COP 5615 – FALL 2020

Project 2 – Gossip Simulator

Yu-Peng Chen
7904-3193

Table of Contents

1. Summary……1
2. Problems and Algorithms……1
3. Program Structure……4
4. Results……6
5. Discussions……9
6. Bonus……9
7. References……11

1. Summary

The goal of this project is to implement two Gossip-based algorithms (i.e., the **Gossip algorithm** and the **Push-Sum** algorithm) to solve two problems (i.e., the **information dissemination** problem and the **aggregate computation** problem) respectively in simulated distributed systems.

This document will first introduce the definition of each algorithm implemented in the program and the topologies used to simulate the actual network, and then describe the program structure that is based on the actor model in F#. Different combinations of hyperparameters (e.g., topologies, number of nodes in a network, definition criterion, etc.) are experimented and the results are shown and discussed in the Result and Discussion sections.

2. Problems and Algorithms

a) Information Dissemination

a-1) Algorithm: the **Gossip** algorithm

A participant is told a rumor and starts to spread it. Each participant who has heard the rumor then starts to spread it in each step. A participant stops gossiping once it has heard the rumor for a specific number of times (e.g., 10). The process terminates when **all participants have heard the rumor for the specified number of times** (e.g., 10).

a-2) Parameters

- Number of times that the participant should hear the rumor
- Frequency of spreading the rumor (explained in detail in the Program Structure section)
- Topology
- Number of nodes within a network

b) Aggregate computation

b-1) Algorithm: the **Push-Sum** algorithm

For this problem, the goal is to compute efficiently the 1) **sum** or 2) **average** of the value stored in each node in a fault-tolerant way. In this project, the Push-Sum algorithm is implemented as described in Kempe et al.'s work [1]:

Algorithm 1 Protocol Push-Sum

1: Let $\{(\hat{s}_r, \hat{w}_r)\}$ be all pairs sent to i in round $t-1$
2: Let $s_{t,i} := \sum_r \hat{s}_r, w_{t,i} := \sum_r \hat{w}_r$
3: Choose a target $f_t(i)$ uniformly at random
4: Send the pair $(\frac{1}{2}s_{t,i}, \frac{1}{2}w_{t,i})$ to $f_t(i)$ and i (yourself)
5: $\frac{s_{t,i}}{w_{t,i}}$ is the estimate of the average in step t

Each node initially holds a value s that is the same as its index (i.e., $node_i$ has the value i) and a weight w .

- To compute the **sum** of the values stored in all nodes, $\boldsymbol{w_i}$ **is initialized to 1 for all nodes**.
- As for computing the **average** of the values stored in all nodes, **only one node starts with** $\boldsymbol{w_i = 1}$ **while all other nodes have the weight** $\boldsymbol{w_j}$, $\boldsymbol{where\ j \neq i}$ **initialized to 0** (e.g., $w_0 = 1,\ w_j = 0, where\ j \neq 0$)

The protocol starts with each node sending the pair of initialized value ($s_{t=0,i}, w_{t=0,i}$) to itself, and then maintains the pair ($s_{t,i}$, $w_{t,i}$) following the protocol shown above**.** It terminates when **the value of the ratio** $\frac{s_{t,i}}{w_{t,i}}$ **does not change by more than a specific value in a certain number of consecutive rounds.**

b-2) Parameters

- The difference between the ratio $\frac{s_{t,i}}{w_{t,i}}$ in consecutive rounds
- The number of rounds that the ratio does not change
- Frequency of spreading the rumor (explained in detail in the Program Structure section)
- Topology
- Number of nodes within a network

c) Topology

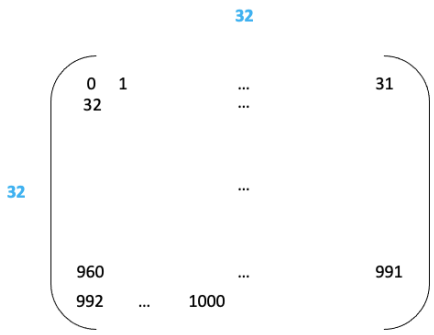
c-1) Full Network

Every actor can talk directly to any other actor.

c-2) 2D Grid

Actors form a 2D grid. If the specified number of nodes **N** is not a perfect square, the grid will be implemented in the following way:

- Calculate the square root of N: $\sqrt{\boldsymbol{N}}$
  - E.g., let N = 1000, $\sqrt{N} = 31. \ldots .$
- Round up to the next integer: $\lceil\sqrt{\boldsymbol{N}}\rceil$
  - E.g., let N = 1000, $\lceil\sqrt{N}\rceil = 32.$
- Layout the $\lceil\sqrt{\boldsymbol{N}}\rceil$ **by** $\lceil\sqrt{\boldsymbol{N}}\rceil$ grid
  - E.g., let N = 1000, we will have a 32 * 32 grid.
- The number of nodes will remain the same as **N** (i.e., it will not be increased to a perfect square, only the layout of the grid will be changed accordingly) to allow for direct comparison between different topologies.
  - E.g., let N = 1000, we will have a 32 * 32 grid with only 1000 nodes placed from left to right, top to bottom (instead of 1024 nodes), as shown in the following figure:



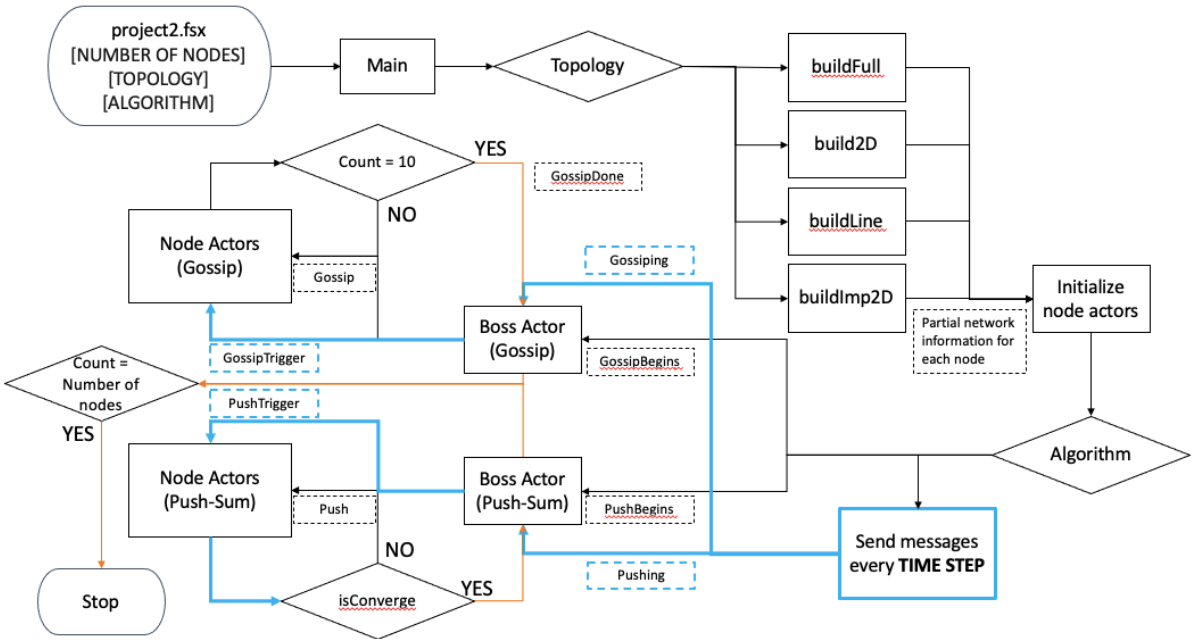
c-3) Line

Actors are arranged in a line. Each actor has only 2 neighbors.

c-4) Imperfect 2D Grid

Similar to the 2D Grid topology, with an extra random neighbor selected from all actors added to each node.

3. Program Structure



In this section, I will describe two critical issues I encountered and the corresponding solutions I came up with during the implementation.

a) How to send messages periodically?

Since an actor is message-driven, it can only send out messages when it receives a message. To trigger each participant who has heard the rumor to spread it in each step simultaneously, several ways are tried out:

- A node actor carries out two actions upon receiving a message:
  1) send message to a random neighbor and 2) trigger itself by sending a message to itself without increasing the counter. This solution didn't work. My hypothesis is that this design result in the case where the mailbox of each node was filled with unnecessary messages after a while.
- A node actor carries out two actions upon receiving a message:
  1) send message to a random neighbor and 2) trigger the sender by sending a message back to it without increasing its counter. This solution didn't work either. My hypothesis is the same as the one described above.
- **Eventually, the implementation that has worked well is the following (colored in blue in the above task flow figure):**
  a while loop was implemented in the main process to ask the boss periodically (e.g., every 200 milliseconds) to trigger the node actors (nearly) simultaneously. In the case of the Gossip algorithm, the boss has to keep track of the nodes that have heard the rumor and are still willing to gossip (i.e., both the nodes that have no

knowledge of the rumor and the ones that have heard enough should not be triggered).

b) How should I distribute the network information?

As we have discussed in the lectures, the central process should only distribute partial information that is necessary. Therefore, the program is implemented in the way that each node only knows its neighbor without having the knowledge of the whole network.

c) How do I sum up the pair ($s_{t,i}$, $w_{t,i}$) at a given moment of time before sending out half of the values?

Due to an actor's nature of being message-driven, the following method is implemented to allow the actors to perform their tasks (nearly) simultaneously in each time step. **The actors recognizes two messages: 1) Push and 2) PushTrigger.**

- For the "Push" message, each node actor only sums up the pair ($s_{t,i}$, $w_{t,i}$) and does not send messages to its neighbors.
- **The "PushTrigger" message will only be sent by the boss actor, which is triggered periodically by the main process**, as shown in the above task follow figure with a blue arrow. Upon receiving the "PushTrigger" message, each node actor performs the following tasks:
  1) computes the ratio $\frac{s_{t,i}}{w_{t,i}}$,
  2) decides whether to converge,
  3) send half of the ratio to its neighbors using a "Push" message,
  4) send half of the ratio to itself using a "Push" message,
  5) and sets the pair ($s_{t,i}$, $w_{t,i}$) to zeros.

  For task 5) described above, the idea behind it is that half of the ratio which is supposed to be kept by the node itself will be added to the pair ($s_{t,i}$, $w_{t,i}$) when the node receives the "Push" message that was sent to itself.

4. Results

a) Gossip

Hyperparameter:

- Number of times that the participant should hear the rumor: **10**
- Frequency of spreading the rumor: **300** milliseconds

a-1) Full Network

Number of Nodes	500	1000	2000	5000	10000
Time (milliseconds)	605	608	610	695	1693

a-2) 2D Grid

Number of Nodes	500	1000	2000	5000	10000
Time (milliseconds)	895	908	1218	1587	2009

a-3) Line

Number of Nodes	500	1000	2000	5000	10000
Time (milliseconds)	8467	14018	18290	32167	46337

a-4) Imperfect 2D Grid

Number of Nodes	500	1000	2000	5000	10000
Time (milliseconds)	608	615	623	739	1572

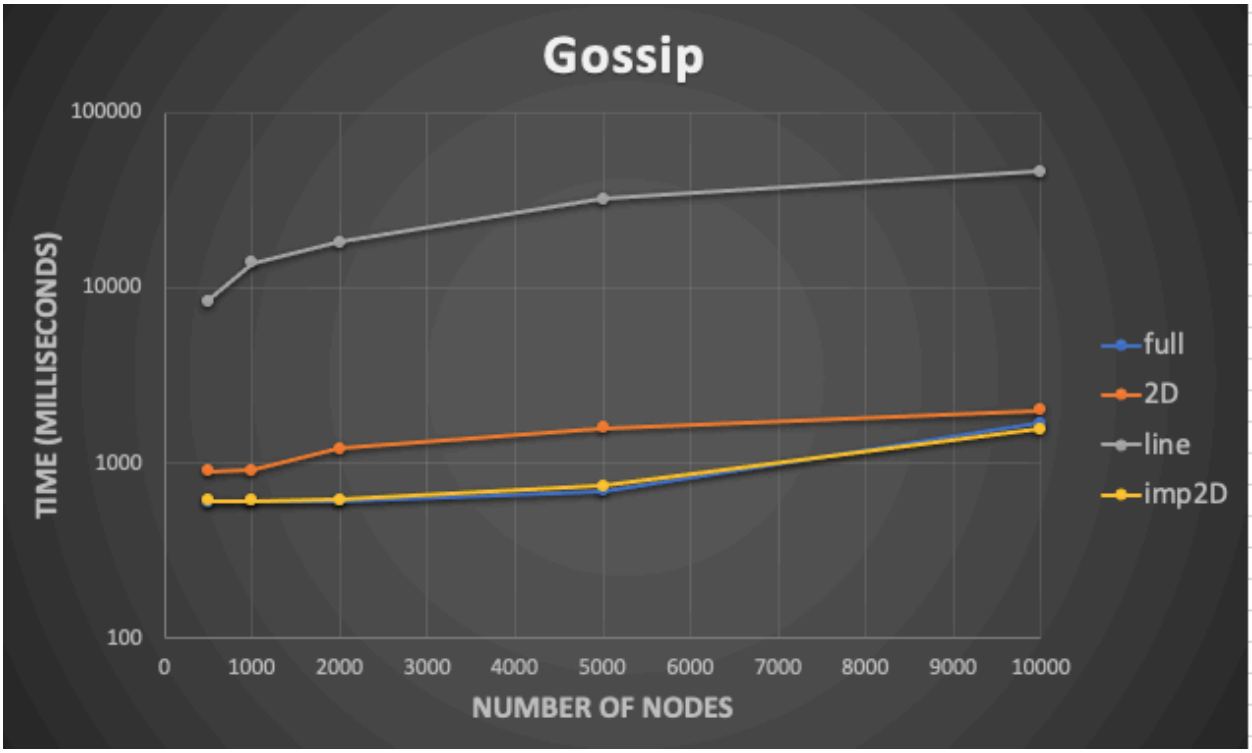


Figure 1. Gossip

b) Push-Sum

Hyperparameter:

- The difference between the ratio $\frac{s_{t,i}}{w_{t,i}}$ in consecutive rounds: $\mathbf{10^{-10}}$
- The number of rounds that the ratio does not change: **5**
- Frequency of spreading the rumor (explained in detail in the Program Structure section): **300** milliseconds

b-1) Full Network

Number of Nodes	50	100	200	500	1000
Correct Ratio (average)	24.5	49.5	99.5	249.5	499.5
Time (milliseconds)	22091 (24.5)	23592 (49.5)	26093 (99.5)	26196 (249.5)	26699 (499.5)

b-2) 2D Grid

Number of Nodes	50	100	200	500	1000
Correct Ratio (average)	24.5	49.5	99.5	249.5	499.5

Time (milliseconds)	49791 (22.75)	55792 (65.46)	61193 (100.68)	69895 (incorrect ratio)	71103 (incorrect ratio)

b-3) Line

Number of Nodes	50	100	200	500	1000
Correct Ratio (average)	24.5	49.5	99.5	249.5	499.5
Time (milliseconds)	59393	76493 (48.74)	94044 (incorrect ratio)	91196 (incorrect ratio)	96500 (incorrect ratio)

b-4) Imperfect 2D Grid

Number of Nodes	50	100	200	500	1000
Correct Ratio (average)	24.5	49.5	99.5	249.5	499.5
Time (milliseconds)	37493 (24.69)	42893 (49.54)	42592 (100.02)	43197 (251.47)	43503 (498.91)

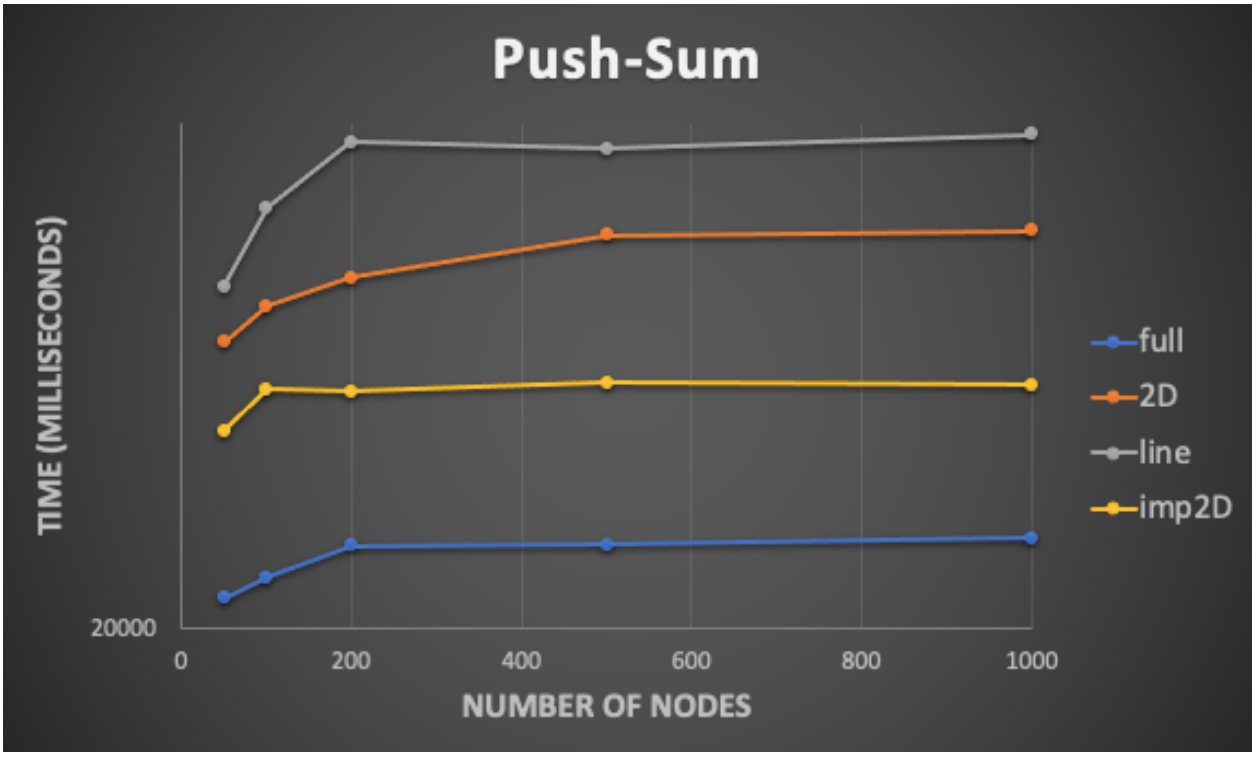


Figure 2. Push-Sum

5. Discussions

a) Gossip

- In general, with the hyperparameter specified in the result section, the comparison of the Gossip algorithm's performance on the four topologies in terms of **convergence time** is: **full < imp2D < 2D < line**. This result makes sense since each node in the full network can talk directly to any other nodes, which results in a faster spreading of the rumor. One interesting finding is that, when the node number is as large as 10,000, the performance on an imp2D network is actually better than the performance on a full network.
- The program cannot converge in a full topology with 20,000 nodes while it converges in the other three topologies with 20,000 nodes. The bottleneck is not because of the dissemination process but the process of building the topology. Every node has to keep track of too much information, which results in excessive runtime. It took the program 143036 milliseconds to build a full network with 10,000 nodes.
- In practice, compared with a full network, **an imp2D network would be the best** considering the memory space and speed trade-off.

b) Push-Sum

- In general, with the hyperparameter specified in the result section, the comparison of the Push-Sum algorithm's performance on the four topologies in terms of **convergence time** is: **full < imp2D < 2D < line**. This result makes sense due to the same reason discussed in the 5.a section above.
- The algorithm performs the most accurately and fastest in **a full network**. It can converge to the correct ratio in an imp2D network and a full network, with a imp2D network having slight errors, while it cannot converge to the correct ratio in a line network and a 2D network.
- The convergence times for each network regarding the number of nodes (from 50 to 1000) are approximately the same. The hypothesis is that the convergence time may be influenced more by the hyperparameters, i.e., 1) the difference between the ratio $\frac{s_{t,i}}{w_{t,i}}$ in consecutive rounds and 2) the number of rounds that the ratio does not change.

6. Bonus

a) Implementation

For the bonus part, please refer to the "project2-bonus.fsx" file, in which a failure model is implemented by intentionally stop the message transmission process of a specified number of nodes.

a-1) New parameter

Command line:

```
dotnet fsi --langversion:preview proj2.fsx
[NUMBER OF NODES] [TOPOLOGY] [ALGORITHM] [PERCENTAGE OF FAILURE]
```

The fourth parameter is introduced to define the failure model. The parameter is an integer from 0 to 100 that represents **the percentage of nodes that will be dead**. A dead node means that **it will not send any messages to its neighbor**.

a-2) Mechanism

The failure model is implemented with the following steps

1) Calculate the number of nodes that will be dead:
   [NUMBER OF FAILED NODES] = [NUMBER OF NODES] * [PERCENTAGE OF FAILURE] / 100
2) Randomly select the nodes that will be dead based on the result in 1)
3) During the initialization of each node, the failed node's convergence flag is set to true, which will stop the node from sending messages.

b) Results

Only the full network was tested before the submission deadline of this report.

b-1) Gossip (Full Network)

- Number of nodes 500

Failure percentage	1%	2%	5%	10%	20%
Time (milliseconds)	23094 (24.5)	NA (24.1)	NA (25.4)	NA (23.97)	NA (26.25)

b-2) Push-Sum (Full Network)

- Number of nodes 50 (correct ratio: 24.5)

Failure percentage	1%	2%	5%	10%	20%
Time (milliseconds)	611	899	1205	906 / NA	1204 / NA

c) Discussions

Only the full network was tested before the submission deadline of this report.

c-1) Gossip (Full Network)

- Fault tolerance is higher for the Gossip algorithm.

c-2) Push-Sum (Full Network)

- Fault tolerance is low for the Push-Sum algorithm. Failure percentage that is equal to or large than 2% results in non-convergence.

7. References

[1] D. Kempe, A. Dobra, and J. Gehrke, “Gossip-based computation of aggregate information,” in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, 2003, pp. 482–491.