynamics of the algorithm. c controls the view size while H and S represent the *healing* and *swapping* parameters.

Algorithm 7 The skeleton of a gossip-based peer sampling service.

```
1:  loop
2:    wait(Δ)
3:    p ← selectGPSPeer()
4:    sendPush( p, toSend() )
5:    view.increaseAge()
6:
7:  procedure ONPUSH(m)
8:    if pull then
9:      sendPull( m.sender, toSend() )
10:   onPull(m)
11:
12: procedure ONPULL(m)
13:   update(m.buffer,c,H,S)
14:   view.increaseAge()
15: procedure UPDATE(buffer,c,H,S)
16:   view.append(buffer)
17:   view.removeDuplicates()
18:   view.removeOldItems(min(H,view.size-c))
19:   view.removeHead(min(S,view.size-c))
20:   view.removeAtRandom(view.size-c)
21:
22: procedure TOSEND
23:   buffer ← ((MyAddress,0))
24:   view.shuffle()
25:   move oldest H items to end of view
26:   buffer.append(view.head(c/2 − 1))
27:   return buffer
```

Figure 1: Gossip based peer sampling [1].

1. What is the effect of increasing or decreasing H ?

   The parameter H defines how aggressive the protocol is when removing old nodes. The larger the value of H, the sooner the older items will be removed from the view and vice-versa. Note that the protocol does not actually check the liveness of the node before removing it from the view. It instead

relies on the fact that nodes which are not alive will have unrefreshed or very old descriptors in the views of other nodes.

2. Can you explain similarly the effect of increasing or decreasing S ?

   From the algorithm above, we see that the items received from the peer are appended to the end of the view. Then the first S items *i.e.*, items owned by the node get removed from the view. Recall that these were exactly the items that were sent to the peer previously. Thus a large value of S will result in higher probability of keeping items received from the peer while a low values of S results in mixing of items from both peers. S controls the number of items swapped between peers, hence the name *swapping* parameter.

Exercise 3:

Gossip can be used not only for information dissemination but also for information processing. In the lecture we saw how to compute mean of values in a network in a Gossip-style protocol. Let us recall the algorithm below. For an in-depth discussion, refer to [1].

```
1:  loop
2:      wait Δ
3:      p ← Random Peer
4:      sendPush(p, x)
5:
6:  procedure OnPush(m)
7:      sendPull(m.sender, x)
8:      x ← (m.x + x)/2
9:  procedure OnPull(m)
10:     x ← (m.x + x)/2
```

The algorithm can be shown to converge to the global average in the network. Now that we have established that we can compute the average, can we also compute other means ? Design a Gossip-style algorithm to compute geometric mean of values in the network.

Let $x_1, x_2, \ldots, x_n$ be the attributes of n nodes. If we can compute the average of x_i's, we can also computes averages of any transformations on x_i. Precisely, $g(x_1, x_2, \ldots, x_n) = f^{-1}(\sum_{i=1}^{N} f(x_i)/n)$ where f is the transformation. Interestingly, the choice of appropriate transformation leads us to the desired mean. For example, $f(x) = \log_e x, f^{-1}(x) = e^x$ gives us the geometric mean and $f(x) = 1/x, f^{-1}(x) = x$ gives us the harmonic mean.

Exercise 4 (Gossip-based resource assignment)**:**

Resource assignment to services and applications is crucial in many distributed settings. At one end of the spectrum, a centralized server can maintain a database of resource availability for the whole network. At the other end, one can think of a fully decentralized solution that automatically partitions the available nodes into "slices", taking into account specific attributes of the nodes (such as available bandwidth, storage capacity or number of processing units). The goal of this exercise is to design a gossip-based protocol that can achieve such a decentralized slicing.

We assume:

- The network is composed of a set of n nodes. Nodes are uniquely identified by their IP address.

- There are no failures, node arrivals nor departures.
- Each node is connected to a set of $k = log(n)$ other nodes called neighbors. This means that each node can communicate with this set of neighbors.
- We assume that the neighbors of the node change periodically (every t milliseconds), meaning every node periodically gets a new set of random neighbors (picked uniformly).
- A peer-sampling service provides the current view of k neighbors.
- All nodes execute the same code.
- **Hint**: attribute a random number to each node. Assume that the random number generator is uniform and there are no collisions.

1. [7 points] Design a gossip-based algorithm that arbitrarily divides the system into 10 slices of approximately equal size. Write the pseudocode of your protocol, so that by the end each node knows the other approximately $(n/10) - 1$ nodes in its slice.

```
1:  r ← Sample a random number btw [0,1]
2:  mySlice ← {}
3:
4:  loop                                  ▷ Active Thread
5:      wait Δ
6:      j ← Random Peer
7:      sendPush(j, toSend())
8:
9:  procedure OnPush(m)                   ▷ Passive Thread
10:     if pull then
11:         sendPull(m.sender, toSend())
12:     Merge(m)
13:
14: procedure OnPull(m)                   ▷ Passive Thread
15:     Merge(m)
16:
17: procedure toSend
18:     m.r ← r
19:     m.sender ← i                      ▷ I'm node i
20:     return m
21:
22: procedure Merge(m)
23:     if m.r in my percentile then
24:         mySlice.append(m.sender)
```

2. [8 points] Assume now that each node i has an attribute x_i that measures the availability of some resource (*e.g.*, bandwidth) on node i. We assume that there exists a total ordering over the domain of the attribute values, so that the values in the network $(x_1, ..., x_n)$ can be ordered. Let us also assume that there is a slice specification that defines an ordered partitioning of the nodes. That is, the slice specification is a list of positive real numbers $s_1, ..., s_m$ such that for all $u < v, i \in S_u$ and $j \in S_v$ we have $x_i \leq x_j$. As opposed to question 1., a node does not need to know the other nodes in its slice.

   Design a gossip-based algorithm that assigns each node to one of 10 slices (m = 10) in such a way that it satisfies the slice specification, using only local message exchange with currently known neighbors. That is, we want each node to find out which slice (*i.e.*, which 10th-percentile) it belongs to. For example, each node can then know, assuming x represents the available bandwidth, if it belongs to the 10% of nodes with the highest available bandwidth.

```
1:  r ← Sample a random number btw [0,1]
2:  x is my resource
3:
4:  loop                                   ▷ Active Thread
5:      wait Δ
6:      j ← Random Peer
7:      sendPush(j, toSend())
8:
9:  procedure OnPush(m)                    ▷ Passive Thread
10:     sendPull(m.sender, toSend())
11:     Merge(m)
12:
13: procedure OnPull(m)                    ▷ Passive Thread
14:     Merge(m)
15:
16: procedure toSend
17:     m.r ← r
18:     m.x ← x
19:     m.sender ← i                       ▷ I'm node i
20:     return m
21:
22: procedure Merge(m)
23:     if ((x − m.x) ∗ (r − m.r) ≤ 0) then
24:         r ← m.r                        ▷ Swap random numbers
```

Try playing with the Gossip code at https://github.com/rishi-s8/gossip-sim.

References

[1] Márk Jelasity. *Gossip-based protocols for large-scale distributed systems*. PhD thesis, mi, 2013.