Project – 2 Report (Gossip & Push-Sum Algorithm)

Date: October 10, 2022

Project Teammates

1. Aseem Baranwal (UFID: 84127415)
2. Debi Prasad Das (UFID: 22650204)

What is Working?

We were able to implement all four topologies – Full, Line, 2D and 3D and both the algorithms – Gossip and Push Sum that were required for the project. We were also able to implement the Bonus Part, where even if a node from our topology dies, the program doesn't fail, and the convergence case is reached.

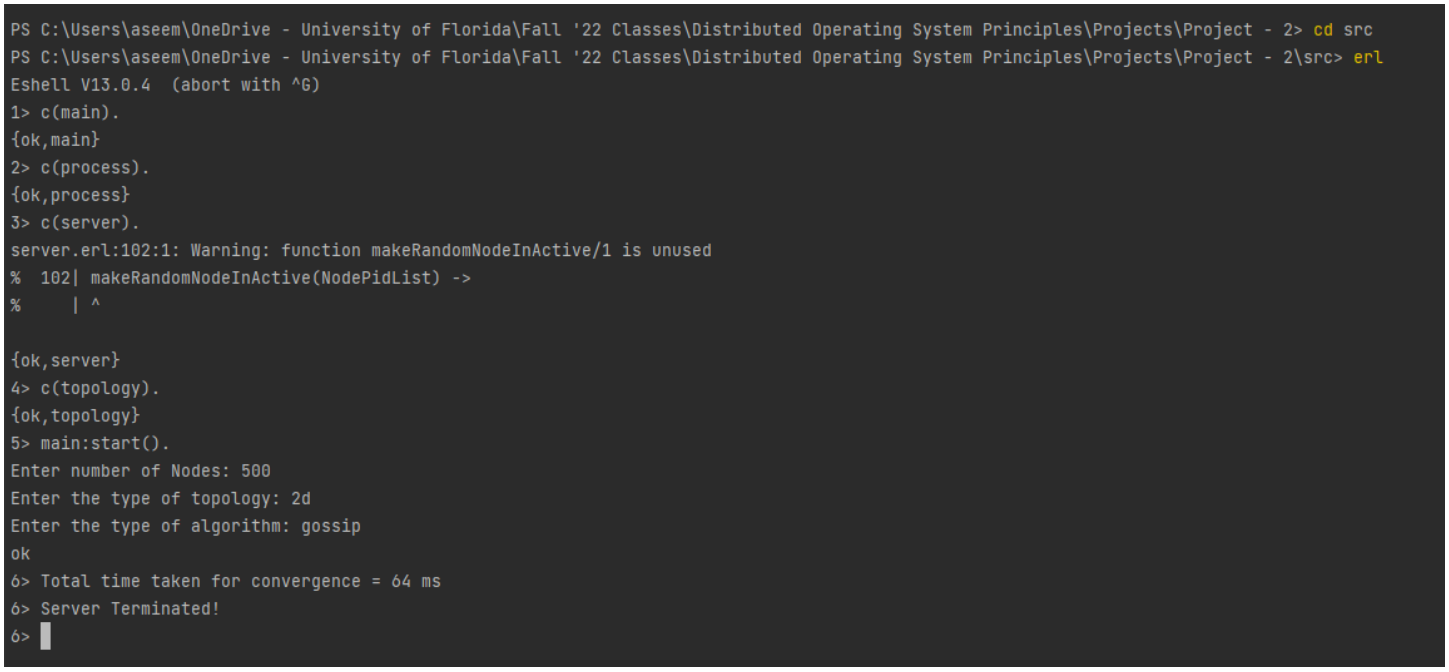
Topologies Implemented

Following topologies have been implemented using the ***gen_server*** module in erlang where each node has its own state (The state comprises of values including current **Convergence Count** for **Push-Sum** & **Gossip**, **State Value**, **Weight Value**, & **State/Weight Ratio**) -

1. **Full Topology** - A node has all the other nodes in the network except itself as its neighbor.
2. **Line Topology** – A node has its front and back as neighbors except the cases where first and last node have only 1 adjacent neighbor.
3. **2d Topology** – Nodes are arranged in a 2d matrix. Each node can talk to its adjacent and diagonal neighbor.
4. **3d Topology** – Nodes are arranged in the same way as 2d but here each node can talk to its adjacent and diagonal neighbor and another random node from the matrix which is not present adjacently or diagonally to the selected node.

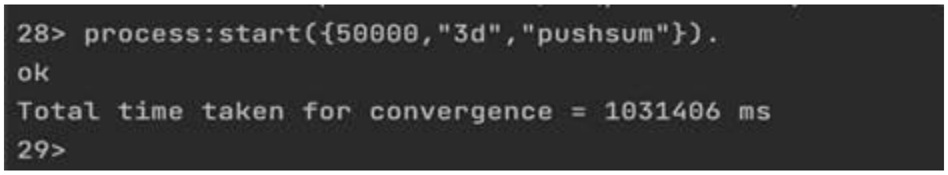
Code Execution Details

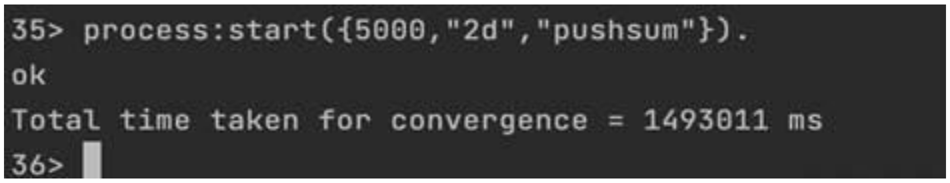
1. We have designed 4 erlang files - *main.erl, process.erl, server.erl and topology.erl*
   a) ***Main.erl*** – This file is the starting file for the project and starts our application.
   b) ***Process*.erl** – This file initializes different processes based on the input provided by the user including the node number, Topology Type and the Algorithm Used.
   c) ***Server*.erl** – This file is implemented as a **supervisor** node for our whole program which takes care of how the processes are working and terminates the program when a particular state (No Neighbor list found or No Alive Nodes Found) is reached and is also responsible for letting the user know that the convergence has taken place.
   d) ***Topology.erl* –** This file is being used to generate the neighbors for a particular process node and the nodes are later saved in the state of the process node.
2. First, we will compile all the files using command: ***c(filename). {Please don't include ".erl" while compiling}.***
3. After all the files are compiled, we start execution of our project using command: ***main:start()***.
4. Run the program in the manner given below (PTO)

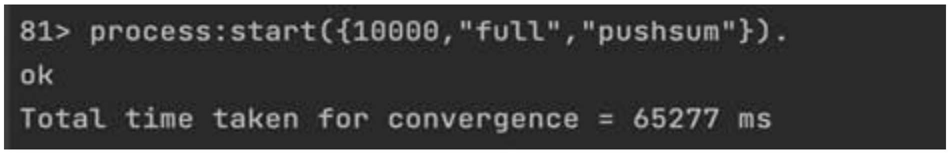

```
PS C:\Users\aseem\OneDrive - University of Florida\Fall '22 Classes\Distributed Operating System Principles\Projects\Project - 2> cd src
PS C:\Users\aseem\OneDrive - University of Florida\Fall '22 Classes\Distributed Operating System Principles\Projects\Project - 2\src> erl
Eshell V13.0.4  (abort with ^G)
1> c(main).
{ok,main}
2> c(process).
{ok,process}
3> c(server).
server.erl:102:1: Warning: function makeRandomNodeInActive/1 is unused
%  102| makeRandomNodeInActive(NodePidList) ->
%     | ^

{ok,server}
4> c(topology).
{ok,topology}
5> main:start().
Enter number of Nodes: 500
Enter the type of topology: 2d
Enter the type of algorithm: gossip
ok
6> Total time taken for convergence = 64 ms
6> Server Terminated!
6> 
```

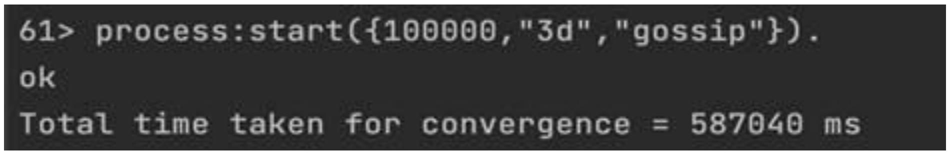
5. It will ask for 3 values from the user:
    a) ***Enter number of nodes:***
    b) ***Enter the type of topology to be used***:
       (Topology types are full, line, 2d and 3d)
    c) ***Enter the type of algorithm:***
       (Algorithm types are *gossip* and *pushsum*)
6. After we enter all the values, the code starts its execution.

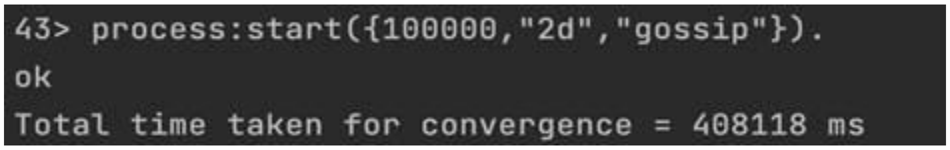
Following are the screenshots attached for our code execution:


```
28> process:start({50000,"3d","pushsum"}).
ok
Total time taken for convergence = 1031406 ms
29>
```

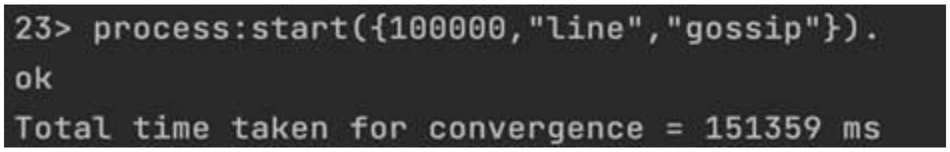

```
35> process:start({5000,"2d","pushsum"}).
ok
Total time taken for convergence = 1493011 ms
36> 
```


```
81> process:start({10000,"full","pushsum"}).
ok
Total time taken for convergence = 65277 ms
```


```
61> process:start({100000,"3d","gossip"}).
ok
Total time taken for convergence = 587040 ms
```



```
43> process:start({100000,"2d","gossip"}).
ok
Total time taken for convergence = 408118 ms
```



```
23> process:start({100000,"line","gossip"}).
ok
Total time taken for convergence = 151359 ms
```

Gossip Algorithm Implementation

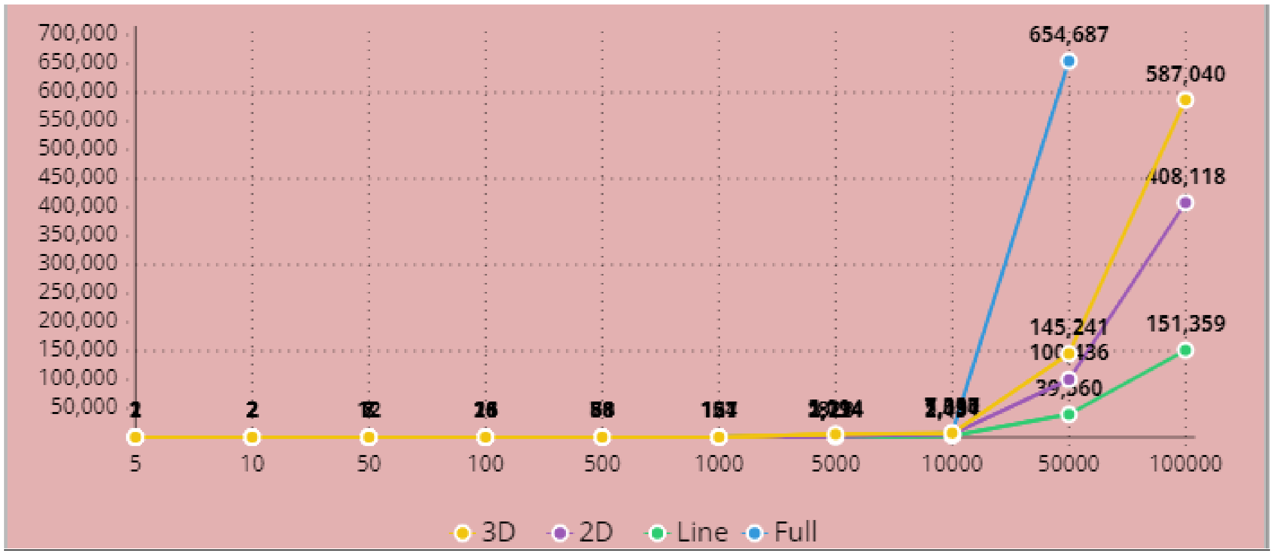
1. When we start the Gossip Algorithm, first we select a random node from the node list in our topology. {Handled in *server*}
2. The selected node selects a random node from its neighbor list and sends a gossip message "Hello" and increments its counter value by 1.
3. The receiver node hears the rumor from the sender node and increases its counter value by 1. Now the current receiver becomes the sender node and selects a random alive node from its neighbor list and sends a gossip message to the selected node.
4. If no alive node is found, the **supervisor (server) process** takes control (which runs in parallel with our main process) and removes the node from the **aliveNode** list, since it can't transmit the message to any of its neighbors. Now that it is done, the server selects a suitable alive Node at random and sets it as the new sender and sends the call back to our main process and normal execution continues in our main process.
5. The selected node hears the rumor and increments its counter value like step – 3 and again it selects a random node from its neighbor list and starts transmitting the gossip message.
6. This process will go on till all the nodes have heard the rumor K times (here we have taken the **GossipConvergenceLimit** as 10).
7. If any node has heard the rumor equal to **GossipConvergenceLimit** times, it stops transmitting the gossip message and becomes dead (not Alive) and gets removed from the alive node list kept on the server, and again the call is passed on to the server process and same process happens as described in Step – 4.
8. The Gossip convergence is reached when all the nodes have heard the rumor **GossipConvergenceLimit** times and all nodes in our node list are dead.
9. The largest network of nodes we tried in our project is 10,000. We ran it on i5 processor.

Results of Gossip Algorithm Implementation

The chart & table for the convergence of various topologies for different nodes is summarized below:

Topology	Number of Processes Run									
	5	10	50	100	500	1000	5000	10000	50000	100000
Full	1	2	7	21	80	167	2114	7337	654687	NULL
Line	1	2	8	16	58	121	828	2451	39560	151359
2d	2	2	7	16	68	151	1734	5094	100436	408118
3d	1	2	12	18	71	154	5094	7110	145241	587040

Times to attain convergence for Gossip algorithm (in milliseconds)



Interesting Findings of Gossip Algorithm

1. In the case of Full Topology, if we increase the number of processes to 100000 then we weren't able to find the time of convergence for Gossip Algorithm.
2. On increasing the number of processes, Line Topology take least time out of all topologies to attain convergence for Gossip Algorithm.
3. We can deduce that for gossip algorithm, **more the number of neighbors in any arrangement, more the time it takes to converge**.

Push-Sum Algorithm Implementation

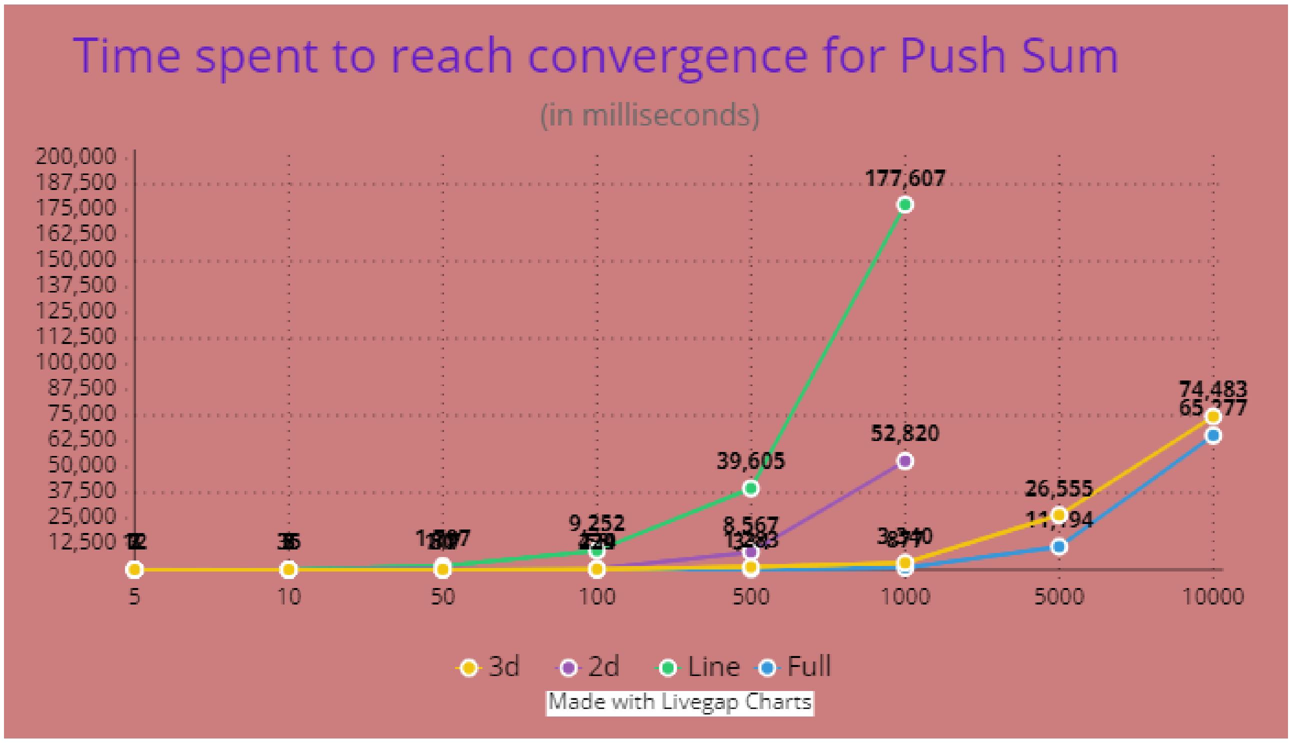
1. When we start the Push-Sum Algorithm, first we select a random process node from the node list in given topology like how we implemented in the Gossip Algorithm above.
2. Our program uses ***gen_server*** to maintain its own state where each process in our process list maintains its *state value, weight value*, and *s/w* ratio along with its *neighbor list*.
3. The selected node selects a random node from its neighbor list and **transmits half of its s and w values to the receiver node** and **updates its own values to s/2 and w/2**.
4. The receiver node receives the s and w values from the sender and adds it to its own s and w values. New s, w values and s/w ratio get updated for the receiver node. The receiver then transmits half of its new s and w values to a random node selected from its neighbor list and this process continues from step 3.
5. A node is converged when its s/w ratio remains unchanged for ***PushSumLimit*** (here taken as 3) consecutive rounds. By unchanged, we mean that the ratio change is less than 10^{-10}. In such as case, it gets removed from the alive node list kept on the server.
6. The **final Convergence** is reached when all the nodes have reached their ***PushSumLimit*** (the value remains unchanged for successive rounds).
7. This whole process continues till all the nodes in our topology are dead.

Results of Push-Sum Algorithm

The chart & table for the convergence of various topologies for different nodes is summarized below:

Topology	Number of Processes Run							
	5	10	50	100	500	1000	5000	10000
Full	2	7	31	58	334	877	11194	65277
Line	12	35	1797	9252	39605	177607	1493011	NULL
2d	1	8	117	439	8567	52820	NULL	NULL
3d	3	5	80	224	1283	3340	26555	74483

Times to attain convergence for Push-Sum algorithm (in milliseconds)



Interesting Findings of Push-Sum Algorithm

1. On increasing the number of processes, Line Topology and 2d Topology take more time out of all topologies to attain convergence for Gossip Algorithm.
2. We couldn't find the time to attain convergence for Push-Sum algorithm for Line and 2d Topology if we increase the number of processes greater than 5000, which was possible in case of Gossip Algorithm. Therefore, we can have the intuition that the graph for convergence versus time is exponential and has steeper curve than what we saw for Gossip curve for various topologies.
3. For large number of nodes, we find that**, more the number of neighbors, less is the time required to converge.**

What is the largest network you managed to deal with for each type of topology and algorithm?

Topology	Largest Network (Node Size)	
	Gossip	Push-Sum
Full	50000	10000
Line	100000	2000
2d	100000	5000
3d	100000	50000