COP5615 – Fall 2019 Project 2 – Gossip Simulator

Team Member –

1. Milind Jayan
2. Vivek k Singh

Project Implementation Detail

The objective is to achieve the convergence of two algorithm Gossip and PushSum by developing the simulator based on actors written in Elixir.

Gossip Protocol

The convergence of Gossip Algorithm occurs when all the nodes in the network has heard the rumor at least once. In our implementation, the state of the node can be either infective, susceptible or removed,

- Infective – a node that has not received an update yet.
- Susceptible – a node which has heard the rumor and is willing to share
- Removed- a node that has already received an update by is not willing to share

The node moves from susceptible to infected when it receives the message for the first time. Correspondingly a node moves from infected to removed, when the node has heard the rumor at least 10 times.

- Node
  - State
    - Status – (infective, susceptible, removed)
    - Count – number of the time a node has heard the rumor.

- The criteria for a node to be removed or converged is when the node hear the rumor 10^{th} time or if all the neighbors of the node are removed.
- When the convergence is achieved the program print the convergence time and terminates.

- Send – Each node picks a random neighbor and transmits the message to that node.

- Receive- Upon receive a node checks the number of times it has received the rumour

PushSum Protocol

The convergence of a node occurs when its sum estimate (s/w ratio) hasn't change more than 10^-10 in three consecutives receive rounds. The following values are maintained in the state of the node:

- Node
  - State
    - Status – (infective, susceptible, removed)
    - Count – number of times the s/w ratio was less than 10 ^ -10
    - S – Actor number (Initially has a value i for actor number i)
    - W – Initial value is 1
    - Ratio(S/W) – Sum estimate (s and w are current values of an actor)

- Starting: One of the actors starts receiving the message from the main process
- Message: Messages are pairs of the form (s, w).
- Receive: Upon receive an actor adds received pair to its own corresponding values. Upon receive, each actor sends the message to a random neighbor.
- Send: the sending node retains half of the s and w before transmitting to a random neighbor
- Termination: If an actor ratio s/w didn't change more than 10^-10 in three consecutive rounds.

Topologies

Network topology plays a critical role in spreading of a message in any protocol. We have implemented these simulators for various topologies. The topology determines which nodes are considered a neighbor.

- Line - Actors are arranged in a line. Each actor has only 2 neighbors (one left and one right, unless you are the first or last actor).
- Full Network - Every actor is a neighbor of all other actors. That is, every actor can talk directly to any other actor.
- Random2D - Actors are randomly position at x, y coordinates on a [0- 1.0] x [0-1.0] square. Two actors are connected if they are within .1 distance to other actors.
- 3DTorus - Actors form a 3D grid. The actors can only talk to the grid neighbors. And, the actors on outer surface are connected to other actors on opposite side, such that

degree of each actor is 6. If the number of the nodes are chosen in a way where it is not possible to form a perfect 3D grid, we arranged them in a nearest possible perfect cube. In this case few actors might not be having six neighbors necessarily.

- Honeycomb - Actors are arranged in form of hexagons. Two actors are connected if they are connected to each other. Each actor has maximum degree 3
- Honeycomb random - Actors are arranged in form of hexagons (Similar to Honeycomb). The only difference is that every node has one extra connection to a random node in the entire network.

File Explanation

- Gossip
  - actors.ex - Gossip algorithm implemented
  - actorsupervisor.ex – Supervises all the actors
  - start.ex – the function that drives the process and checks convergence
- Push_Sum
  - actors.ex – Pushsum algorithm implemented
  - actorsupervisor.ex -
  - start.ex – Pushsum GenServer initialized
- Topologies
  - Topologies.ex – All the topologies are implemented
  - Topomain.ex – Topologies initialized
- application.ex – Entry point of the application

There is a main supervisor which supervises the start module and topologies and a sub supervisor which supervises the nodes. The number of nodes that the sub supervisor supervises would be the number of total nodes in the system.

Instruction for running the code

After downloading the project folder, move to the directory with the main.exs file:

```
cd project_2_final
mix run main.exs arg1 arg2 arg3
```

Argument	Value	Possible Values
arg1	Number of nodes	any positive integer
arg2	Topology	line, full, random2D, torus, honeycomb, honeycombrandom
arg3	Algorithm	gossip, pushsum

The output would display the time taken for convergence. For gossip and push sum the time taken for convergence was measured after at least 90 percent of the nodes have reached convergence for all topologies.

Interesting Observation

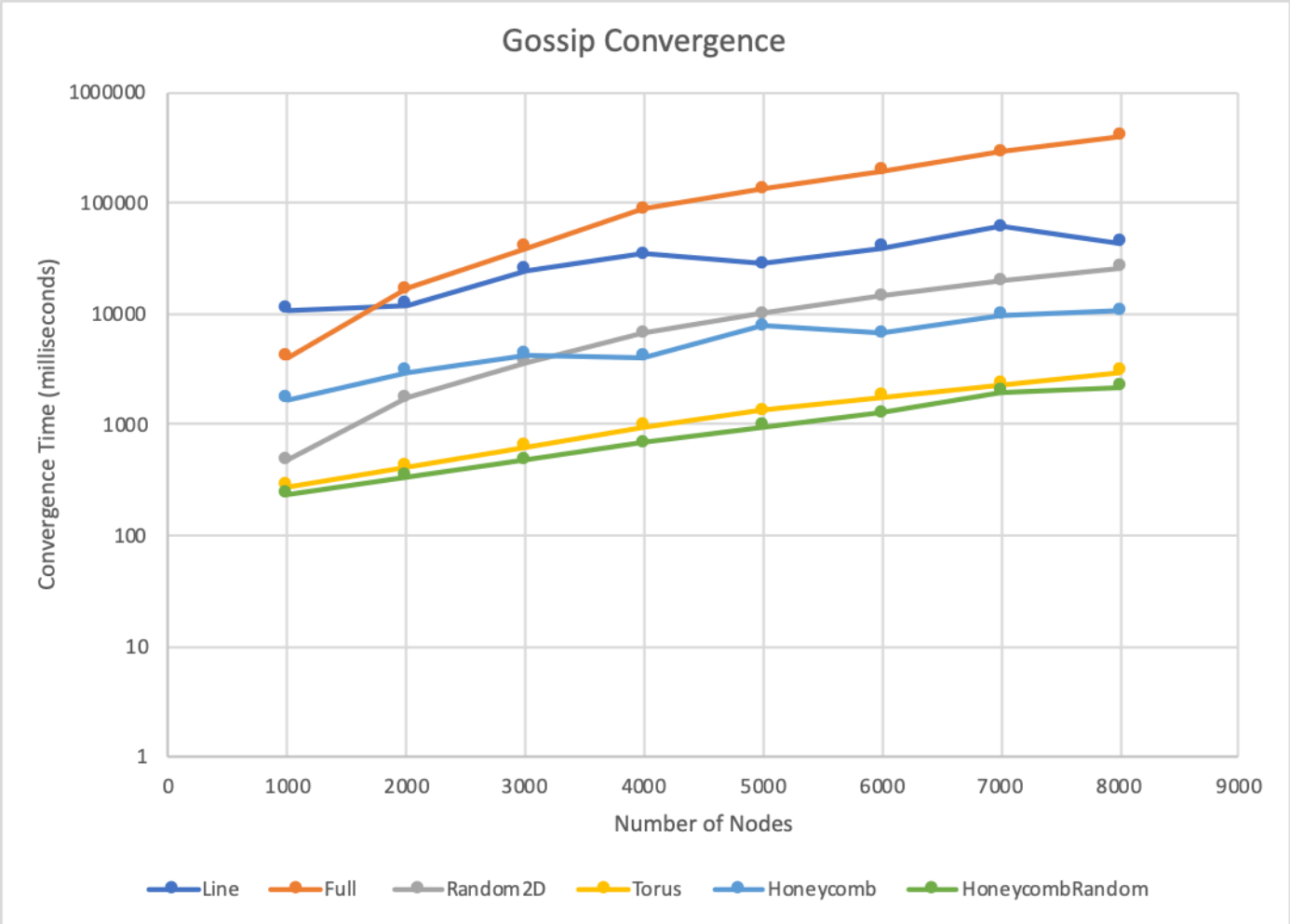
- For gossip and push sum algorithm, the efficiency of algorithm increases when a node transmits a message to a random alive neighbor.
- Torus and Honeycombrandom yield the best result on both the push sum and gossip algorithms
- Gossip algorithm converges faster when compared to push sum
- Good Topologies for gossip protocol are torus and honeycombrandom. For smaller values full network also gave good results
- Honeycomb random yields better results than honeycomb for both push sum and gossip algorithms. This is because the random neighbor being selected help spread the rumor wider in the network and hence it would converge faster

PTO

Result for Gossip

Convergence time for varying number of nodes for different topologies (log time vs number of nodes plotted)

#Nodes	Line	Full	Random2D	Torus	Honeycomb	HoneycombRandom
1000	10884	4095	479	274	1686	239
2000	12037	16974	1718	424	2993	345
3000	25058	39855	3695	632	4242	476
4000	34202	87112	6585	949	4045	685
5000	27908	133980	10001	1324	7831	962
6000	39523	194051	14242	1783	6683	1288
7000	60738	285163	19606	2328	9879	1973
8000	43384	401321	26212	2981	10502	2194



Result for Pushsum

Convergence time for varying number of nodes for different topologies(log time vs number of nodes plotted)

#Nodes	Line	Full	Random2D	Torus	Honeycomb	HoneycombRandom
500	48703	6153	2683	47861	325929	39737
600	50094	9604	2770	69590	389689	39270
700	56953	14272	2680	54256	268606	39504
800	86175	18331	2689	74499	272330	39660
900	139229	28305	2696	87411	229391	39405
1000	91995	33761	2778	52774	321125	39618