PROJECT-2 REPORT

Project partners:

1. Wins Goyal, UFiD – 73571559, *winsgoyal@ufl.edu*
2. Rahul Wahi, UFiD – 30536162, *rahul.wahi@ufl.edu*

Implementing Gossip and Push-sum Algorithm for various topologies:

Model Architecture:

A Supervisor starts the Genserver which will start Multiple Actor nodes and a main worker actor. This main worker will track the information of all those actors which are active and have the message. These active actors send acknowledgment with their state to this main worker, which will then report back to the Supervisor.

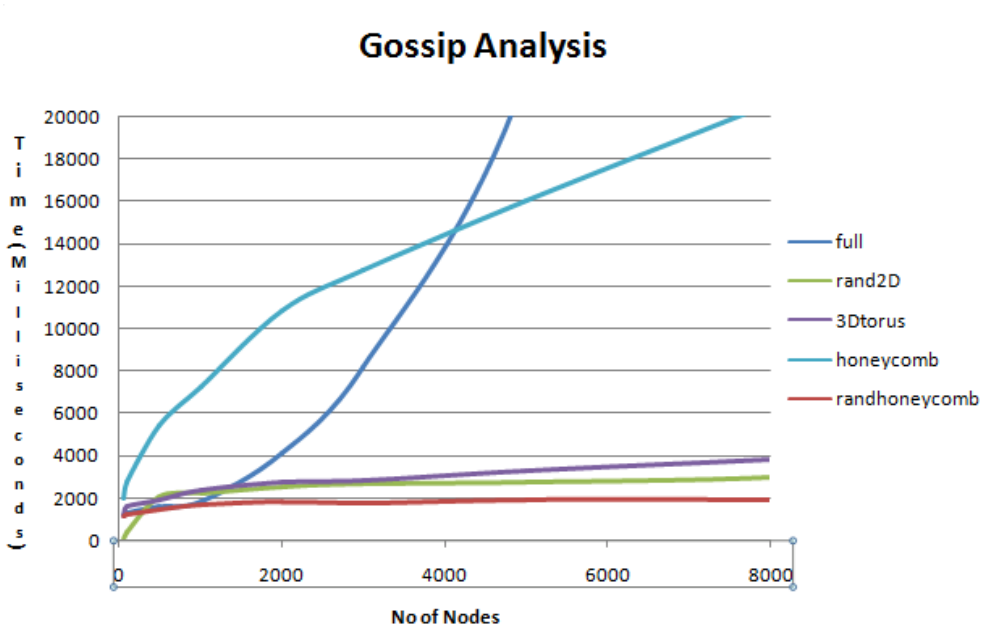
Computation of Convergence Time:

To compute the convergence time, the main worker is maintaining the list of all active actors with the message at a point, and monitors a microlithic timestamp. As the actors start dying, they send this information to the main worker who will then delete that actor from the list. When the list gets empty after the last active actor has died.

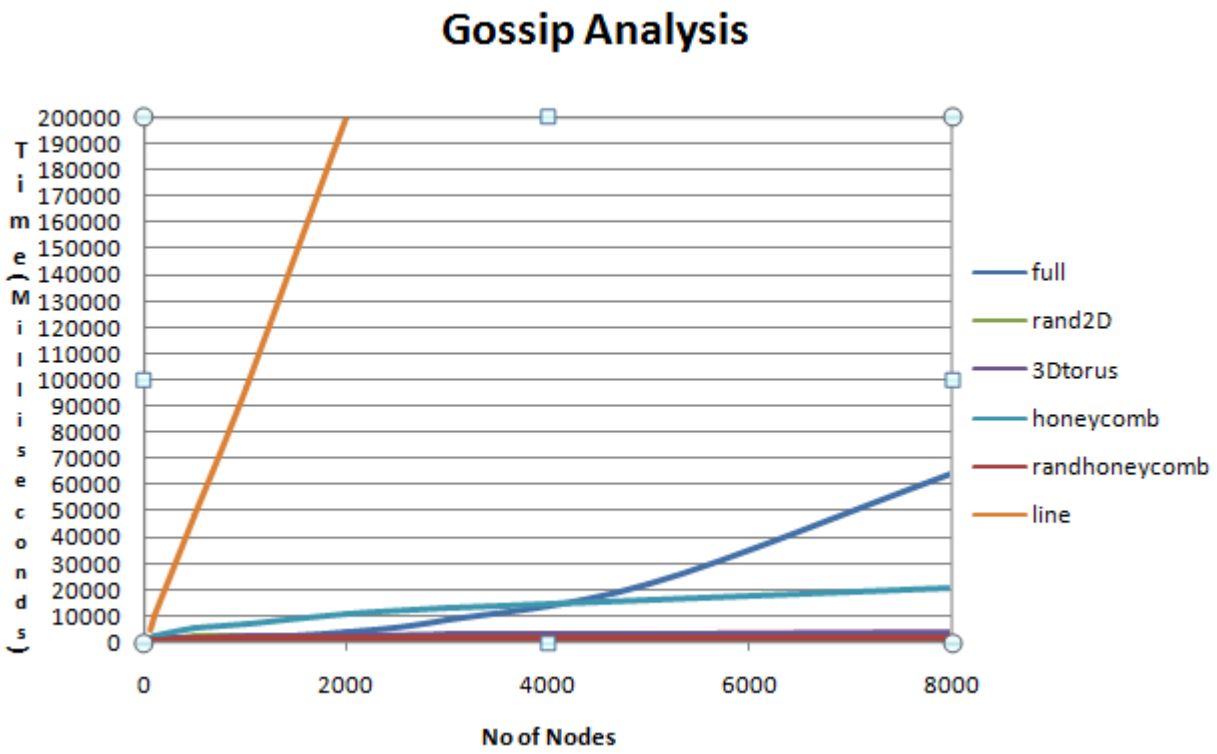
The convergence time will mostly depend on the criteria we take to stop the algorithm and how we send the message. Like, upon receiving the message, the actor will periodically be casting that message to a random neighbor from its neighbors list, and when this actor has heard the message around say 10 times, this actor exits or dies, acknowledging the main worker. This is for Gossip. While, in Push-sum the high criteria is to check if the S/W ratio changes beyond 10^(-10) for the last three times that the actor heard the message.

Different Plots representing the dependency of Convergence Time as a function of the size of the Network (i.e. no. of Nodes in the topologies):

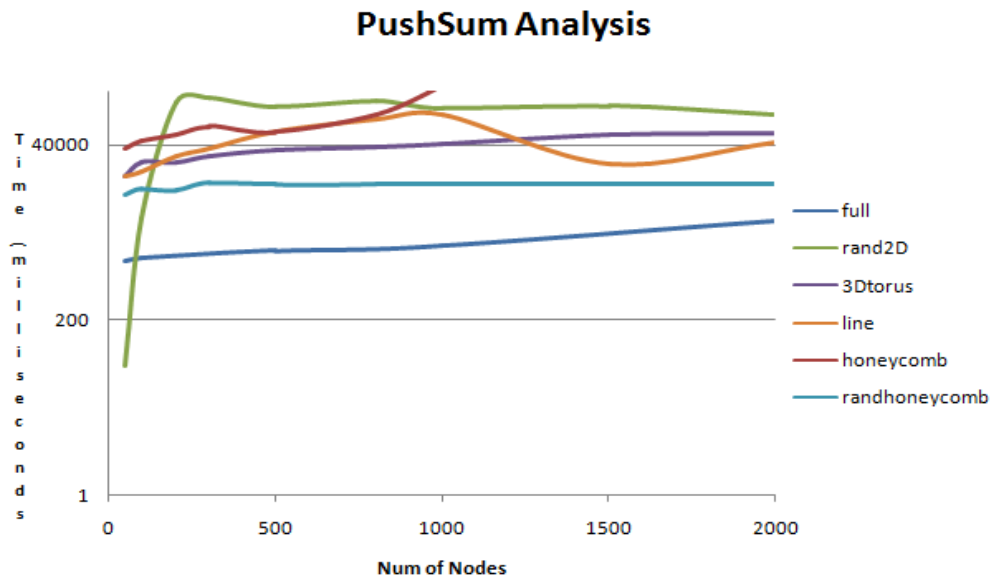
Gossip Algorithm



The above graph plots the Converge Time (in milliseconds) on y-axis and No. of Nodes in the network (on x-axis). To give a more meaning view, the Line network is not taken.
While, the below graph on same axis and scales, includes the Line network.



Push-Sum Analysis:



Observations from different experiments for both Algorithms and the topologies:

- For Gossip Algorithm:
  1. *Honeycomb* beats (performs better than) the *Full* network, if we increase the network size more than around 4000-5000 nodes.
  2. For small networks, say nodes < 1000, *Honeycomb* has the worse convergence time than other topologies, after the *Line* topology.
  3. Of the *Honeycomb* based topologies, the *random* topologies performs significantly better than the *simple honeycomb* topology. By simple, we mean the network in which no node has any random neighbor node besides its fixed at most 3 neighbors.
  4. *Line* topology is the worst for this Algorithm.
  5. *Random2D, 3D-torus* and *Random-honeycomb* perform similarly best for this algorithm. These have the least convergence time when increasing the network size even beyond 10000 nodes, as compared to other topologies.
  6. *Full* network has the worst performance after *Line*, as the number of nodes increase. But, the reasons for both having bad performance could be totally different.
- For PushSum Algorithm:
  1. Push-Sum generally has slow convergence rate than the Gossip algorithm for various topologies that we have experimented on.
  2. *Honeycomb* has the worst convergence time for larger networks, while *random Honeycomb* has the best for the same algorithm.
  3. *3Dtorus* performs better than *random2D* for Push-Sum while for Gossip, it is the opposite.
  4. *Full* network is also worse than *Line* network in this case which is not the case in Gossip.

Important Inferences:

We can make different interesting inferences based on our observations from the plots and experiments done on our Gossip-PushSum simulator.

1. Why *Random Honeycomb* seems to perform better?
   Because every node is also connected to a random node in the topology, the casting spreads more randomly from different locations while the topologies with least random neighbors would perform worse because the message may not spread from different random locations but along a particular direction at a time.
2. For Gossip Algorithm, *Honeycomb* plot seems to follow a logarithmic function. Also, the *Line* topology gives a line plot while the *Full* topology gives a polynomial plot.
3. *Random2D* tends towards *Full* topology when the network size is significantly increased.
4. What else we could do?
   (a) We can try starting the simulator from different starting points in the topologies and check whether the convergence is affected by this. Like, we can try transmitting the message from the middle of the *Line* or from the center of the *Honeycomb*.
   (b) We can implement the multi-casting model in the simulator, i.e. each nodes sends the message to multiple neighboring nodes at a time, and then the convergence time dependency in that implementation. However, in real life scenarios, the multi-casting architecture is not generally followed.