COP5615: Dist Oper Sys Princ, Fall 2018

Project 2

Date - 10/1/2018

Team Members

Sourav Dutta (9863 3443)

Coding environment

Erlang/OTP 21

IEx 1.7.2

Linux Ubuntu 18.04 machine with 6 cores

Part 1

Instructions

The code is in directory **project2**

Command format: mix run lib/proj2.exs <number of nodes> <topology> <algorithm>

e.g.

```
cd project2
mix compile
mix run lib/proj2.exs 3600 3D push-sum
mix run lib/proj2.exs 2000 full gossip
```

The plots were generated using:

```
python python/get_results.py gossip
Python python/plot_results.py gossip
```

Parameters:

1. **Number of nodes:** This denotes the size of the network and accepts an integer value. For rand2D and torus the value is rounded of to the nearest square where as for 3D it is rounded of to the nearest cube.

2. **Topology:** This is the topology of the network created and should be one of **'full', '3D', 'rand2D', 'torus', 'line'** or **'impline'** without the quotes.

3. **Algorithm:** This value denotes which algorithm to test and takes in either **'gossip'** or **'push-sum'** without the quotes.

Common details

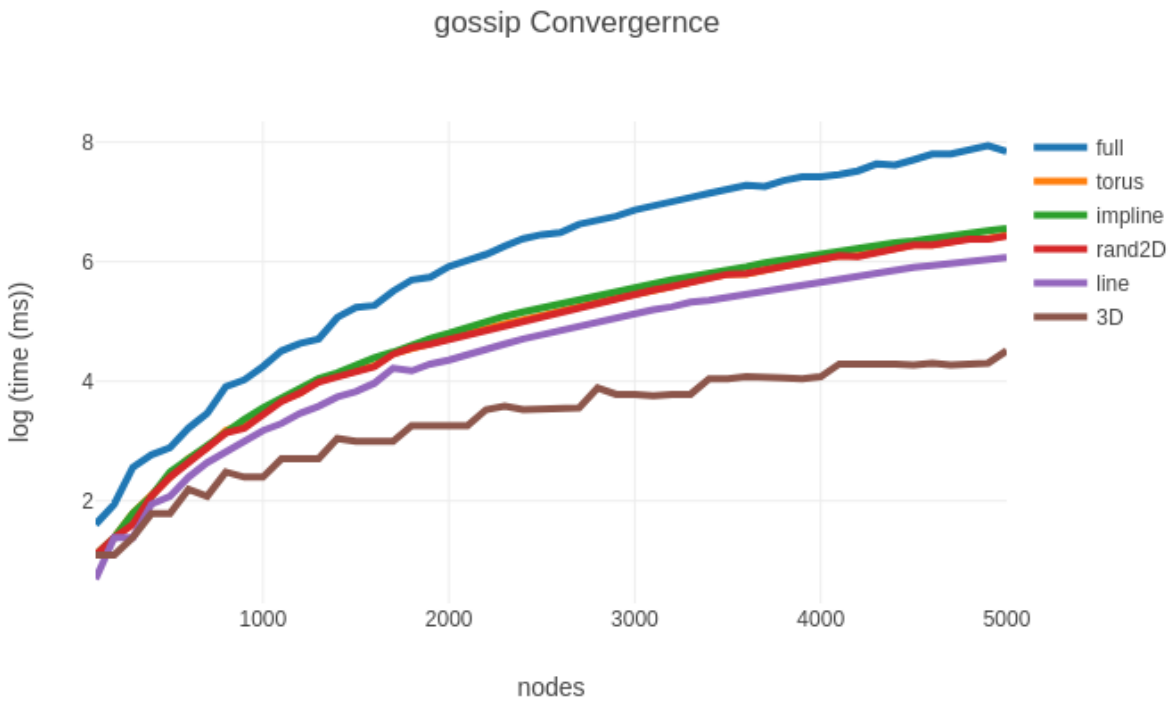
1. Maximum number of nodes tested is 5000.
2. The last line of the output is the time in milliseconds. The time difference is calculated by subtracting the time before the algorithm started from the time at which the algorithm converged.

Gossip protocol

In this protocol one of the random actors was sent a message to start. Once an actor received a message it started to spread the rumor to random neighbors at each round. The actor stopped spreading the rumor if it has received the rumor back 10 times. The algorithm converges once all actors has received the message at least once.

The gossip protocol was run on all 6 topologies. The experiment was conducted by varying the number of nodes from 100 to 5000 with step count of 100 for each topology.

The timings for convergence is measured in millisecond and the log of the same value is used for visualization below.



Note: Rand2D and torus plots are overlapping.

Push-Sum protocol

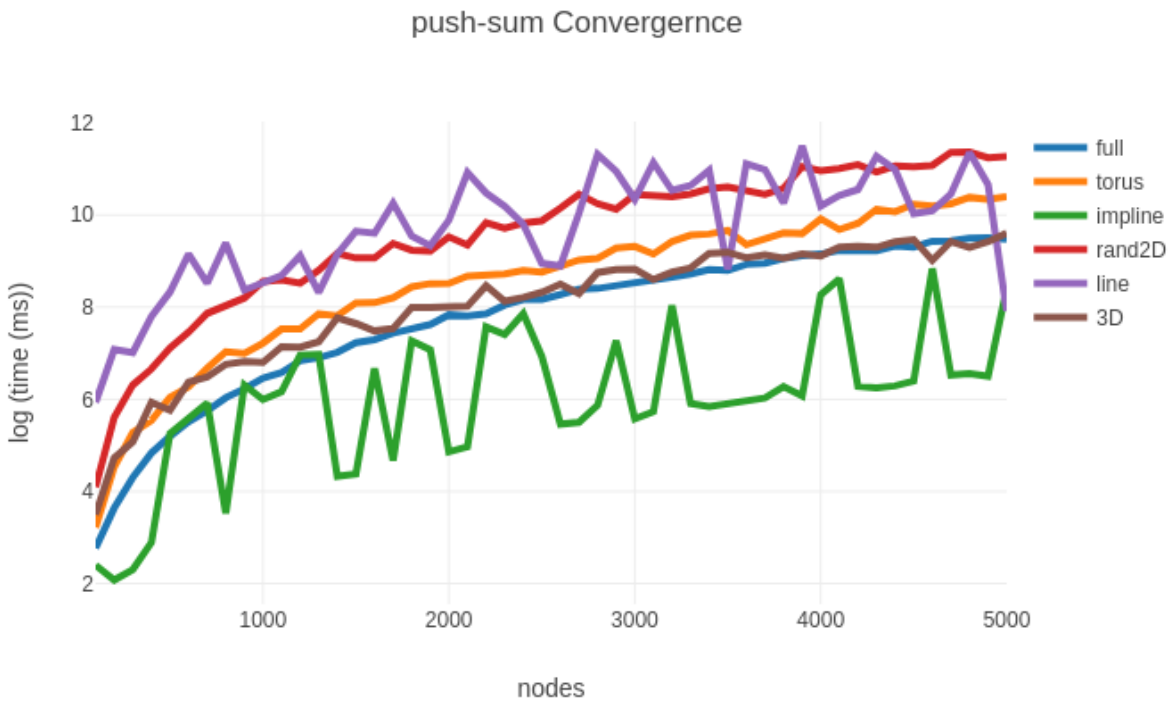
For the push-sum protocol, each actor was initialized with a number s = i, and w = 1. One of the actors was sent a message to start the process. At each step the actor received a set of numbers {s, i}. The actor then summed the incoming s and its own s. Similarly, it summed the incoming w and its own w. Then the actor kept the half of those values for itself and send half of those values to another random neighbor. At each step the ratio of the new s/w was checked if the absolute difference between the previous ration and current ratio was within 10^{-10}. If the ratio remained same for 3 consecutive runs for any one node then the algorithm converges.

The algorithm was very unstable if all nodes had to be converged as it did not converge most of the time. This is mostly because the algorithm reached a state all the neighbors of the current node terminates due to terminating condition.

I used two scenarios. One where the nodes were initialized with numbers 1 to n, where n is equal to the number of nodes. The other scenario was where I randomly assigned nodes with numbers in range 1 to 10000.

The push-sum protocol was run on all 6 topologies. The experiment was conducted by varying the number of nodes from 100 to 5000 with step count of 100 for each topology.

The timings for convergence were measured in millisecond and the log of the same value was used for visualization below.



Bonus

Instructions

The code is in directory **project2-bonus**

Command format: mix run lib/proj2.exs <number of nodes> <topology> <algorithm>

e.g.

```
cd project2-bonus
mix compile
mix run lib/proj2.exs 3600 3D push-sum
mix run lib/proj2.exs 2000 full gossip
```

1. The probability of failure can be changed in function should_fail in file lib/utility.ex
2. The time for sleep can be changed for individual algorithms in files lib/push_sum_network_node.ex and lib/gossip_network_node.ex under the condition “if Proj2.Utility.should_fail() do”

Maximum number of nodes tested is 5000.

In the bonus part a temporary failure was introduced in actors. On receiving a rumor in case of gossip and message in case of push-sum, a node can sleep for 20 milliseconds with a probability p. Increasing the sleep time increased the runtime of the program quite a lot as the usual runtime for most cases was roughly in the order of 10 to 500 milliseconds. So, the probability p of failing a node at each step was varied from 0.1 – 0.4.

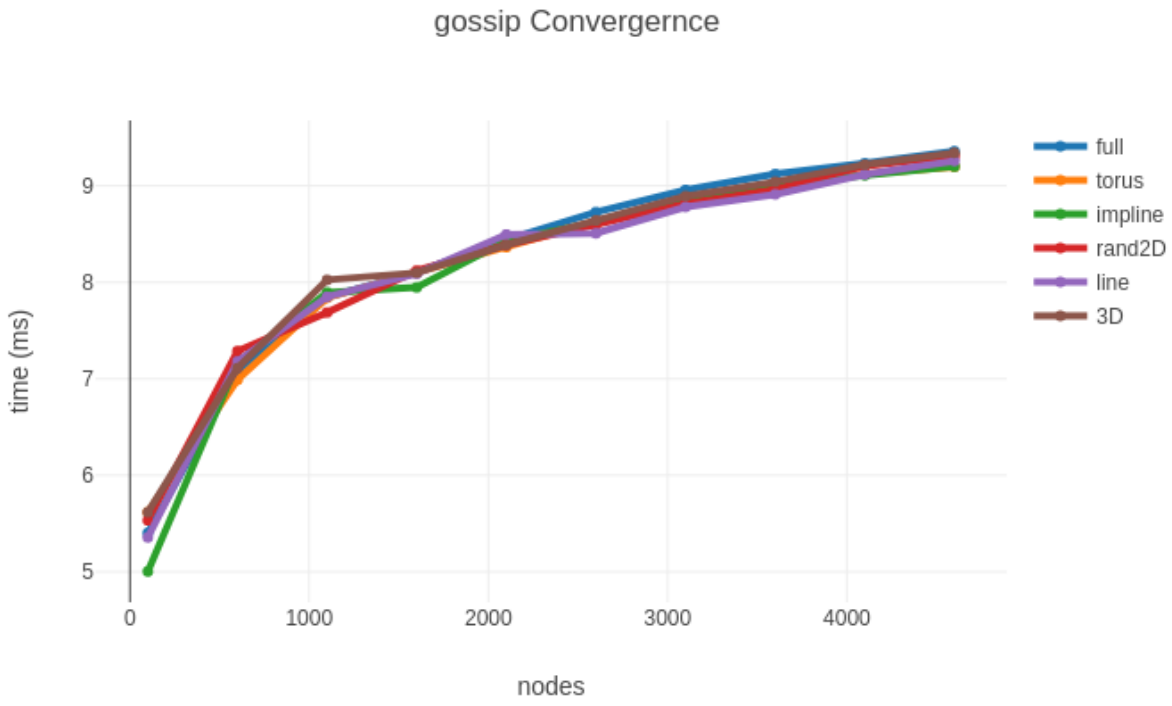
Observation for gossip: A temporary failure in node disables the node from processing incoming messages. It resumes when it comes back online. As it can be seen in the first graph, with a small value of sleep, all the algorithms perform roughly in similar way. This can be due to small running time of the algorithms. Any difference in those small run times are nullified by the temporary failures.

As the failure probability increases, the time taken for different algorithms changes. 3D topology started to take more time than others to converge. However, line topology performed similarly with failures. One of the reasons could be that for line to perform this way can be because the number of neighbors were at most 2 for line. For most cases the failure would happen to a node which has already heard a gossip. So, once few messages have been passed around and the gossip has travelled some node, a failure in such a node is less likely to affect the left and right side of that node as they have already heard the gossip and have started spreading it.

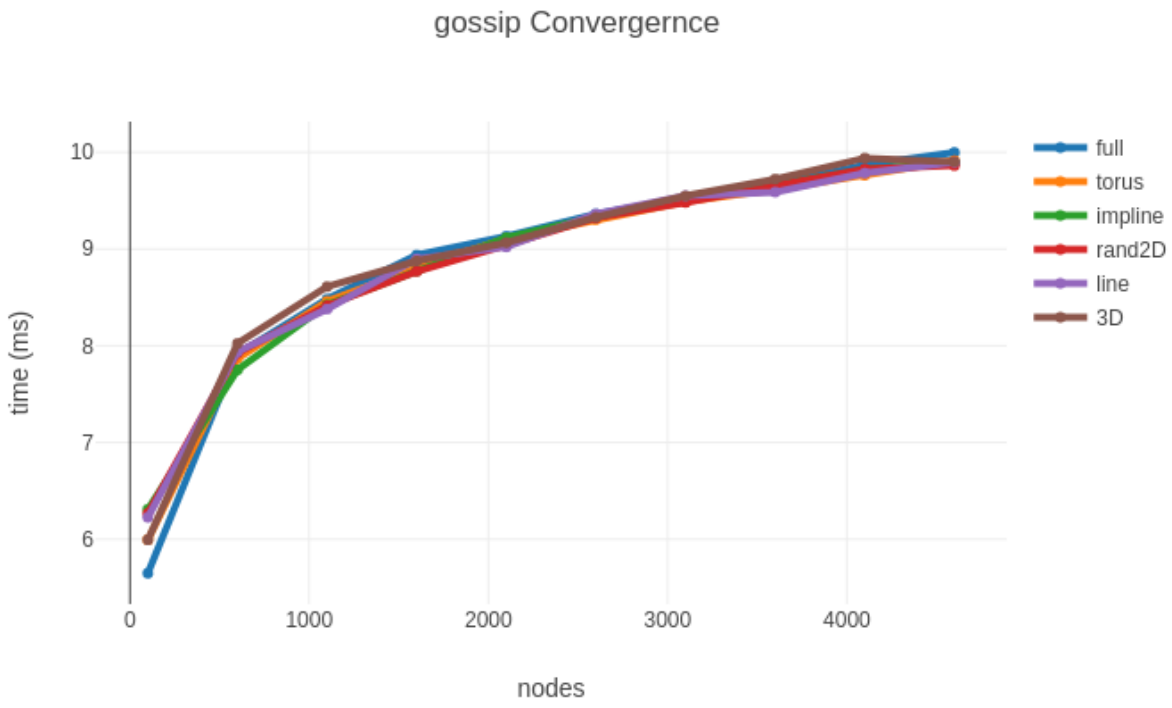
But for a 3D topology, since there are more neighbors, there are more chances of a new node failing and thus reducing the overall transfer of gossip.

The graphs for different probability of failures for gossip are below:

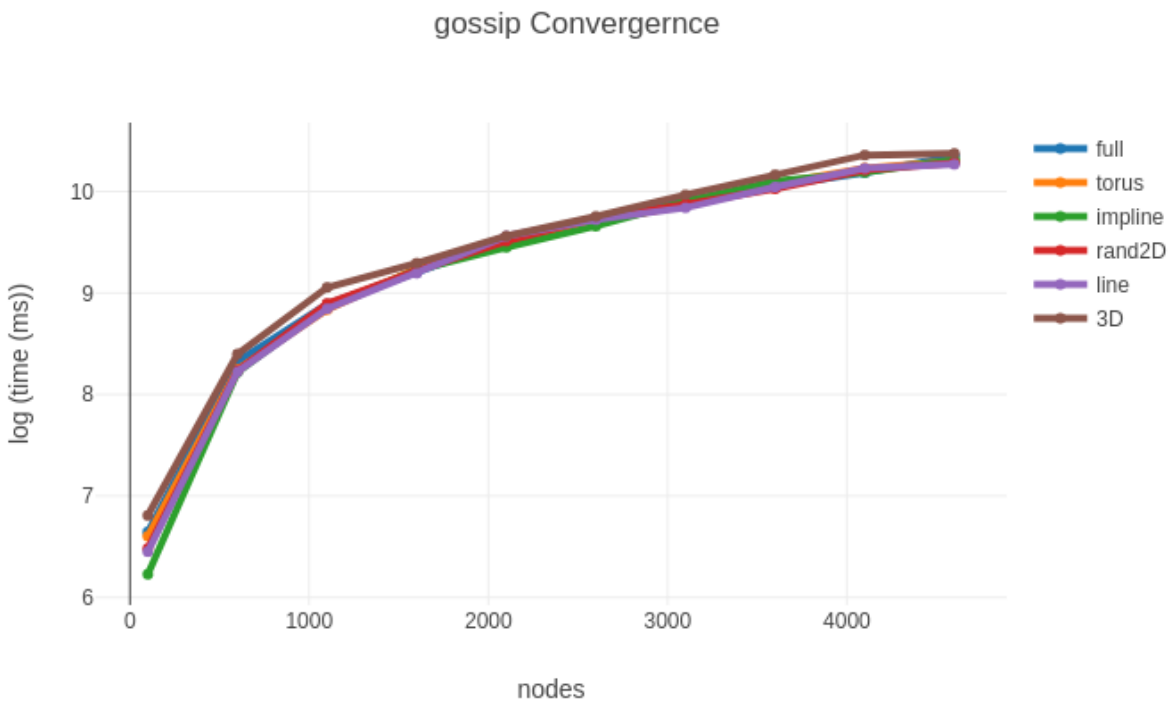
10 percent chance of failure:



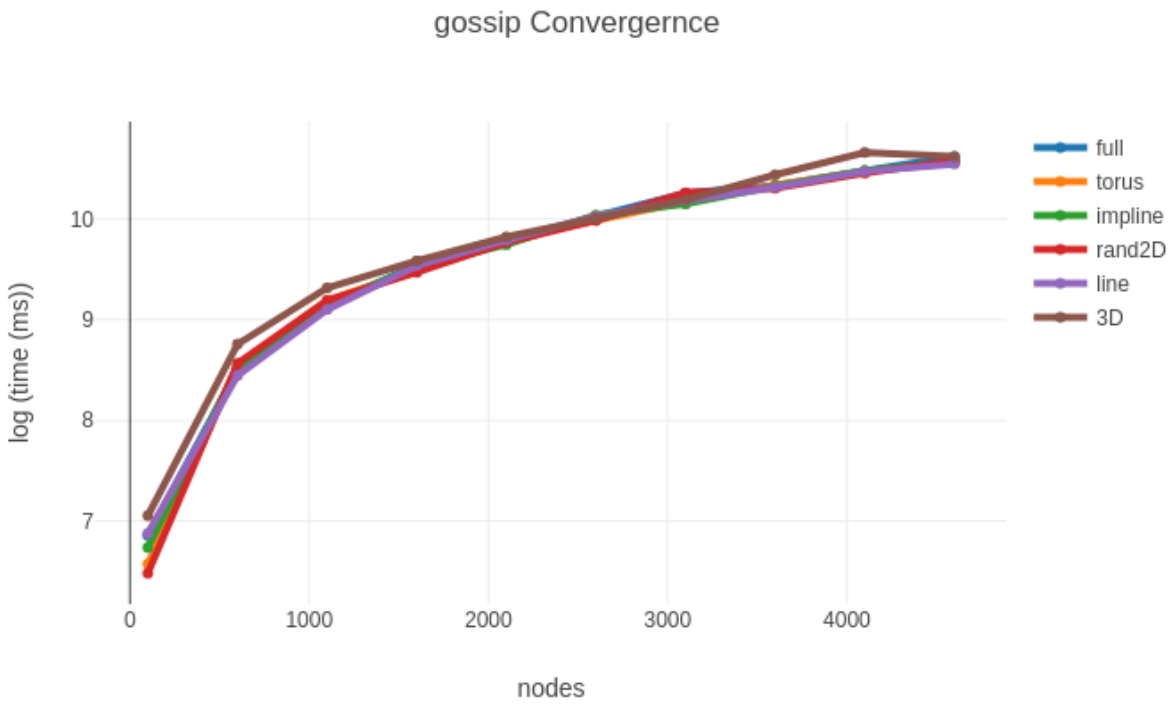
20 percent chance of failure:



30 percent chance of failure:



40 percent chance of failure:



The push-sum algorithm takes a lot of time to converge with a time delay.