DOS Project 2

Kshitij Ranganath

Brief Description:

This problem is aimed at implementing the Gossip and Push-Sum algorithms for information propagation. These algorithms are in turn implemented on various topologies like full,3D Grid, Random 2D Grid, Sphere, line and imperfect line.

- The Gossip protocol works by initiating the process from a single actor which forwards the message to the other actors. Once infected with a rumour a node keeps sending gossip messages to one of it's randomly selected neighbor from the neighbor list. A node dies typically when it has received the copy of gossip enough number of times(for our case its 10).The point of convergence is reached when the number of dead actors gets stabilized.
- The Push-Sum algorithm works by sending messages in the form of pairs(s,w) where s is the value of the actor number and w = 1 for each actor. The propagation converges when the s/w ratio doesn't change when compared to a pre-defined value (10^-10 in our case) for three consecutive times.

Network topologies Used:

- A full network is a topology in which every node is connected to every other node and a message is sent from one actor to any other actor in a random fashion.
- 3D Grid network is a topology in which an actor sends messages to its immediate neighbour be it Up, Down , Left, Right , Front and Back if they exist. The neighbour is selected at random from the neighbour list.

- Rand2D is decided by first randomly allocating a node on a 2D plane between square given by [0,0],[0,1],[1,0],[1,1]. And the neighbours are decided by the distance between the assigned values in plane.(for our case its <=0.1)

- Torus/Sphere consists of curved geometry with each node having 4 neighbours, 2 in the plane of cross-section and 2 in the plane of the circle viewed from top.
- Imperfect 2D behaves like a Line network except that it sends an extra message to a random neighbour in the network.
- Line topology involves sending messages back and forth to an actor's front and back neighbours.

Running the program:

GOSSIP and PUSHSUM

- enter the number of nodes
- enter the topology
- enter the algorithm
- set the number of nodes to be killed as 0

BONUS

- enter the number of nodes
- enter the topology
- enter the algorithm
- enter the number of nodes to killed

Output Calculation:

The time taken to converge is calculated by the difference in system times when the process is initiated until the number of dead actors stabilizes. This is in the order of milliseconds. All the network geometries/topologies are working.

Execution Steps:

The time taken to converge is calculated by the difference in system times when the process is initiated until the number of dead actors stabilizes. This is in the order of milliseconds.

- Extract Zip File
- Run iex -S mix inside the file directory
- Proj.main(100,”pushsum”,”Line”,0) => Proj.main(max_no_of_nodes,”Algorithm”,”Topology”,no_of_nodes_tobe_killed)
- [max_num , count_of_dead_processes , neighbour_meta_list, start_time , finish_time] = Actor.get_state(:id0)
- finish_time – start_time
- The last step gives us the time taken for the algorithm.

Algorithm: Gossip

Line 2D : MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	141
10	235
25	265
50	813
100	812
500	2969
1000	2844

FULLY CONNECTED: MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	313
10	422
25	797
50	1538
100	2219
500	3297
1000	6281

Imperfect Line 2D : MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	172
10	313
25	515
50	468
100	593
500	735
1000	766

3D GRID : MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
8	203
27	312
64	500
125	516
1000	782

Toroid : MAX_SIZE : 1024

Number of Nodes	Time to converge in mS
4	188
9	296
25	453
49	360
100	531
1024	703

Random 2D : MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	140
50	141
100	219
500	1532
1000	2297

Algorithm: Pushsum

Line 2D : MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	16
10	15
25	78
50	344
100	703
500	17891
1000	21375

FULLY CONNECTED: MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	0
10	0
25	32
50	62
100	266
500	2094
1000	5391

Imperfect Line 2D : MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	0
10	0
25	16
50	31
100	62
500	907
1000	1504

3D GRID : MAX_SIZE : 1000

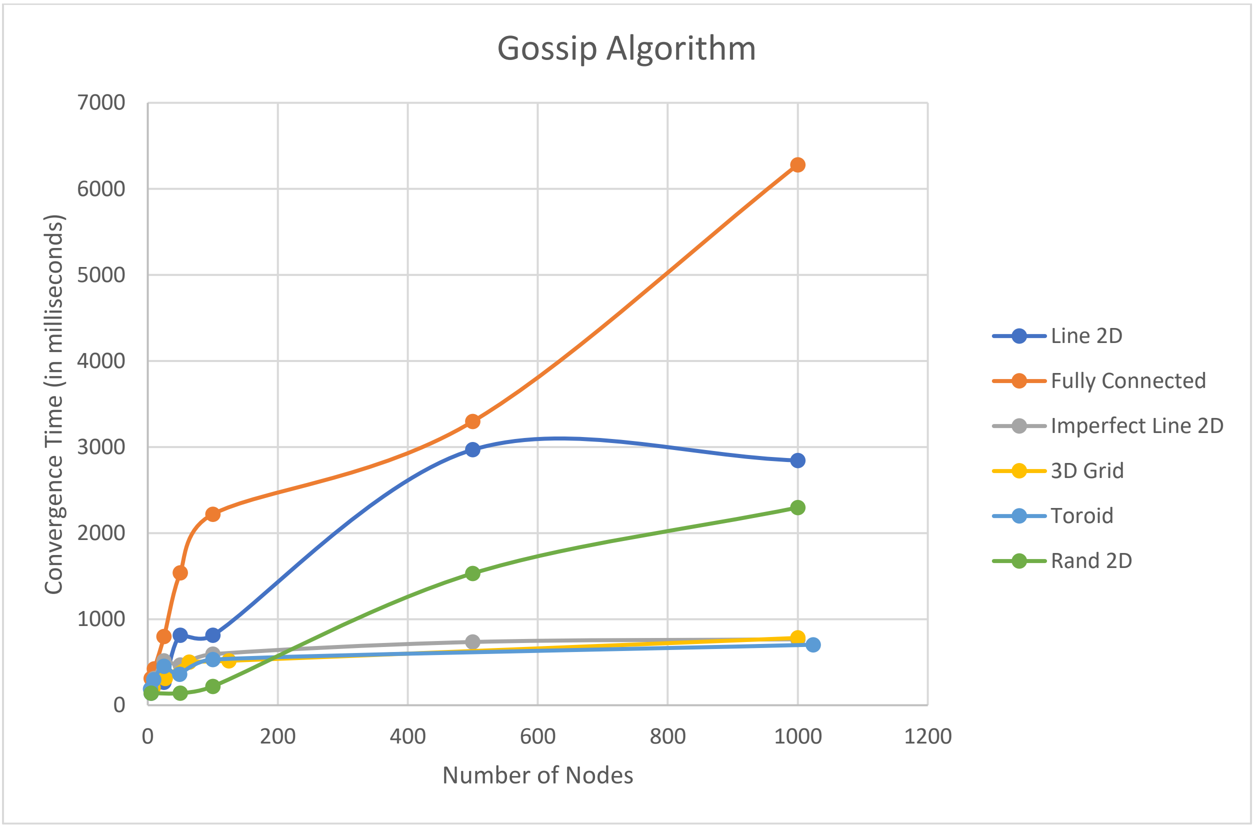
Number of Nodes	Time to converge in mS
8	0
27	16
64	47
125	125
1000	3641

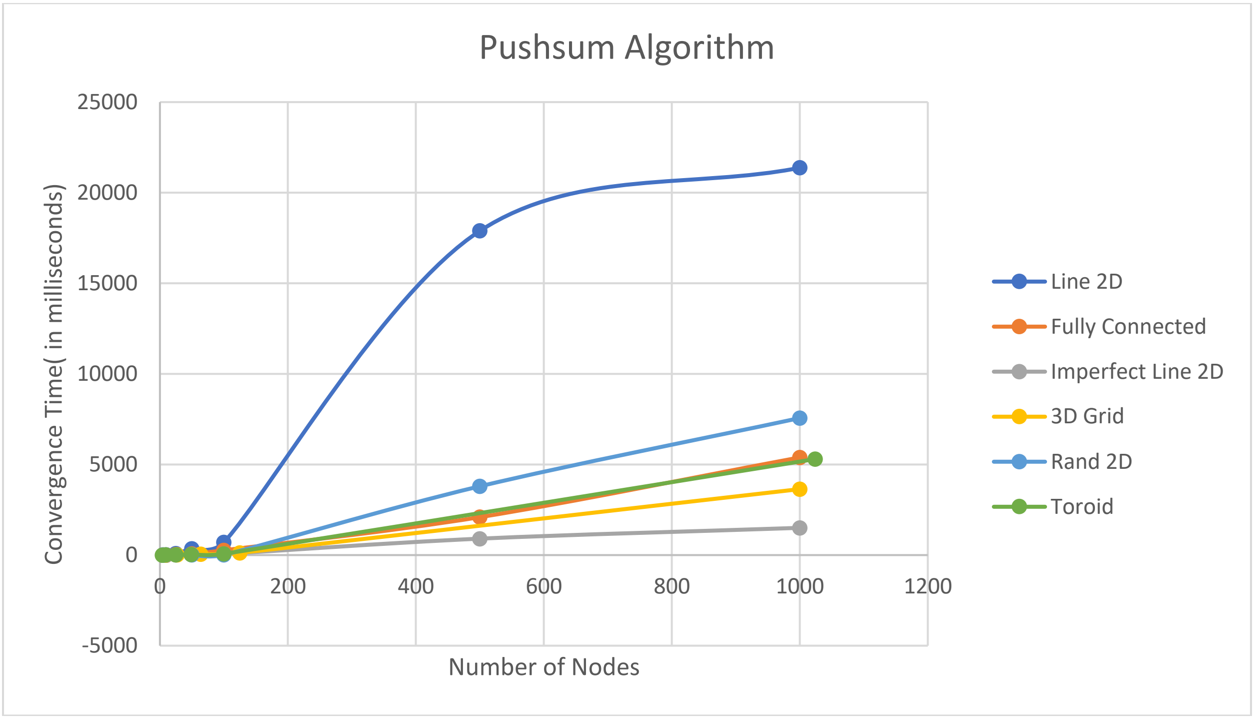
Toroid : MAX_SIZE : 1024

Number of Nodes	Time to converge in mS
9	0
16	15
25	16
49	32
100	71
1024	5297

Random 2D : MAX_SIZE : 1000

Number of Nodes	Time to converge in mS
5	0
50	15
100	16
500	3797
1000	7563





Result:

In Gossip Algorithm Full Network has the highest time, as it delivers the most gossip/rumour as its connected to every other node. It keeps sending gossip even when a lot of its neighbours die as its still connected or in a connected graph.

Whereas Line, Grid3D, etc. are surrounded by far fewer neighbours therefore when gossip is initiated a lot of these neighbours die leading to island phenomenon in the graph/network topology from which nodes cannot send or receive gossip and yet have heard the gossip less than 10 times.

In Pushsum algorithm, since a message is transmitted by a node only when it receives it and the ratio of s/w hasn't converged. In this schema, nodes with more neighbours are better at spreading their values to the network. Even though the a neighbour is randomly selected from the list of neighbours, the likelihood of faster dissemination of change through multiple possible neighbours decreases the convergence time.

Also having a directionality in spreading helps, Ideally a breadth first propagation would lead to the fastest convergence but due the randomness of selecting a neighbour for higher dimensional networks or networks with a large degree we often see the randomness creates problems of revisiting an already visited node, therefore Imperfect Line 2D is compromise between directionality and low degree and therefore achieves good pushsum convergence times.

BONUS PART

Robustness of Gossip

I implemented a failure model in which I randomly deleted a certain number of nodes (taken as input from user) and saw how much time the remaining number of nodes took to converge. The network has 100 nodes initially.

Gossip Protocol Failure Model Convergence Times

Line 2D

%Nodes Failed	Time to converge in mS
5	296
10	375
20	156
25	203
50	187

FULLY CONNECTED:

%Nodes Failed	Time to converge in mS
5	1250
10	1735
20	2531
25	1484
50	3078

Imperfect Line 2D :

%Nodes Failed	Time to converge in mS

5	625
10	672
20	578
25	641
50	860

3D GRID :

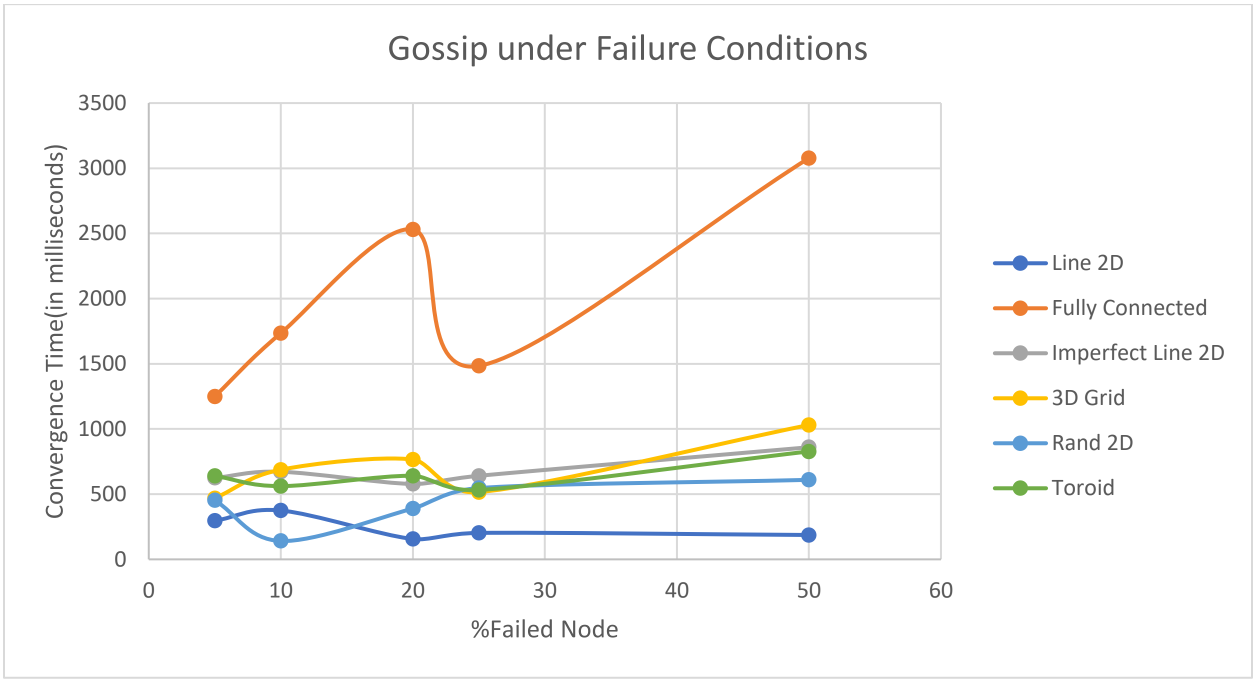
%Nodes Failed	Time to converge in mS
5	469
10	688
20	766
25	516
50	1031

Toroid :

%Nodes Failed	Time to converge in mS
5	640
10	562
20	641
25	532
50	828

Random 2D :

%Nodes Failed	Time to converge in mS
5	453
10	141
20	390
25	547
50	610



Robustness of Pushsum

I implemented a failure model in which I randomly deleted a certain number of nodes (taken as input from user) and saw how much time the remaining number of nodes took to converge. The network has 100 nodes initially.

Pushsum Protocol Failure Model Convergence Times

Line 2D

%Nodes Failed	Time to converge in mS
5	0*
10	0*
20	0*
25	0*
50	0*

FULLY CONNECTED:

%Nodes Failed	Time to converge in mS
5	16
10	0*
20	0*
25	0*
50	0*

Imperfect Line 2D :

%Nodes Failed	Time to converge in mS
5	0*
10	0*
20	0*
25	0*
50	0*

3D GRID :

%Nodes Failed	Time to converge in mS
5	0*
10	0*
20	0*
25	0*
50	0*

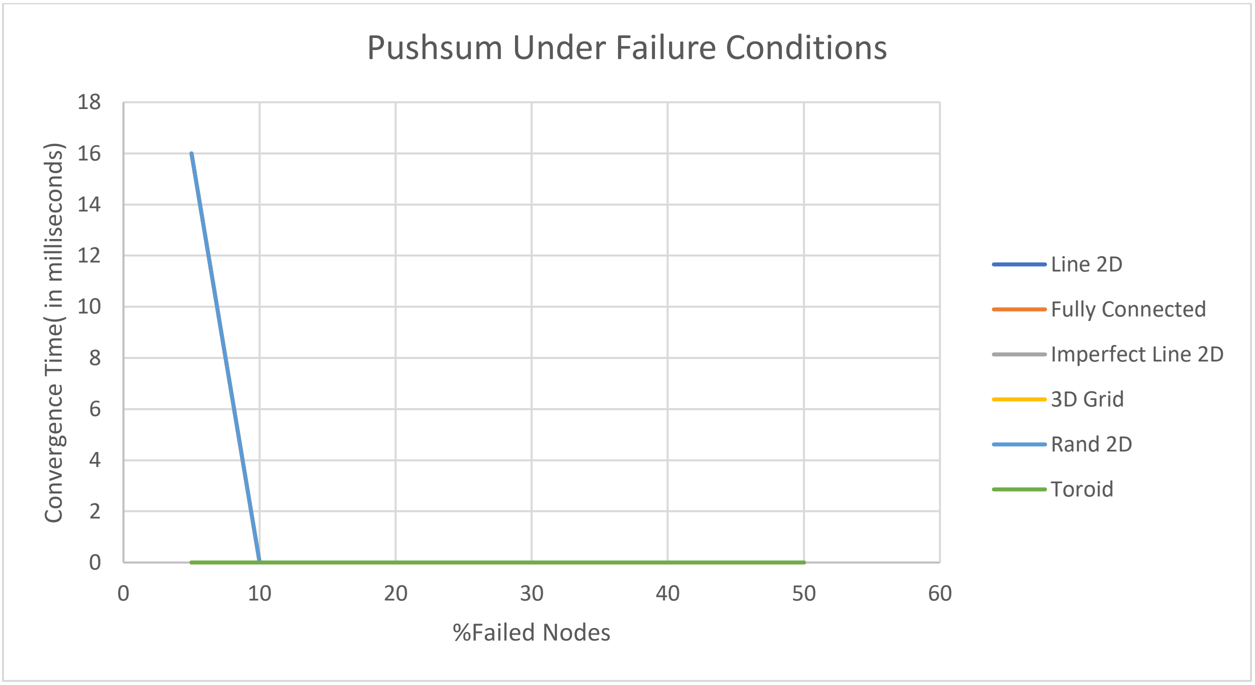
Toroid :

%Nodes Failed	Time to converge in mS
5	0*

10	0*
20	0*
25	0*
50	0*

Random 2D :

%Nodes Failed	Time to converge in mS
5	16
10	0*
20	0*
25	0*
50	0*



Results for Bonus:

Our implementation of the pushsum algorithm is not very fault tolerant, as a node sends message to randomly selected neighbour when it receives a pushsum message, but if for some reason this node is dead, then the push sum message is lost from the network and the program terminates immediately.

If we have each node fire a push sum every fixed interval of time irrespective of receiving a message this problem can be resolved.

Only certain kind of network topologies are even remotely capable of handling this as we see that Full and Rand 2D able to manage with 5-10% node failure for the pushsum algorithm. But every other topology fails as it leads to a very disconnected graph and message can be lost if the neighbour for whom it was intended is suddenly killed.

For gossip however, the island formation inside graphs or regions disconnected from the rest of the graph increases tremendously when there is a failure happening. Most network topologies become disconnected graphs at 20% Node failure.

Randomization especially for Rand2D has its own problems of having the possibility of disconnected regions of graphs, with distances > 0.1 between nodes even without failures. Fully connected again takes the most time as it goes with most iterations of gossip and still has reasonably connected network to work with for low node failure percentages. It terms of fault tolerance full networks emerge a clear winners followed by randomized 2D and imperfect line 2D as they don’t depend on a predefined geometric structure and can still be connected even when multiple randomly selected nodes fail.