Gossip and Push Sum Algorithm Implementation using Erlang

Readme File

Team Members

Lohit Bhambri (GatorID: lohit.bhambri)

Hussain Imthiaz Hussain (GatorID: imthiazh.hussain)

Build Process

- Download the repository git clone https://github.com/lohitb009/Gossip-and-Push-Sum-Simulator
- Open cmd in src folder of project. Compile the files using: 'c(gossip).' 'c(pushSum).' 'c(supervisorMod).'
- Run using 'supervisorMod:startSupervisor(TotalNodes, Topology, Algorithm).' Where
    - TotalNodes is a number of nodes in network
    - Algorithm can take values "Gossip" and "PushSum" as strings
    - Topology can take values "Line", "2D", "Imperfect3D" and "FullNetwork" as strings

What is Working

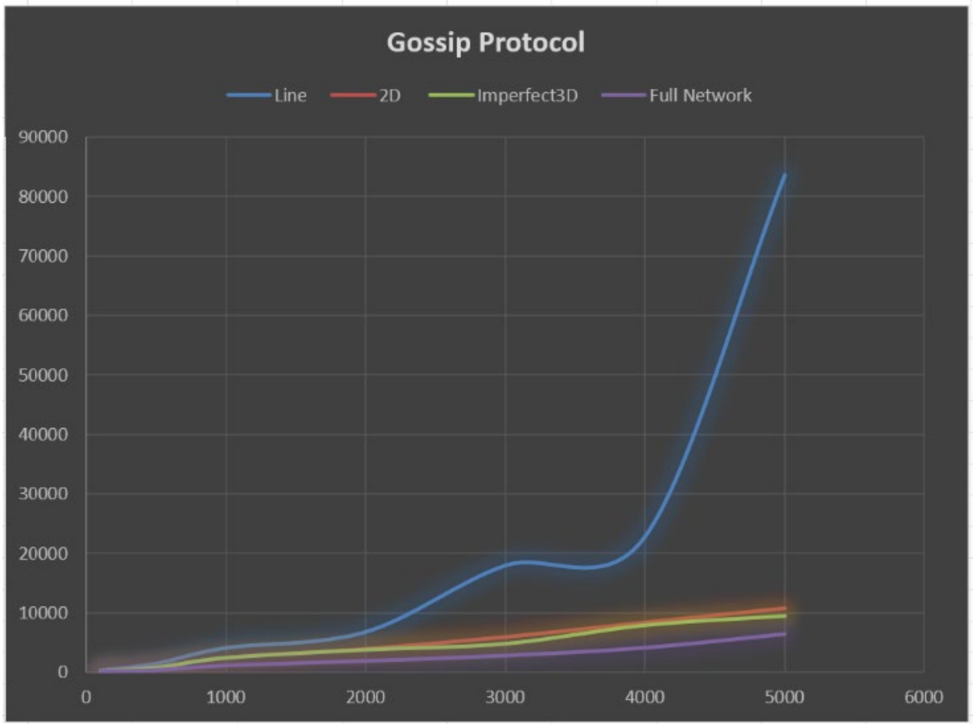
All combinations of algorithms (gossip, pushsum) and topologies (line, full, 2D, and Imp3D) were implemented. Convergence was achieved based on the conditions set in the problem statement. i.e. If a node participating in gossip received 10 messages, it converges. Similarly, in case of pushsum, we converge a node based based on change in ratio of previous and new sum and weight. (If the change is less than delta (here 10^-10) in 3 consecutive rounds.

Additionally, as an extra feature, we have implemented a message acknowledgement protocol. It helps a receiving node send back an acknowledgement to the sender node. This enabled the run-time analysis of node availability and was extended to analyze node failure (explained in bonus.zip).

Gossip

Normal Scale

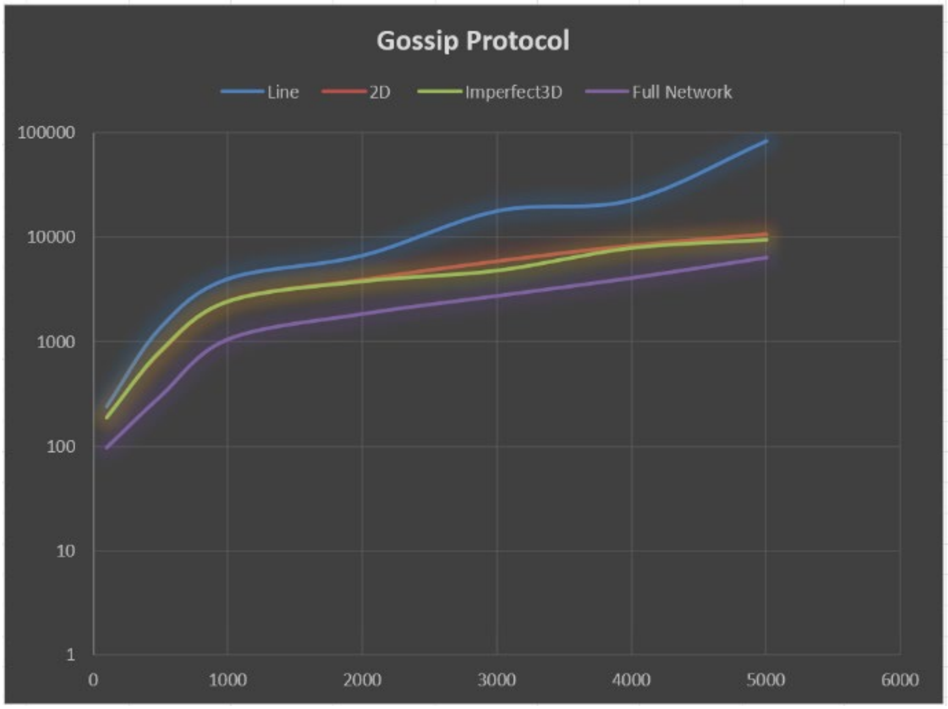
Note: Time calculation was done using 'statistics(wall_clock)'. The total (wall clock time at final iteration of convergence) – (wall clock time at user input stage) was used to calculate the running time of execution. The total wall clock time is printed every iteration of execution.



Y Axis – Convergence Time (in milliseconds)

X Axis – Number of Nodes in Network

Logarithmic Scale



Y Axis – Convergence Time (in milliseconds)

X Axis – Number of Nodes in Network

Conclusion:

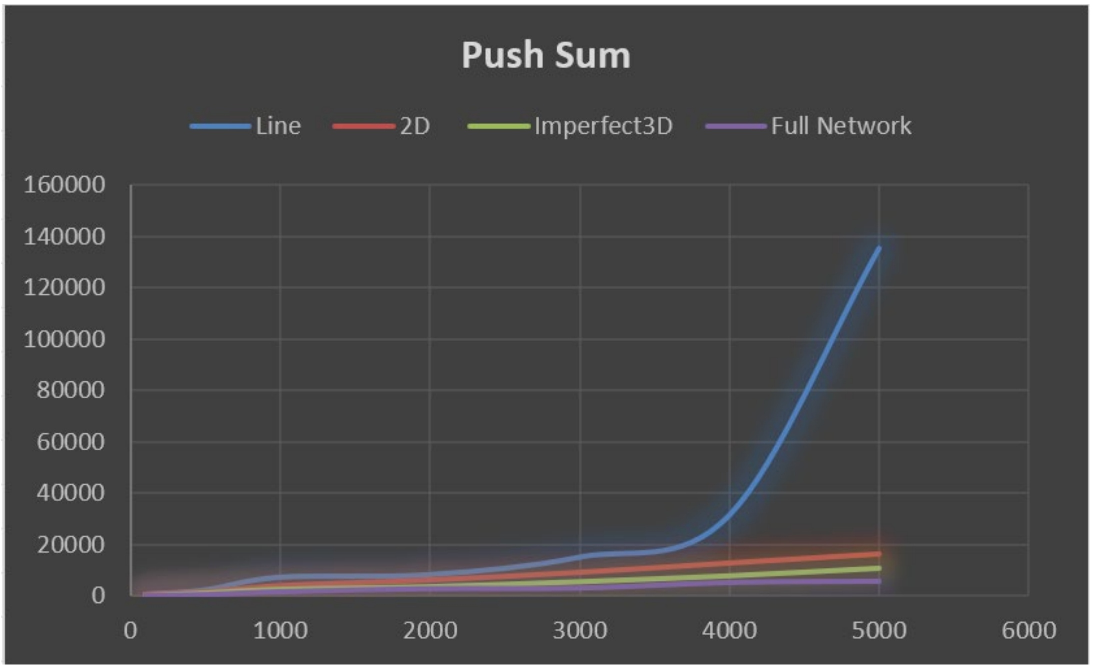
Convergence Time ∝ $\dfrac{1}{\text{Cardinality of Neighbours}}$

- Line topology took the longest time to finish execution. (2 neighbours)
- Full topology took the shortest time to finish execution at almost all values of n.
- 2D and imperfect 3D converged with an execution time more than full but lesser than line topology.
- **Additional Mechanism Implemented:** In many executions, all the neighbours of a node had converged but other nodes were still alive. In this case, transmission was not possible. Hence the node sends a message to the supervisor/main process which in turn sends a message to a random alive node, to continue the p2p communication.

PushSum

This algorithm executes by passing the values(s, w) as a message to another node. During the spawning of nodes, the value of s is the 'index' of the node. i.e. [1,2,3,4,5]. The weight is kept same for all nodes w = 1. We converge a node based based on change in ratio of previous and new sum and weight. (If the change is less than delta (here 10^-10) in 3 consecutive rounds.

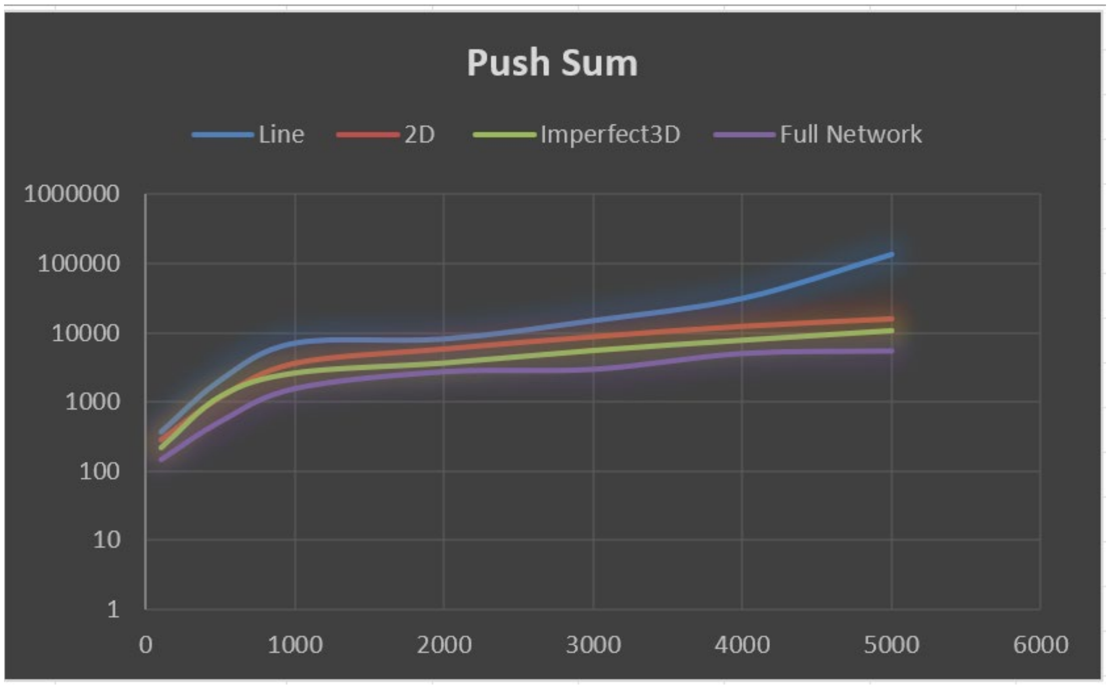
Normal Scale



Y Axis – Convergence Time (in milliseconds)

X Axis – Number of Nodes in Network

Logarithmic Scale



Y Axis – Convergence Time (in milliseconds)

X Axis – Number of Nodes in Network

What is the largest network you managed to deal with for each type of topology and algorithm

	Gossip	Pushsum
Line	10,000 nodes	10,000 nodes
2D	20,000 nodes	10,000 nodes
Imperfect 3D	20,000 nodes	10,000 nodes
Full Network	20,000 nodes	10,000 nodes

An important observation:

In all algorithms, the rate (Speed) of convergence of nodes decreases as the algorithm executes from its initial 'all-nodes-alive' state to the final few convergences. This happens because as and when nodes converge, they form an envelop around alive nodes. These alive nodes are unable to transmit to any neighbours (since they converged) and ask the supervisor/main_process for help. The supervisor in turn sends a fresh message to a random alive node to continue the algorithm. All this overhead causes decrease in rate of convergence towards the final few iterations of the algorithm.