Project 2: Gossip Simulator

Group Members

- Amit Asish Bhadra (66494087)
- Rishab Aryan Das (05369273)

Topologies

At a point of time, an actor sends the message to any one of its neighbors selected randomly.

- Line: Here each actor talks to its nearest 2 neighbors except at the start and end index. This calls the functions **find_left_neighbor** and **find_right_neighbor**
- Full: Here each actor is able to talk to every other actor in the network
- 2D: We first round off the number of nodes to the lowest square which is just above the number. Then we create a 2D grid of size sqrt(new number) * sqrt(new number). In this network, each actor can talk to the nearest actors in the grid. This calls the functions **find_left_neighbor_2d, find_right_neighbor_2d, find_up_neighbor_2d, and find_down_neighbor_2d**
- Imp2D: On top of the 2D grid approach, we sample another random actor which isn't a neighbor.

How to find neighbors:

- **find_left_neighbor (i, maxIndex) :** If i is not 1 then return (i-1) else return -1
- **find_right_neighbor (i, maxIndex) :** If i is not maxIndex then return (i-1) else return -1
- **find_left_neighbor_2d (i, maxIndex) :** If (i-1)%maxIndex is not 0 then return (i-1) else return -1
- **find_right_neighbor_2d (i, maxIndex) :** If (i+1)%maxIndex is not 1 then return (i+1) else return -1
- **find_left_neighbor_2d (i, maxIndex) :** If (i-maxIndex) is greater than 0 then return (i-maxIndex) else return -1
- **find_left_neighbor_2d (i, maxIndex) :** If (i+maxIndex) is less than equal to then return (i-i+maxIndex) else return -1

How to store the neighbors:

- For each topology, call the correct functions and store it in a temp array
- Filter out the array for any entry of value -1
- Save the new array as a state variable called neighbors for that particular actor

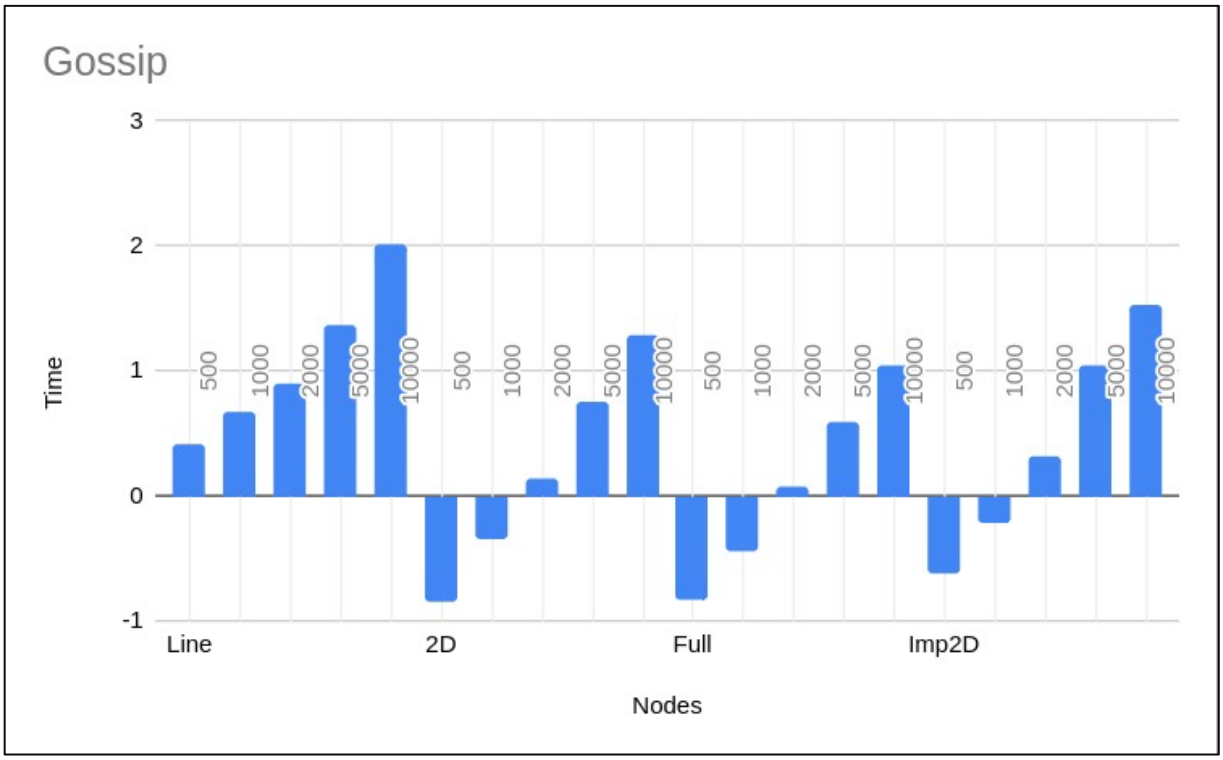
Gossip Protocol

Implementation:

- We start from a certain actor
- This actor now selects one of its random neighbors and propagates the rumor
- Each actor selects a random neighbor and sends messages until they converge
- The convergence criteria is met when the actor has received the rumor 10 times
- Once all the nodes converge, we print out the time taken
- Since this is DOS, we use multiple active messages in the network.

Algorithm	Topology	Nodes	Time(seconds)	Log(Time)
Gossip	Line	500	2.56236	0.4086401461
		1000	4.70784	0.6728216945
		2000	7.875572	0.8962821063
		5000	22.973043	1.361218525
		10000	103.7761308	2.016097474
	2D	500	0.1399017	-0.8541770082
		1000	0.4451317	-0.3515114764
		2000	1.3576932	0.1328016428
		5000	5.6144223	0.7493050759
		10000	19.3643405	1.287002711
	Full	500	0.1449562	-0.8387632045
		1000	0.3590009	-0.4449044627
		2000	1.1647985	0.0662508027
		5000	3.9078222	0.5919347958
		10000	11.0559765	1.043597107
	Imp2D	500	0.2420331	-0.6161252366
		1000	0.6173405	-0.2094752306
		2000	2.1020224	0.3226373397
		5000	10.9505902	1.039437527
		10000	33.4396269	1.524261423

Here is the graph of Gossip -
X-axis : $\log_{10}$(time)
Y-axis : Topologies and number of nodes



Push-Sum Protocol

Implementation:

- Each actor has a state defined by the tuple (s, w). Initially for all actors, s is its index and w is 1
- For our implementation, we also keep the last 3 states – state_1, state_2 and state_3
- We choose one actor and ask it to start the process. This actor selects a random neighbor and sends it (s/2, w/2) and changes its own s=s/2 and w=w/2.
- The actor that receives. The message then receives the new s and w values and it adds them to its own s and w values. Similarly, before sending, it sends half of the updated s and w values and keeps half of them to itself
- This goes on and convergence is reached for an actor when the ratio s/w doesn't change by 10^-10 in 3 consecutive visits.
- Here we print out the time taken when all the actors converge.
- In my demonstration, I showcase Push sum with single active message in the system and multiple active messages in the system to show the difference in time taken to converge.

Single Active Message:

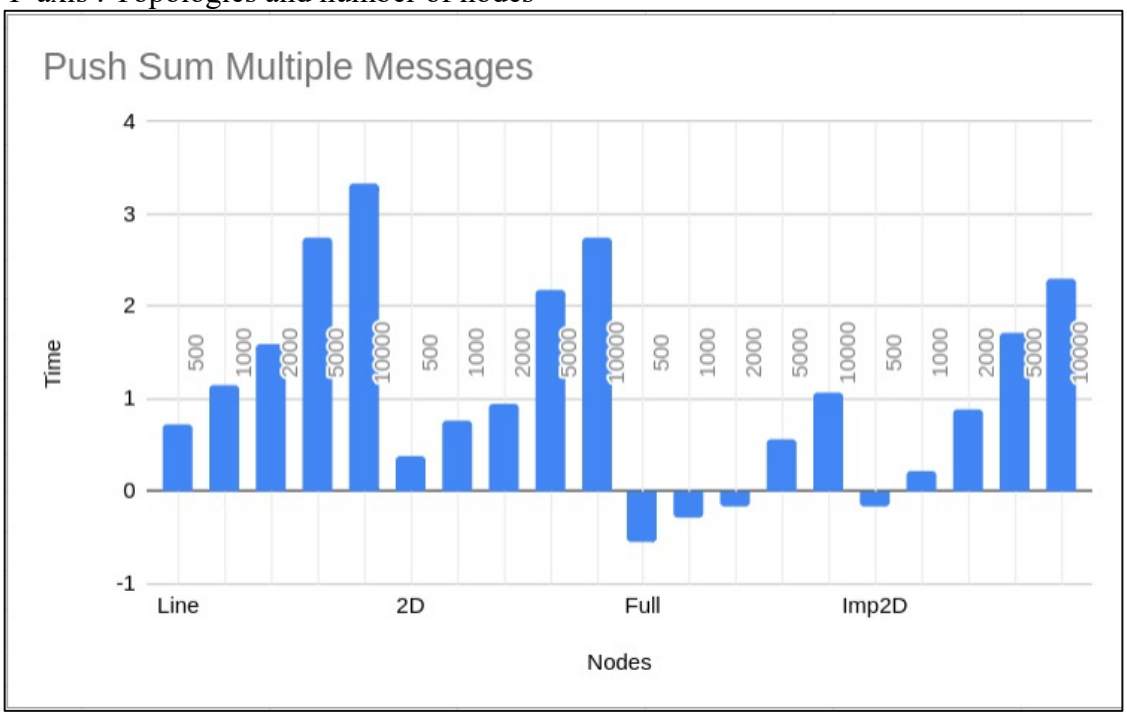
Algorithm	Topology	Nodes	Time(seconds)	Log(Time)
Push-Sum	Line	500	565.35201	2.752318941
		1000	1067.487627	3.02836285
		2000	NA	
		5000	NA	
		10000	NA	
	2D	500	10.7631637	1.031939946
		1000	31.3228528	1.49586131
		2000	147.1764773	2.167838404
		5000	1044.69414	3.018989159
		10000	1852.8	3.267828542
	Full	500	0.2658493	-0.5753644789
		1000	0.542047	-0.2659630549
		2000	0.8742512	-0.0583637629
		5000	3.4642192	0.5396053644
		10000	9.0086725	0.9546607989
	Imp2D	500	2.0554344	0.3129036207
		1000	5.6346105	0.7508639002
		2000	20.01881	1.301438258
		5000	119.724128	2.078181683
		10000	520.1034095	2.716089701

The values NA means that these topologies didn’t converge for well over an hour.

Multiple Active Messages:

Algorithm	Topology	Nodes	Time	Log(Time)
Push-Sum Multiple Messages	Line	500	5.442838	0.7358254082
		1000	13.89686	1.142916682
		2000	39.13049	1.592515286
		5000	547.09932	2.738066175
		10000	2183.94827	3.339242347
	2D	500	2.458599	0.3906877006
		1000	5.82188	0.7650632496
		2000	8.99375	0.9539408113
		5000	150.87089	2.178605452
		10000	546.13067	2.737296567
	Full	500	0.290107	-0.537441792
		1000	0.52031	-0.2837378272
		2000	0.681138	-0.1667648902
		5000	3.61961	0.5586617794
		10000	11.750318	1.07004962
	Imp2D	500	0.701298	-0.1540973996
		1000	1.694531	0.2290495183
		2000	7.65138	0.8837397714
		5000	52.33928	1.718827744
		10000	199.3648	2.299648481

Here is the graph of Push Sum(multi) -
X-axis : $\log_{10}$(time)
Y-axis : Topologies and number of nodes



Observations:

- The order of convergence for Gossip is Full < Imp2D < 2D < Line
- The order of convergence for Push Sum is Full < Imp2D < 2D < Line
- The order of convergence matches the topology because in Full we are able to send more requests and when there are multiple active messages in the network the topology with the highest neighbors always converges first. In comparison, line can send its messages only to its neighbors and thus it takes longer.
- For Push sum, the ratio of s/w in single active message averages to a value of ((n+1)*n/2*n), where n is the number of nodes in the network, during convergence and this is a logical ratio because we are adding the s values because the s values range from 1 to n and effectively all the s values are being added up.