DOS Project2

Nitin Kosuri (UFID: 29694624)
Venkatesh Punuru (UFID: 18957381)

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, 2D, imperfect 2D and line.

- The Gossip protocol works by initiating the process from a single actor which forwards the message to the other actors. The point of convergence reached is when each of the actors listens to the message 10 times.
- 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 ration doesn't change when compared to a particular predefined value.(In our case 10^-10) for three consecutive times.
-

A glimpse of the network topologies:

- A full network is a topology in which a message is sent from one actor to any other actor in a random fashion.
- 2D network is a topology in which an actor send messages to its immediate matrix neighbours.
- Imperfect 2D behaves similar to a 2D network except that it sends an extra message to a random neighbor in the network.
- Line topology involves sending messages back and forth to an actors front and back neighbours.

Results Expected:

From the description of the Network topologies whether it be Gossip or Push Sum the order in which the time taken to converge from less to higher time is given by full < imperfect 2D < 2D < line.

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

Implementation Details:

Project2.Scala:

This scala file is the entry point and hosts the main method. Our implementation has 3 parameters in the order of NumNodes, Topology, Algorithm

- *NumNodes*: number of actors in the network
- *Topology*: top stands for the name of the topology ex: line full etc
- *Algorithm*: stands for the algorithm used – Gossip or Push-sum

Master.scala :

This file hosts the code for distinguishing the combinations of the topology and the protocol. Depending on the input parameters, the topology and the protocol are implemented here. Apart from this the logic of stopping the actors once they have reached their convergence criteria and printing the time are also hosted in this file.

Implementation.scala :

It includes the methodology of starting the actors. The major functionality of this file is creating the topology with its respective characteristics. For example if the user enters 2D as the network topology and Gossip as the protocol, the Boss actor construct the topology by looking at the actors neighbors and storing them in an adjacency matrix for further message propagation.

Worker.scala:

This file consists of the information pertaining to a single actor. This includes the information of its neighbors which is stored as an array Buffer. Here Array Buffer has been used because the random addition. It hosts the logic of self-propagating the message to itself so that it delivers the message in parallel to its other neighbors while its neighbors are carrying forward the message

Interesting Observations:

The kind of convergence issues have inculcated a good understanding of the gossip and push sum protocols. In order to facilitate the convergence aptly we have introduced two new ass assumptions which basically are the number of starting nodes and the number of nodes an actor forwards the message to its neighbors once it closes to average.

The order of convergence of the network topologies from the results obtained has been fairly shown as

T(full) <T(imperfect 2D) < T(2D) < T(line)

Line topology has been the most difficult one to converge as the number of neighbors is limited for an actor and the message propagation is not that frequent as it is with the other networks

Relative performances of the various topologies have been fairly constant.

Maximum Values for Gossip

Full – 5000 Line – 1200 2D – 4300 Imp 2D – 4300 (rounded to near values)

Maximum Values for Push Sum

Full – 900 Line – 600 2D – 800 Imp 2D – 800 (rounded to near values)

How to Run:

1. *Deploy the files.*
2. *Compile the program files.*
3. *Initiate the process by typing :*

scala Project2 50 full gossip

For better convergence, we can add an extra parameter which specifies the number of nodes which are aware of the rumour.

scala Project2 50 full gossip 10

scala Project2 50 full push-sum 10

Possible values of topologies:

- Full
- line
- 2D
- imp2D

Convergence assumption:

We have seen that a few nodes are left over while almost around 90% of the total nodes reach convergence. Also, we have observed that gossip algorithm converge better than the push sum.

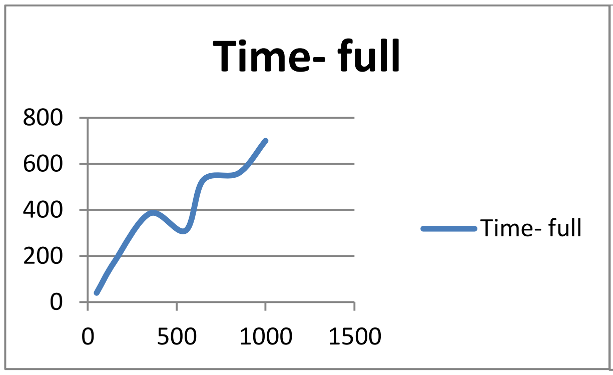
Hence, our assumption is that we are considering that the network has converged when 90% of the nodes terminate for gossip and 70% of the nodes for push sum.

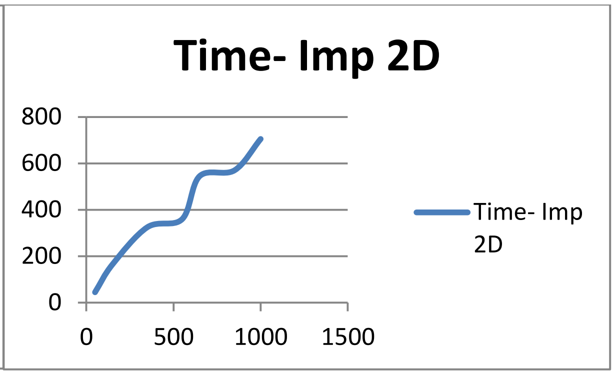
Results:

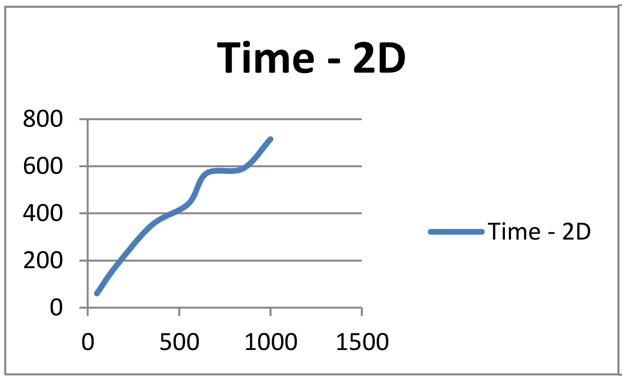
The time taken vs the number of actors is plotted as a graph for each of the network topology. The expected ordering of timing with each of the topology has been fairly met. The values and the graphs including the consolidated graph have been disclosed as under

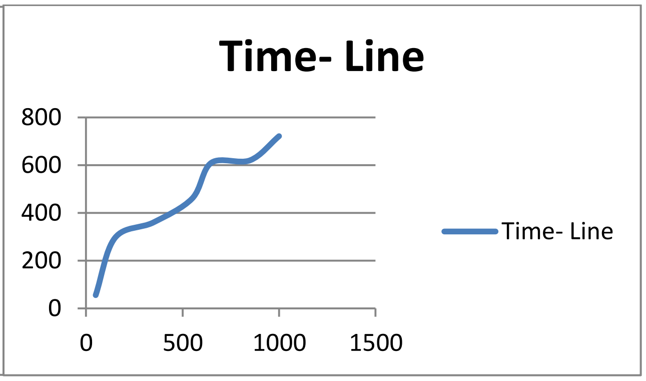
No. Actors	**Full(Time)**	**No. Actors**	**2D(Time)**
50	43	50	52
150	178	150	174
350	375	350	346
550	330	550	447
650	550	650	575
850	573	850	580
1000	720	1000	720
No. Actors	**Imp 2D(Time)**	**No. Actors**	**Line(Time)**
50	45	50	55
150	165	150	292
350	330	350	363
550	365	550	470
650	540	650	613
850	575	850	620
1000	708	1000	725

Individual Graphs for each of the topologies (Time – Yaxis and No of Actors X axis):

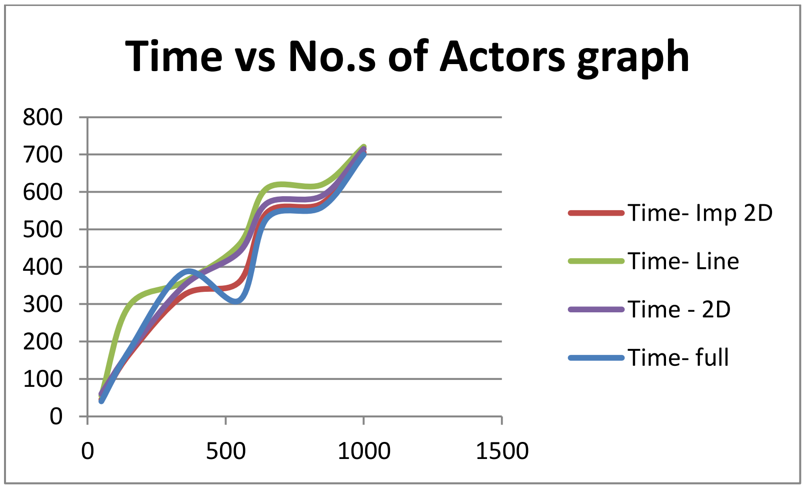




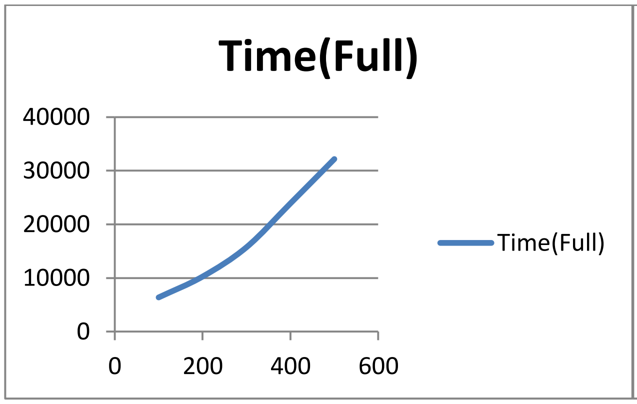


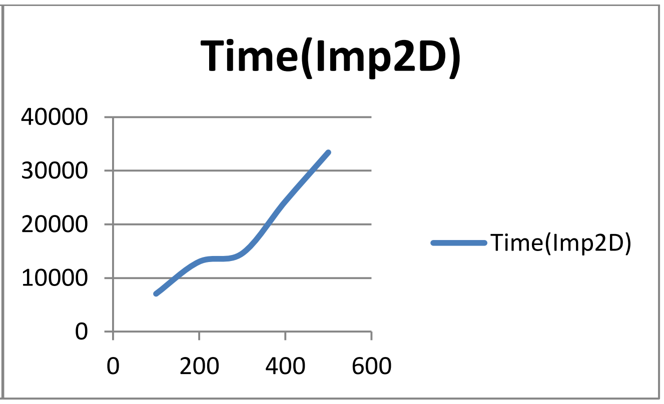


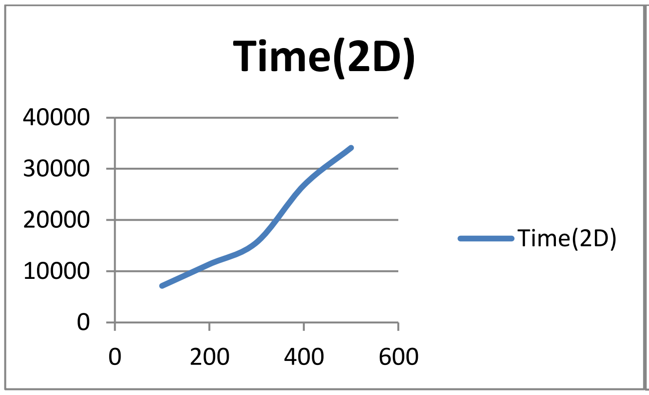
Consolidated Graph:

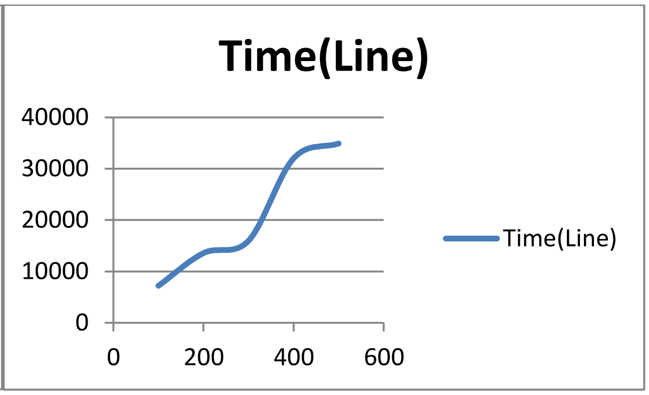


PUSH SUM:

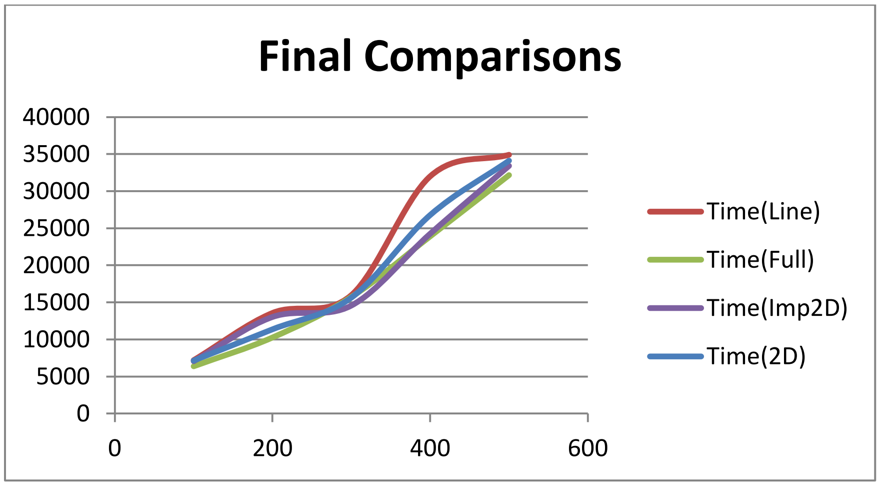








Consolidated Graph:


Final Comparisons
Time(Line)
Time(Full)
Time(Imp2D)
Time(2D)