COP 5615 Fall 2014

Distributed Operating Systems
Alin Dobra

Project 2

Gossip Simulator using Akka and Scala

Submitted By:

Team Member : 1

Name : Abhishek Singh Panesar
Email : asp.abhi18@ufl.edu
UFID : 5111-8921

Team Member : 2

Name : Nishit Sadhwani
Email : n.sadhwani@ufl.edu
UFID : 5393-5598

How to run the code

Normal: Project2.scala

scalac Project2.scala
scala Project2 numNodes topology algorithm

Bonus: Project2bonus.scala

scalac Project2bonus.scala
scala project2 numNodes topology algorithm Num_FailingNodes

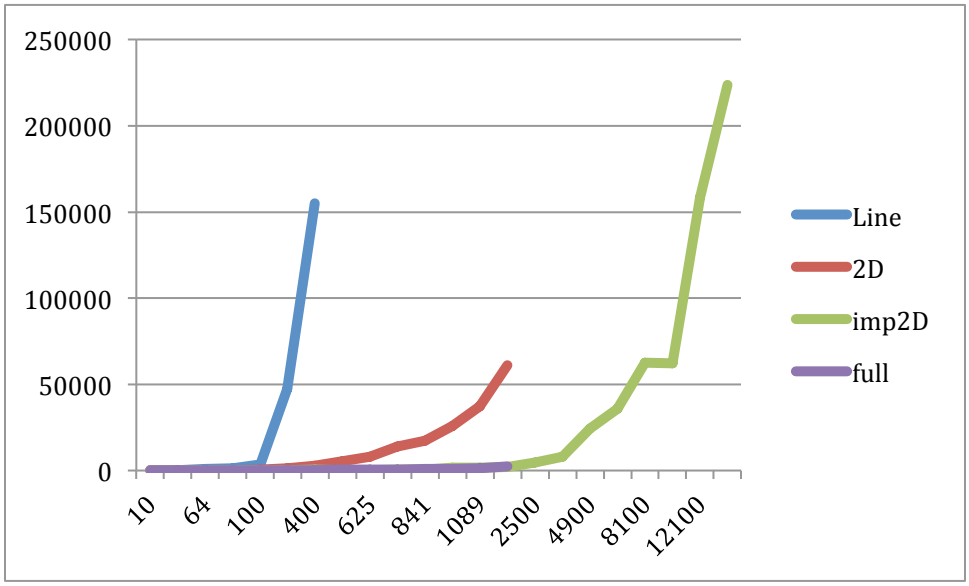
In this you need to give number of nodes that you want to fail as an argument (4 th argument).

What is working

We implemented both the normal Gossip Simulator and the simulator with Failure of certain nodes, basically we implemented 80% failure rate and 10% failure rate for the Push Sum algorithm. We also implemented a 40% failure rate for the Gossip algorithm.
First implemented the 4 topologies, namely line, 2D, imp2D and full. After the topologies were created, we made a master which is responsible only for sending the first message and then responsible for finally turning down the whole system.
The master sends the first message after matching the topology. Master sends the first message to gossip algorithm if the user gives input as gossip and the Master sends the first message to pushsum if the user gives input as push sum. The nodes in the network then take care of themselves and the message passing.

PushSum Graphs

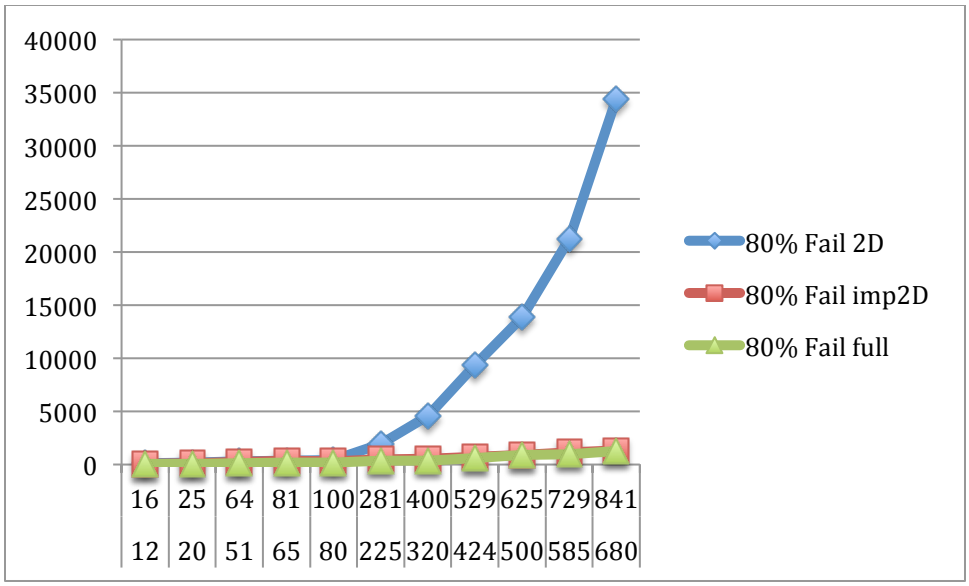
Without Failure Model:



On Y-axis, you see the convergence time of the algorithm and on X-axis you find the total number of nodes.
The graph can be interpreted, as the Line network is the worst because it has a steep rise even when the number of nodes is less. Next we can see that even 2D network goes steep after a certain number of nodes, it's the second worst network in this case. Next we see that full and imp2D are almost similar, the two graphs compliment each other. The imp2D was able to run for more number of nodes that is 14400 because it had a low building time, whereas full network had a larger building time.

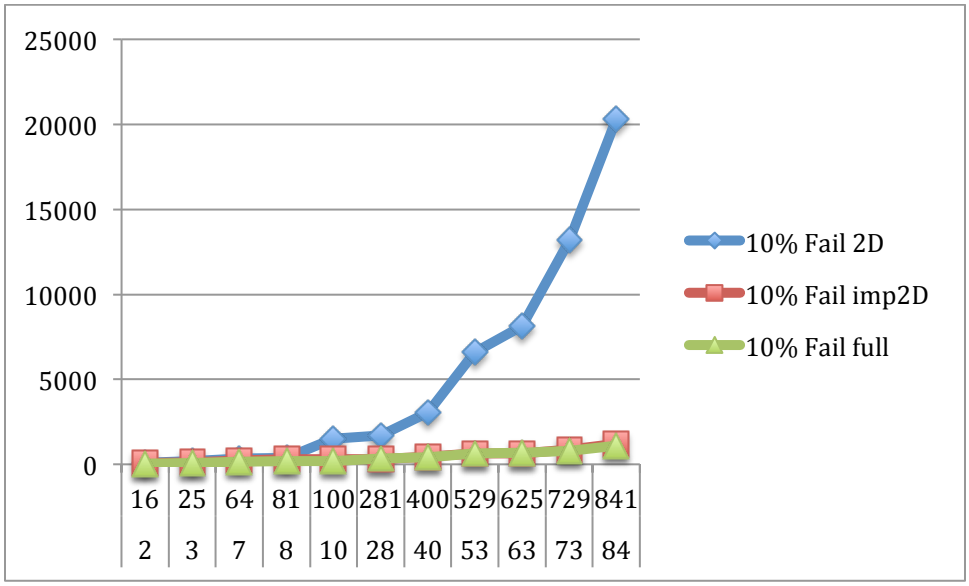
With Failure Model (Bonus):
We ask the user about how many nodes he/she wants to fail and then we randomly generate a list of nodes that are to be shutdown.

80% Failure of nodes



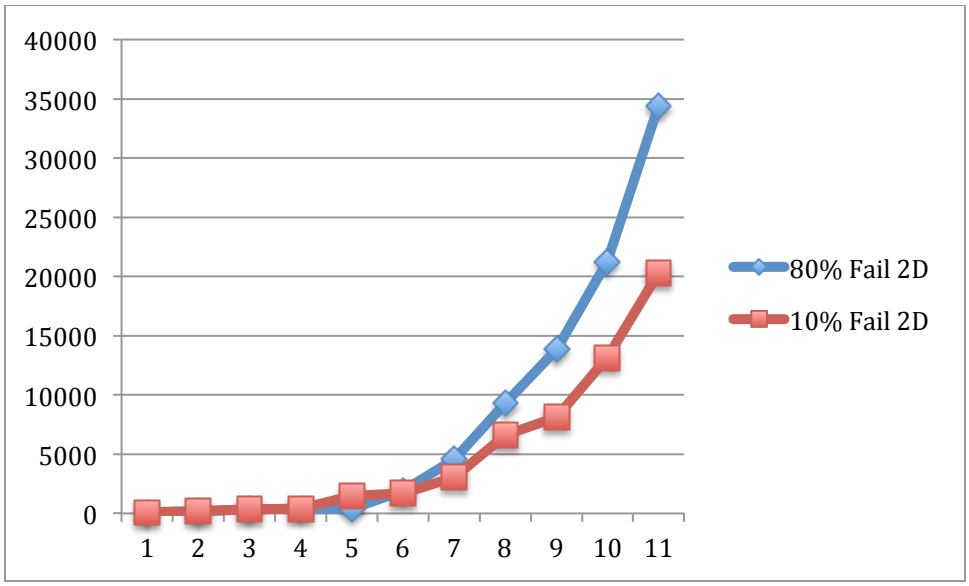
On Y-axis, you see the convergence time of the algorithm and on X-axis you find the total number of nodes.
Here we find a similar observation as normal pushsum, but the interesting finding is that even in case of 80% failure, the pushsum is taking a bit longer than the normal pushsum.

10% Failure of Nodes

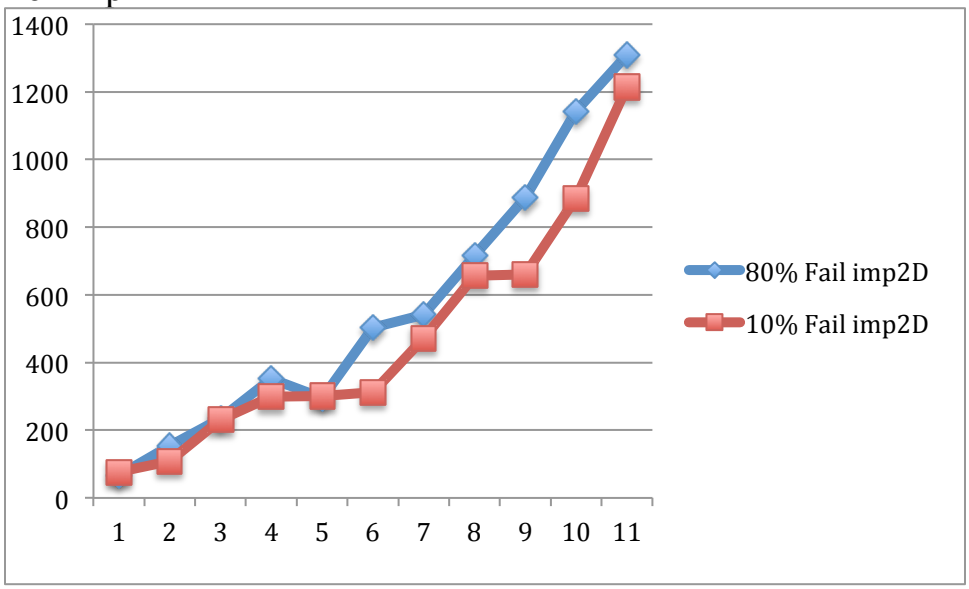


In this case we observe that the algorithm takes lesser time than 80% failure model, mainly because more number of nodes work and converge faster.

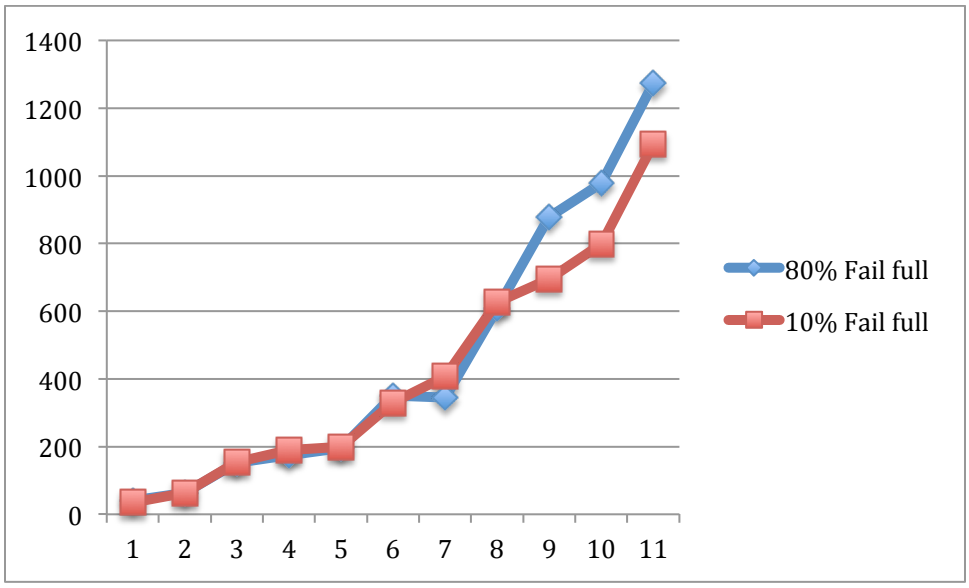
Other graphical comparisons between 80% failure model and 10% failure model
For 2D


40000
35000
30000
25000
20000
15000
10000
5000
0
1 2 3 4 5 6 7 8 9 10 11
80% Fail 2D
10% Fail 2D

For imp2D


1400
1200
1000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11
80% Fail imp2D
10% Fail imp2D

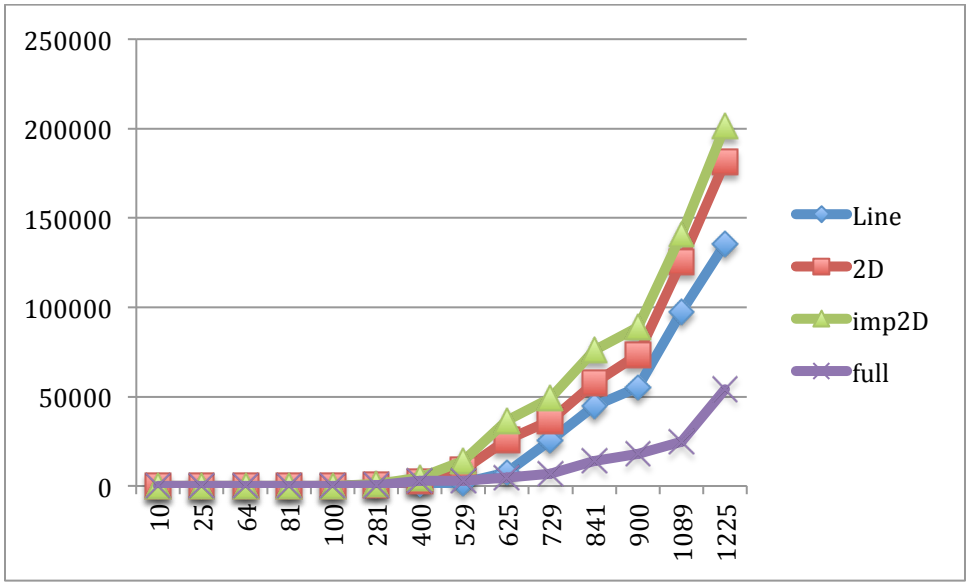
For full



The basic observation is that for less failure, the convergence is faster.

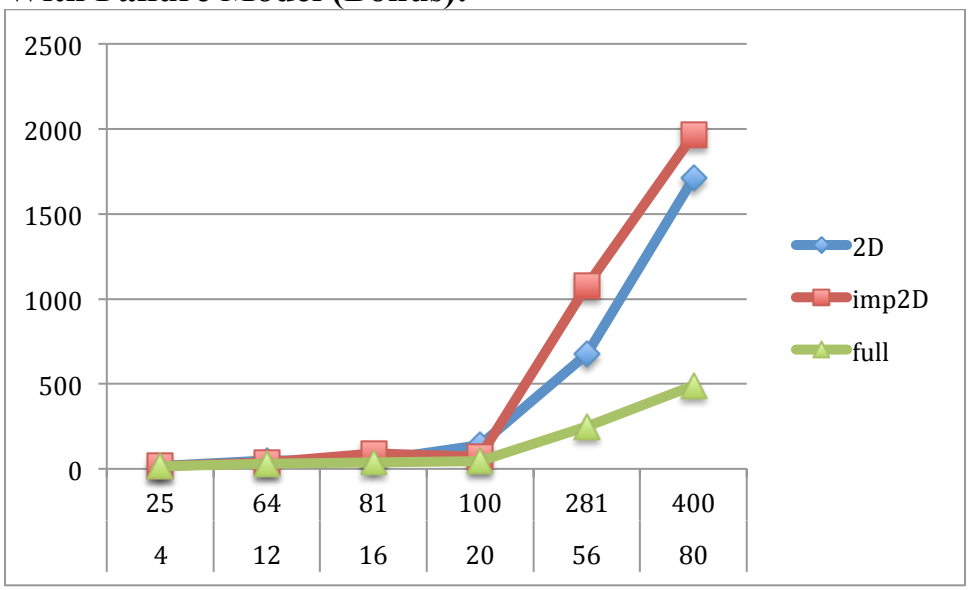
Gossip Graphs

Without Failure Model:



Clearly, full network is the best gossip network as the graph of full network is quite flat and does not rise steeply like other combinations. Another interesting observation is the reverse order of performance between 2D, imp2D and line, the reason I think for this weird observation is that we used lists everywhere, for storing neighbors, visited nodes, nodes etc. If we would've used arrays for storing it would have been easier because we wouldn't have needed to traverse the list every time. So it took a lot of time first to build the topology and then next to traverse the list's. I think that the performance with array's would have been better.

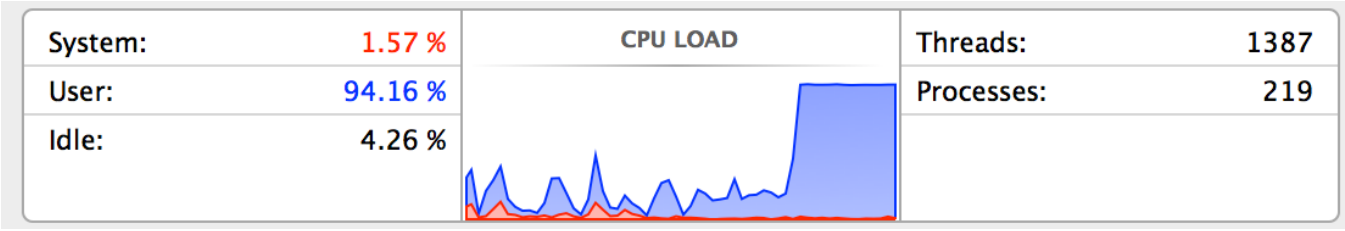
With Failure Model (Bonus):



This is the graph of a 20% failure model.
Again clearly the full network is better, because the graph of full network is the flattest. Next, we again observe the 2D and imp2D. The performance of imp2D should have been better but as I explained above, I think that the List's played a spoilsport. Another quick observation is that the more the number of nodes that fail. The lesser time it takes for the gossip to travel.

Interesting Findings

First of all 94.16% of usage of CPU.



In case of failure model (bonus):

- If the network is a line network, if the node fails, then we need to remove that node and interconnect their neighbors because if that doesn't happen, on part of the network will be disconnected.
- The gossip algorithm takes less time to spread the rumor to all the nodes, if the network is well connected, mainly because the nodes have reduced.
- In case of the 80% failure model and 10% failure model, we see that 80% failure model requires more time to converge in case of push sum.

In without failure case:

- Full network is the best to spread rumor the fastest but has a higher build time as well. Whereas imp2D can give an almost equivalent performance till a certain point and has a lesser build time.
- In implementation of topologies, and assigning neighbors, it would have been easier if we had used an array implementation instead of a List.

Attached Files for details

- **An Excel file for all the graph and exact values of number of Nodes and time.**
- **Screenshot images**