COP5615 Project 2

Due Date: October 1st 2019 11:59PM

Isabel Laurenceau 7393-5064
Anshika Saxena 9530-5566

What Is Working

We have implemented both the Gossip and Push Sum algorithms on all six topologies. To run our program:

1) Go to the main directory where mix.exs is located
2) Build using “mix escript.build”
3) Run using “./project2 [numNodes] [Topology] [Algorithm]”

   Where numNodes can be any integer number >=2.

   Where Topology can be:

   - Full
   - Line
   - twoD
   - threeD
   - Honeycomb
   - Honeycomb_rand

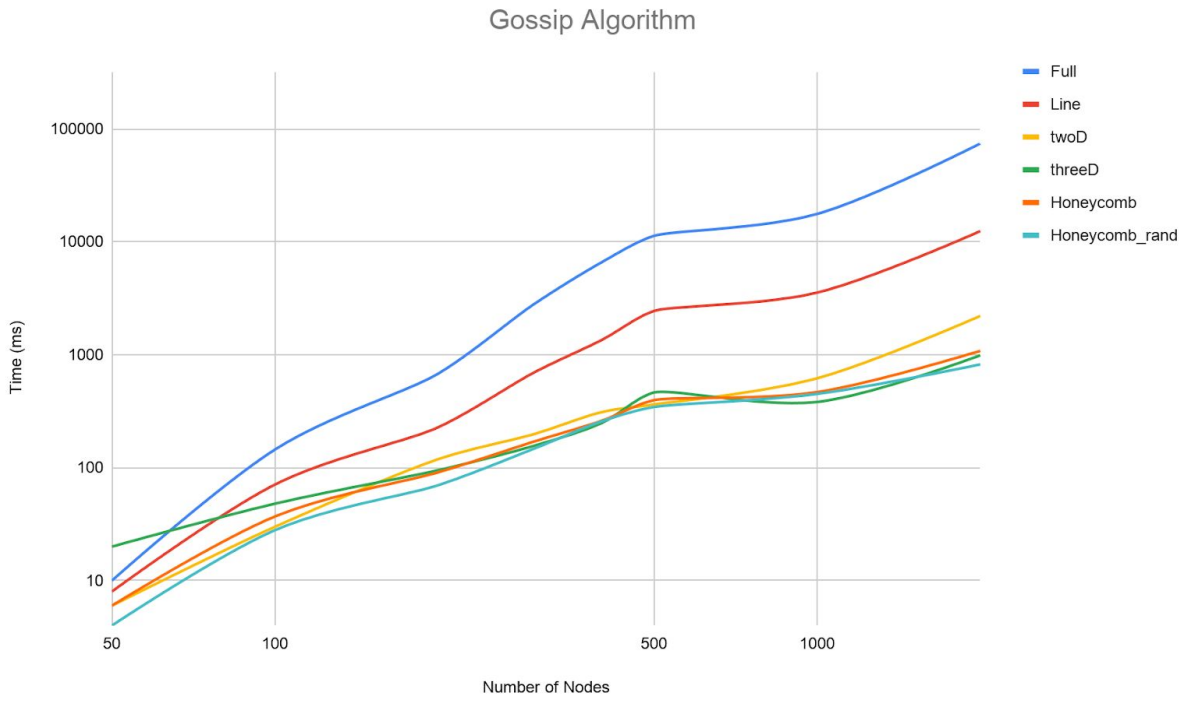
   Where Algorithm can be:

   - Gossip
   - Push

We determined the amount of time it takes to converge by using System.monotonic_time() after we send the first rumor message and then using the same function after the Supervisor dies. We take this difference and convert it to the milliseconds time unit as shown in Tables 1 and 2.

Table 1: Gossip Algorithm Run Time (ms)

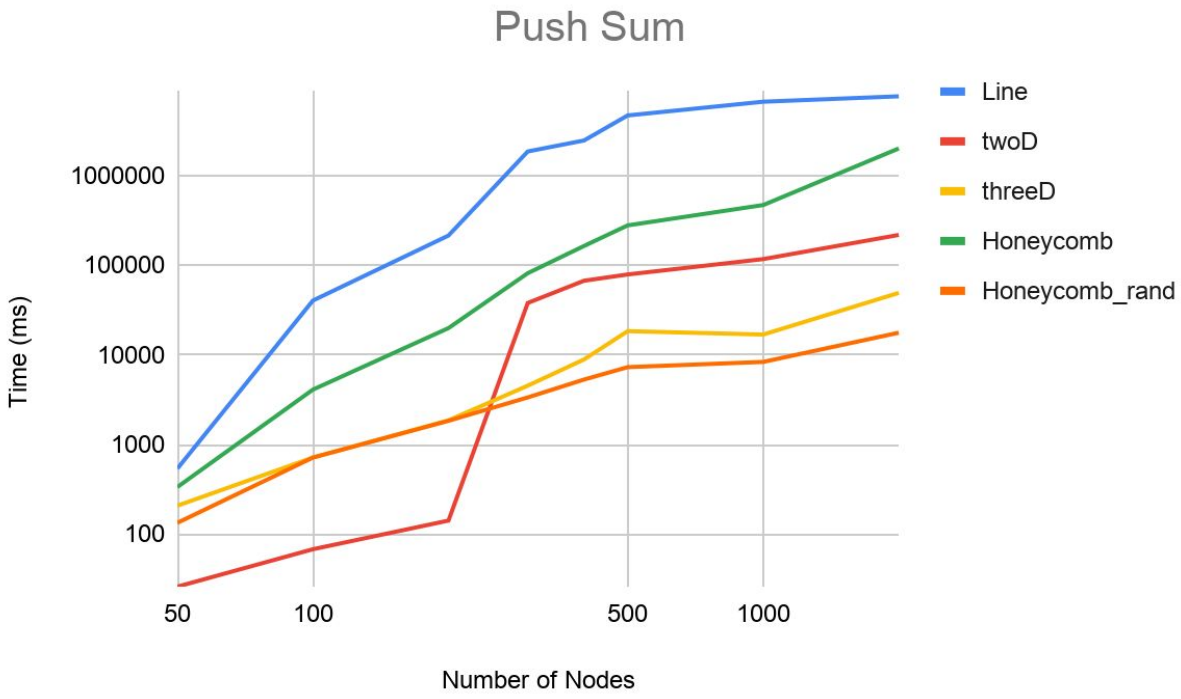
Num Nodes	Full	Line	twoD	threeD	Honeycomb	Honeycomb_rand
10	10	8	6	20	6	4
50	145	71	30	48	37	28
100	681	228	119	95	91	70
200	2796	694	198	156	170	147
300	6576	1348	310	248	260	257
400	11296	2440	363	463	396	345
500	17658	3550	619	382	467	450
1000	74277	12441	2207	989	1083	821
2000	322199	43335	11102	1939	2223	1866



Graph 1: Gossip Algorithms in logarithmic scale

Table 2: Push Sum Algorithm Run Time (ms)

Num Nodes	Full	Line	twoD	threeD	Honeycomb	Honeycomb_rand
10	51	540	26	208	336	134
50	459	40573	68	718	4112	717
100	1502	215580	142	1866	19977	1844
200	5043	1870725	38137	4551	81950	3365
300	11009	2482750	67496	8925	164965	5324
400	19193	~4721200	79313	18476	279949	7325
500	30268	~6721200	117717	16934	472153	8381
1000	127859	~7721200	218994	49388	2021840	17748
2000	552610	~8721200	392301	160230	5759052	37798



Graph 2: Push Algorithms in logarithmic scale

Interesting Findings

Algorithms

- In terms of the gossip algorithm, nodes are not affected by how many times their neighbour has heard a rumor. In push sum, however, the number of times the node's neighbor has heard the rumor affects the node directly. It affects the node if their neighbour has heard it once versus several times because the value they send to the node gets added it's own s and w which are directly related to the number heard. This means push sum has a more causal relationship between the nodes than gossip does.
- Push sum converged to a sum more accurately as we increase the number of nodes which is interesting as it reveals a direct relationship of number of nodes and number of rounds in which rumor/message is sent.

Topologies

- Between topologies we found that the method of assigning neighbors either systematically or random makes a large difference. For example, when 2D begins it's x and y coordinates are assigned randomly and it's neighbors are determined based on their difference. In the 2D grid where neighbors are assigned randomly the supervisor has to play a larger role in monitoring the nodes while in the 3D the nodes are assigned systematically and they can work among themselves. The supervisor has to be aware that 2D may have nodes that start with no neighbors. This is not true in topologies such as 3D or line where the nodes are systematically assigned and guaranteed to have a neighbor. The supervisor in the latter case is used more for cleaning at the end.
- Another interesting finding was that full is not the fastest topology. Because in full nodes have the most neighbors one could assume that the node would terminate faster than when it has less neighbors but as we ran experiments we saw this not to be the case. This makes sense though because even though the node has more neighbors who can pick them, those nodes are also picking randomly among a larger pool of neighbors so the probability of getting picked decreases.
- Moreover Full topology is the only topology where the probability of everyone getting the rumor is very high . Most of the nodes hear the rumor at least once.
- Honeycomb topology - Its the only algorithm that keeps updating the neighbors list as it adds the nodes. Because of this nature of the algorithm , the Genserver has to do more work as every time one Genserver process adds someone as their neighbor, they have to add themselves as their neighbor too
- We must supervise our nodes and terminate the entire program under two conditions. The first is the obvious which is that all the nodes have heard the rumor requested number of times or ,in the case of push sum , the difference in ratios is less than 10^{-10}. The second termination is when a certain percent of our nodes have already terminated. We need to implement this because in the case of certain topologies such as twoD some

nodes may be left without any neighbors and still not have reached their termination clause. In all terminologies but twoD we set the termination level, or the percent of nodes that must finish before the program terminates to 90%. This level gave us the best output with most of the nodes getting the rumor/message.

- In twoD we implemented a dynamic percentage which changes every time we run the program . This is determined based off the nodes that initially have no neighbors (which we therefore know will never hear the rumor and hence will eventually die by themselves) and the nodes that are connected to one (which we know will therefore most likely hear the rumor). The combination of these two are subtracted from the total number of nodes. Once we reach that amount of nodes that have terminated our project also terminates. This percentage works well for most of the runs.

Largest Number for Gossip

Full	Line	twoD	threeD	Honeycomb	Honeycomb_rand
2500	5000	5000	5000	5000	7000

Largest Number for Push Sum

Full	Line	twoD	threeD	Honeycomb	Honeycomb_rand
3000	300	3000	5000	2000	7000