COP5612 Fall 2018

Project 2 Gossip Simulator

October 1, 2018

Group members

Prahlad Misra [UFID: 00489999]

Saranya Vatti [UFID: 29842706]

Implementation for Gossip and Push-Sum

In order to simulate the Gossip spread, we used multiple GenServers and a Supervisor. A Supervisor process is first created and it parses the inputs given through the command line which include the number of nodes. According to the inputs, the supervisor created the number of nodes specified. The kind of node again depends on the algorithm that is taken as input.

In case of topologies like 3D and Sphere, the number of nodes are rounded up to the closest cube and square respectively. The Supervisor picks a random process and propagates the message (in case of Gossip) or starts the push sum algorithm for sum computation. Each node then transmits the message to other node (picked by topology), and also calls itself to ensure continuous propagation.

The state of the GenServer worker holds the information it requires. In Gossip, it holds the number of times it has heard the message, while in Push-Sum, it holds the sum, weight and the ratio. Once it crosses a threshold of max_count, it stops transmitting and sends a message to the Supervisor to increment the count of nodes completely heard. The Supervisor keeps listening for the nodes who have heard the message.

We assume convergence once 90% or more coverage is done. The Supervisor dies and the time taken is measure from the creation of nodes to the ending of the Supervisor. We ran the process for for 90% and 100% convergence in case of Gossip. Once all the nodes have heard the message some max number of times, the process is ended. For Push-Sum algorithm, the convergence is achieved when the ratio of (s/w) does not change by more than 10^-9 in three rounds of calling the same process. Once that happens for all processes, we print the ratio achieved and exit.

Observations

We plotted the number of nodes run with both the algorithms and all the topologies. With Gossip, we see the high variance when directly plotted verses the time taken by the process. Time is calculated by the System.monotonic_time provided by Erlang. The linear graph is in Figure 1. The logarithmic graph is in Figure 2.

1. We notice that the **line** takes the maximum time. It takes a little lesser time in 90% convergence but a lot higher in 100% convergence. This is perhaps because the end nodes have lesser probability of some node reaching them to deliver the message.
2. In a similar observation**, fully connected takes the least time**. If a small network, this can be preferred. However, the overhead is maintaining a network of all nodes that is accessible from any node.
3. Interesting observation is that the performance can be improved significantly just by adding an additional node in the **imperfect line**. This perhaps can be preferred with a balance between overhead and time taken.

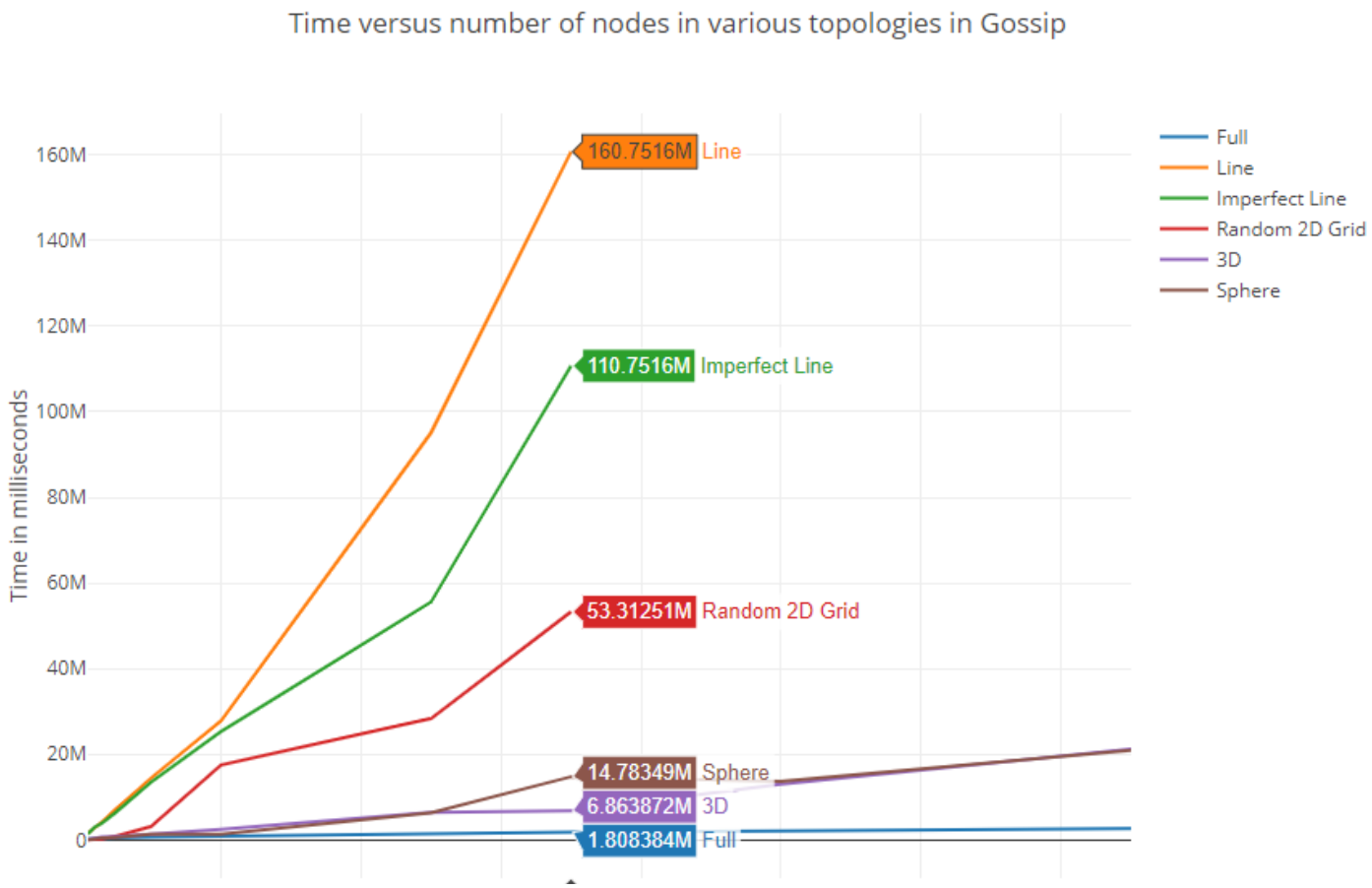

Figure 1 Nodes versus Time for Gossip

4. Since the Sphere and 3D Grids are almost fully connected for a small number of nodes, we can see that the times taken by them is similar as well, as expected.
5. The plotted graphs show the times each topology has taken for the Gossip algorithm to reach 100% convergence with max count as 50. That means, each process has to hear the message 50 times for it to stop propagating.
6. While it is true that maximum connection will intuitively suggest minimum time, taking into consideration the amount of extra work done by the processes which are not receiving anymore, we can see that using either Random 2D or an extra random node in Imperfect Line are most profitable.

Similar observations can be made in convergence of Push Sum as well.

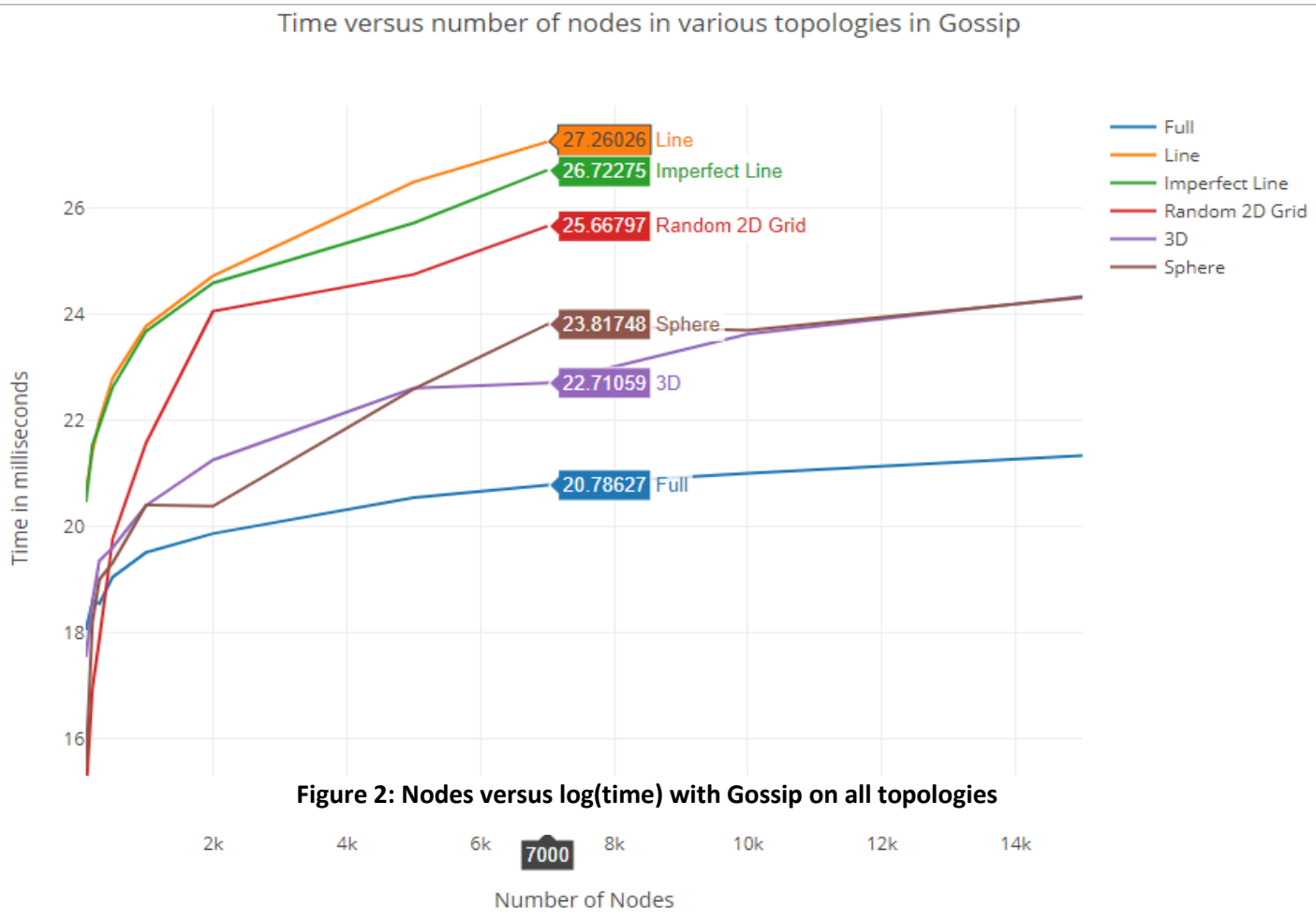

Figure 2: Nodes versus log(time) with Gossip on all topologies

Table showing the logarithmic times of different topologies with Gossip

Time in microseconds -> / Number of nodes	Full	Line	3D	Imperfect 2D	Sphere	Random 2D Grid
100	18.05528	20.666224	20.46999151	17.54689446	15.52356	14.95419631
200	18.61102	21.425216	21.54898743	18.6110248	18.20945	16.96578428
300	18.55075	21.993646	21.9099048	19.357552	19.00843	17.87651695
500	19.05257	22.80191	22.63300939	19.60917874	19.32193	19.77313921
1000	19.51964	23.785963	23.68905955	20.40832974	20.40939	21.58824615
2000	19.87344	24.728133	24.59608782	21.26033152	20.39124	24.06507949
5000	20.55075	26.502521	25.72858487	22.61332905	22.59502	24.75853554
7000	20.78627	27.260258	26.72275251	22.71059122	23.81748	25.66797083
10000	21.01053			23.63469763	23.70563	
15000	21.34096			24.34186625	24.32115	

Graph showing the nodes versus topologies for Push Sum

Additional technical details are in the ReadMe.