COP5615: Distributed Operating Systems
Project -2

Submitted by:
Aarthi Kashikar, UFID: 0366-8968
Sri Kanth Juluru, UFID: 9279-1918

Specifications of the machine where the experiments were performed on are as follows.
Model: Dell Inspiron 5593
OS: Windows 10 Home
Processor: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
Memory: 16 GB RAM, 2667 MHz

Contents of zip folder:
1. Main project folder (project2)
2. Report.pdf

We can run the project by going inside the project2 folder, and run this command
dotnet fsi --langversion:preview proj2.fsx numNodes topology algorithm
In place of numNodes, we will pass number of nodes. Possible values of topology are "Line", "2D", "FullNetwork", and "Imperfect2D". Possible values of algorithm are "Gossip" and "Pushsum". For example, dotnet fsi --langversion:preview proj2.fsx 200 2D Gossip

Implementation details:

The implementation follows as below:
1) The commands number of nodes, topology and the algorithm to be implemented are received from the command prompt.
2) The given number of actors are spawned and placed in a list.
3) The chosen topology is implemented in the form of a map. The key of the mao is the index of the node and the value is a list of all the neighbour's of the node.
4) A Master actor is present that initiates the gossip or possum based on the command.
5) In the master actor, a random node is selected with the range as 0 and number of nodes -1. Based on the algorithm, desired information is sent to this particular node. In case of gossip, a message is sent and in case of possum, sum and weight are sent.
6) The specified algorithm is called. The implementation of the algorithm is implemented in the Node actor.
7) Once convergence is reached the master node calls for program termination.

Topologies:

When the number of nodes is Topologies are created using the concept of adjacency list with Map data structure containing node index as key and the list of it's neighbours as value.

Line:

Line topology is simple to implement. For each node, nodes on the right and left are added by incrementing and decrementing by 1 respectively. In case of the first node, the left node is skipped and for the last node of the line, the right node is skipped because they do not exist.

2D Grid:

In this topology, each node has 4 neighbour nodes except if they are on the edge of the 2D grid. The potential size of the side is found by calculating the square root of the number of nodes. Using this the nodes above and below any particular node is found, and for right and left nodes, index is incremented and decremented by 1 respectively. In case the number of nodes provided is not a perfect square, we do not get a perfect n*n grid. There will be some left over nodes if n*n grid is implemented. Here, we are considering all the nodes where the remainder nodes are added to the last line of the grid.

Imperfect 2D:

The implementation for this is similar to 2D. Including this for all nodes, a random index is also added. A check is made if the chosen random index is already in the list in which case a new random index is chosen. This is done till a random index which is not already added is found. The implementation of a non perfect square number of nodes is similar to that of 2D where remainder nodes are added to the last line of the grid.

FullNetwork:

For each node, all the other nodes in the topology are placed in the list and this key value pair is added to the map.

Gossip Protocol:

In the Gossip algorithm, the count of the node is incremented every time it gets the information. Once a node receives the information 10 times, it stops to transmit further to its neighbours. When all the nodes receive the gossip 10 times, it is said to be a convergence.

The implementation of the Gossip algorithm is as follows. When the master node is promoted for gossip algorithm, it picks up the random node chosen and sends a message. The node here contains a state called nodeCount which maintains the count of the number of times the message is received by that node/actor. Everytime the actor receives the message, the numCount in the state is incremented by 1. Then, from the node's neighbour list from the map, a random node is chosen using rand.Next(). The message is then transmitted to this neighbour which further sends it to any of its neighbours. Since this is asynchronous multiple nodes will be sending messages to any random neighbour at the same time. When a converged node receives the message again, a random node is chosen and the message is sent to it since according to the question, converged node should not transmit messages.When a node receives it 10 times, it sends a “Done” message to the master which maintains the count of the number of converged nodes. This count is maintained to terminate the program when all the nodes are converged.

Push Sum Algorithm:

In the Push Sum algorithm, each actor maintains sum and weight. Sum is initialized with the actor index and weight with 1. Master calls an actor node randomly and passes new sum and new weight. Receiving actor adds this sum and weight to the sum and weight stored in its state. Ratio of this updated sum and updated weight is calculated and compared to previous values. Current actor calls a random neighbor actor and updates itself with the sum and weight reduced to half. If the difference in ratio is less than 10^(-10) for 3 consecutive calls, then we treat it as convergence for that node. When all the node actors converge then the algorithm ends.

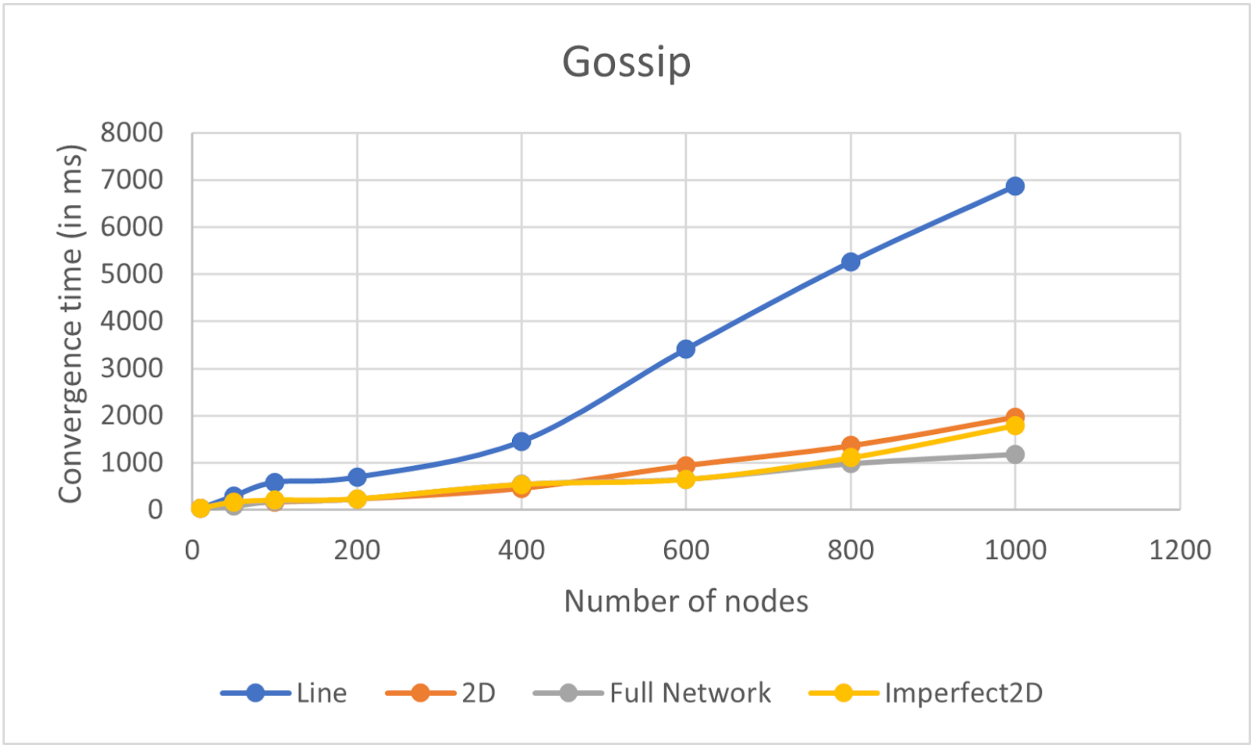
For implementing push sum algorithm, we maintain sum and weight in the worker actor state as we need this information in future calls. We initialize the sum value with the index of the actor node. In our case, the index of actor nodes starts with 0 and goes on till numNodes-1 where numNodes represents the number of worker actors. We calculate updatedSum by adding the actor's sum and incoming sum. Similarly, updatedWeight is addition of actor’s weight and incoming weight. Ratio of updatedSum and updatedWeight is taken and compared to the actor's previous ratio. If the absolute difference between ratio and previous ratio is less than 10^-10, we increase the counter. When the absolute difference check passes consecutively, the counter is incremented. Otherwise the counter is reset. When the absolute difference check passes 3 consecutive times, then we call it convergence for that node and increment the count of converged nodes at master. We reduce updatedSum and updatedWeight to half and will make a call to its neighbor node. Actor updates its state values by calling itself with these reduced values. When a converged actor is called, it will simply redirect that call to neighbor nodes with incoming sum and weight. Our algorithm is terminated (converged) when all the node actors converge.

Findings :

Gossip :

On running the algorithm with different topologies and with different numbers of nodes, we found the following findings. The line topology takes the longest to converge for all the ranges of nodes. That is because the nodes are only in contact with 2 nodes which makes the gossip travel less frequently. For the small range of nodes, the difference between the time taken to converge for 2D and imperfect 2D is less whereas for a bigger range of nodes, the difference is significant. There is only one additional neighbor for each node in imprefect2D which does not contribute much when the number of nodes is small and as the number of nodes increases the extra neighbour makes a difference. The full network topology takes the least amount for all the range of numbers. This is because every node talks to every other node in the topology which makes the nodes receive the gossip much faster.

	Convergence Time (in ms)			
Number of nodes	Line	2D	Imperfect2D	Full Network
10	36.8795	33.546	35.1892	36.4966
50	292.4072	133.4183	165.0061	77.9494
100	588.9099	166.5421	207.6605	181.301
200	697.6381	229.0418	235.5083	232.6787
400	1455.3856	459.6802	541.9026	547.0699
600	3414.9054	944.0032	646.3353	653.1377
800	5267.1551	1363.5483	1105.4552	985.4532
1000	6873.2375	1972.6267	1790.8469	1183.3927

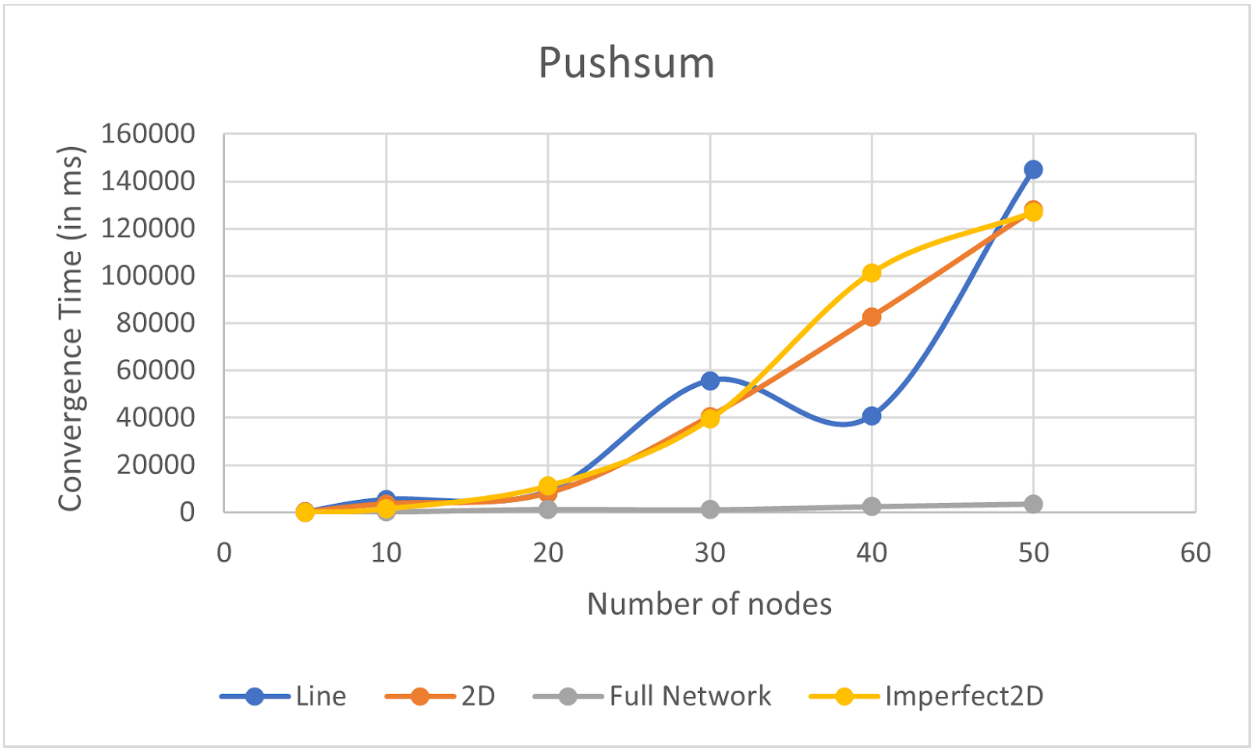


Pushsum:

PushSum algorithm is run with different topologies, different number of nodes and convergence time is mapped with number of nodes in the below attached graph. We can see that time for convergence is least for full network as each node is connected by all the remaining actors in the network. Hence even with random calls to neighbors, the chance of reaching different neighbors is high, which in turn results in less convergence time. Convergence Time is high for Line topology as there are a maximum of 2 neighbor nodes and chances of calling converged nodes is high. As the number of nodes increase, we see a steep increase in time taken. Convergence times for 2D and Full Network topologies lie between Line and Imperfect2D and we see a growth similar to linear.

For below mentioned values, convergence is treated as Absolute difference between previous ratio and current ratio is less than 10^(-3) for 3 consecutive times. This change was made as the machine we are running this algorithm is having difficulty in computation for higher accuracy (10^-10). We could successfully converge a few nodes with 10^-10 accuracy compared to 10^-3.

	Convergence Time(in ms)			
Number of nodes	Line	2D	Full Network	Imperfect2D
5	188.5635	175.2529	101.3004	69.0014
10	5402.3993	3612.7658	226.1925	1581.3986
20	8838.81	8107.5895	1324.4742	11242.0898
30	55745.686	40534.2698	1151.6073	39438.6439
40	40733.9098	82672.6769	2535.6354	101390.0806
50	144847.8182	127939.4873	3654.3778	127042.5377



Pushsum
Convergence Time (in ms)
160000
140000
120000
100000
80000
60000
40000
20000
0
0
10
20
30
40
50
60
Number of nodes
Line
2D
Full Network
Imperfect2D