

COP5615: DOSP Project 2 Report

Name: Pranath Reddy Kumbam

UFID: 8512-0977

Problem Statement

Implement algorithms for group communication and for aggregate computation namely, Gossip and Push-Sum and compare and benchmark the convergence time for each algorithm on various topologies.

language: Erlang

Implementation

As suggested in the problem statement, For Gossip, a node in the network stops transmitting once it has heard the rumour ten times. While the “full” topology is able to achieve complete convergence, other topologies are tested based on a set required percentage of convergence (Ratio of nodes that need to converge). In the case of Push Sum, a node terminates if the ratio s/w does not change more than 10^{-10} in 3 consecutive rounds. We record the convergence time in milliseconds for both algorithms while testing on all four topologies. Due to the randomness involved in the algorithms, we take the average of five tests.

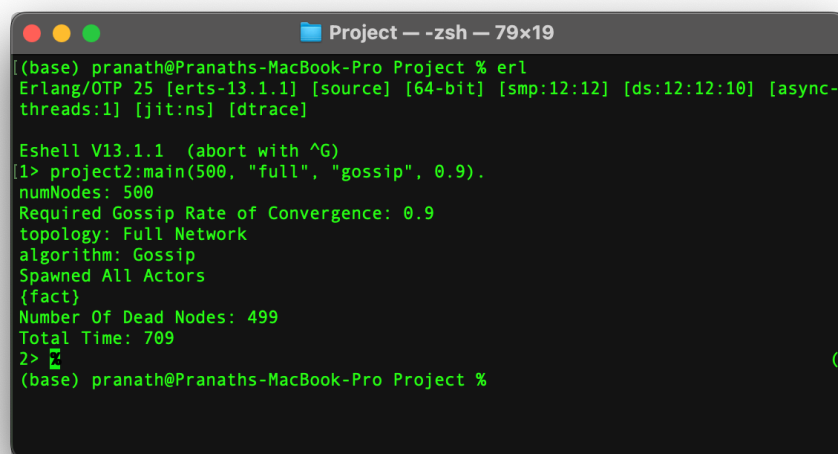
Execution Details

The program takes four parameters, number of nodes, the algorithm, topology, and lastly the rate of convergence required for gossip.

-> `c(project2).`

-> `project2:main(500, "full", "gossip", 0.9).`

Examples



```

Project — -zsh — 79x19
(base) pranath@Pranaths-MacBook-Pro Project % erl
Erlang/OTP 25 [erts-13.1.1] [source] [64-bit] [smp:12:12] [ds:12:12:10] [async-
threads:1] [jit:ns] [dtrace]

Eshell V13.1.1 (abort with ^G)
[1> project2:main(500, "full", "gossip", 0.9).
numNodes: 500
Required Gossip Rate of Convergence: 0.9
topology: Full Network
algorithm: Gossip
Spawned All Actors
{fact}
Number Of Dead Nodes: 499
Total Time: 709
2>
(base) pranath@Pranaths-MacBook-Pro Project %
  
```

Figure 1: Example of Gossip

```

Project — -zsh — 79x19
((base) pranath@Pranaths-MacBook-Pro Project % erl
Erlang/OTP 25 [erts-13.1.1] [source] [64-bit] [smp:12:12] [ds:12:12:10] [async-
threads:1] [jit:ns] [dtrace]

Eshell V13.1.1 (abort with ^G)
[1> project2:main(600, "full", "push-sum", 0.9).
numNodes: 600
Required Gossip Rate of Convergence: 0.9
topology: Full Network
algorithm: Push-Sum
Spawned All Actors
Started Push Sum
{fact,0,0}
Number Of Dead Nodes: 599
Total Time: 1676
2>
(base) pranath@Pranaths-MacBook-Pro Project %

```

Figure 2: Example of Push Sum

Submission Details

The submission zip file contains the code, screenshots, times (all the recorded values used for the plots), results, and plots.

Results

Push Sum

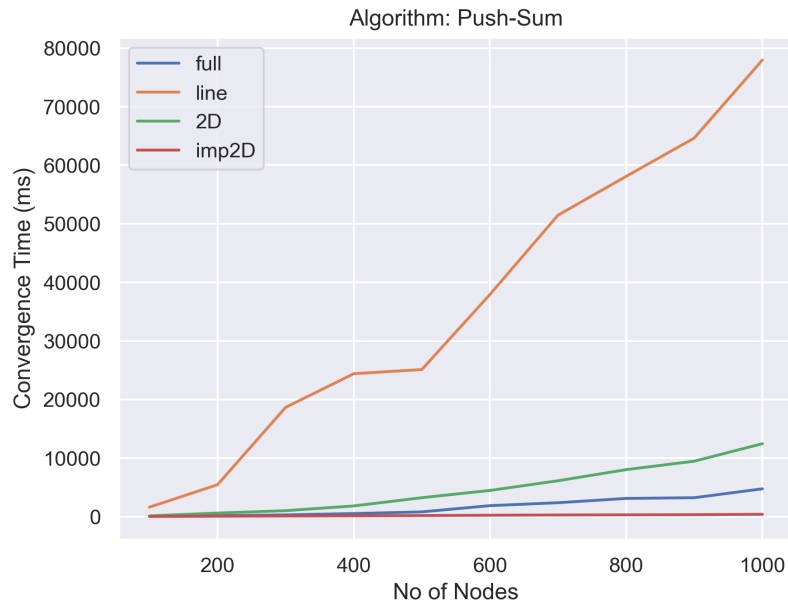


Figure 3: Dependence Plot (convergence times vs nodes)

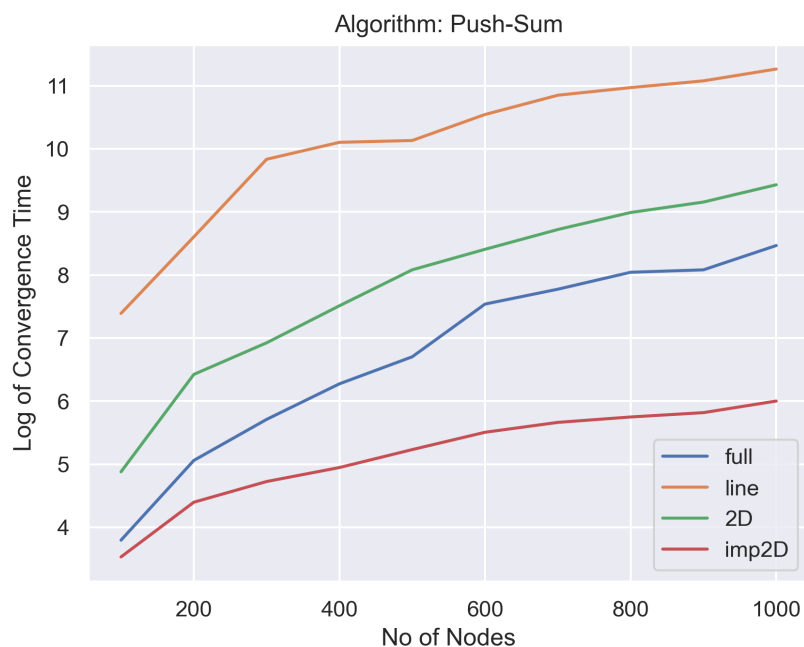


Figure 4: Log Dependence Plot (log(convergence times) vs nodes)

Gossip

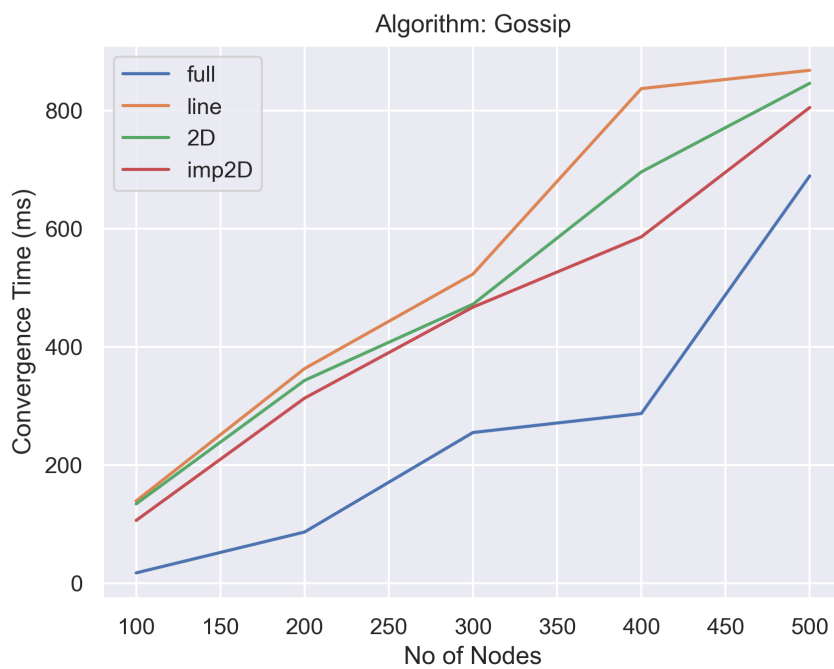


Figure 5: Dependence Plot (convergence times vs nodes)

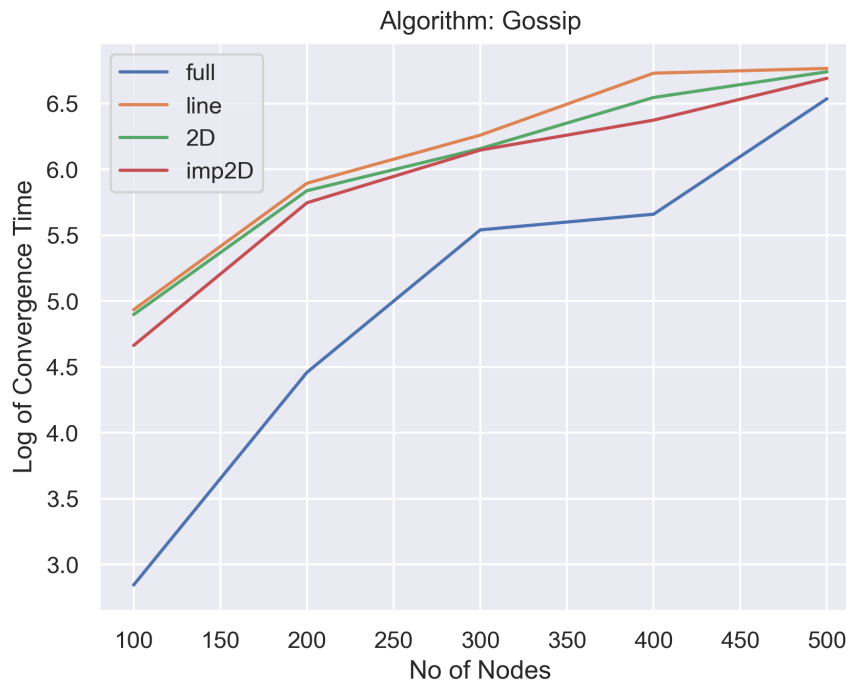


Figure 6: Log Dependence Plot ($\log(\text{convergence times})$ vs nodes)

Observations and Notes:

1. As seen in the results, line topology has the maximum convergence time for both algorithms. This makes sense since each node in the line network can only communicate with two adjacent neighbours making it the most constrained topology.
2. One would expect the “full” topology to have the least convergence time, but in the case of Push-sum, we see that an imperfect 2D grid has the lowest convergence time, which could be due to the addition of a random neighbour for each node. But considering the nature of the topology, Full might be a better choice if the spread of rumour is more important than the speed of convergence.
3. In Gossip, while the “full” topology is able to converge completely, the other topologies fail to pass the rumour to all the nodes in the network. Line, in particular, is notoriously slow and inconsistent and is often prone to quickly converging to a node whose both neighbours are terminated.
4. Overall, both Gossip and push-sum show similar characteristics.

Note: While testing, due to the nature in which the actors are generated, the shell needs to be restarted for each run, or else we might run into a bad argument error or get stuck at a particular node.

Largest Network Run:

	Gossip	Push Sum
Full	10,000	10,000
Line	5,000	5,000
2D	5,000	10,000
imp2D	5,000	10,000
