Distributed Systems Project 2
Gossip Simulator

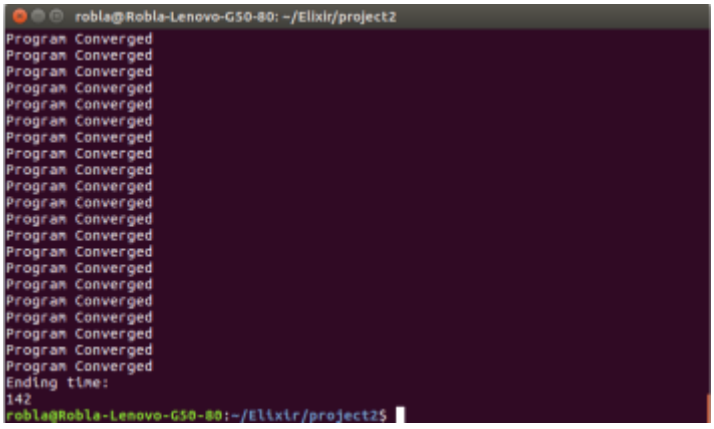
1. Group Details:

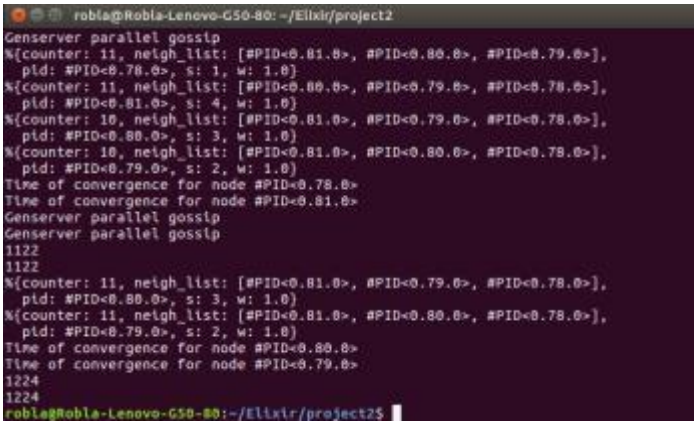
Members

1: Manjary Modi ,UFID: 38408368, manjary.modi@ufl.edu
2: Rameshwari Obla Ravikumar,UFID: 16161302, rameshwari.oblar@ufl.edu

Steps to run the application:

1. In shell, navigate to the directory where mix.exs exists
2. Run command
>mix escript.build
[Ignore any warnings that you notice]
>./project2 <numNodes> <Topology> <Algorithm>
<numNodes> can be any number
<Topology> can be any of full,2D,imp2D,line
<Algorithm> can be any of push-sum or gossip
Example: ./project2 20 full push-sum
After step2, the program will start running based on the topology and algorithm chosen and terminates when the convergence is reached
3. The time taken to achieve convergence is printed on the screen before the program exits.





Application Details:
Our application consists of two ex files – project2.ex and Actor.ex

- project2.ex contains the main module which is used for getting the required inputs from the user, build topologies for the created actors, bind neighbor actors to each actors and call Gossip or Push-Sum algorithm.
- Actor.ex contains the actual implementation for creating the actors, bind neighbors with actors, gossip propagation and push-sum calculation.

[Explanation for each module and method is written in the code]

2. What is working?

Topology Creation:
As soon as the program starts, based on the topology chosen, the main module creates the required number of Actors and binds its neighbor actors with each actor. For 'full' topology, every actor is a neighbor of all other actors, that is, every actor can talk directly to any other actor. For '2D' topology, actors form a 2D grid, that is, the actors can only talk to the grid neighbors. We have implemented simple planar mesh and the network is basically a matrix of nodes, each with connections to its nearest neighbors(diagonal nodes are not considered as neighbors). For 'imperfect 2D' topology, we have selected the same neighbors as that of the 2D grid but randomly selected one non-nearest neighbor and added it to the neighbor list. In 'line' topology, all the actors are arranged in a line. Each actor has only 2 neighbors (one left and one right, unless you are the first or last actor).

GenServer for managing actors:
We have used Genservers in Actor.ex to create the Actor processes, maintain states like process ID, s value,w value, neighbor list of a particular actor, and counter for storing the threshold value.

Gossip Algorithm:
For implementing the gossip algorithm, we tried two different approaches.

- First approach is called 'sequential gossip' where the main process transmits a message to one Actor. This actor picks a random neighbor and transmits the message. The process of gossip continues only when an actor receives a message. When all the actors reaches a threshold value of '11' (arbitrary number), that is, when all the actors have received the message for 11 times, the application terminates. We have collected data for full topology and noticed that this method does not scale well properly and takes longer time to converge as every actor has to wait for someone to send a message for every transmission. So, we resorted to another approach.

Number of Nodes	**Time to Converge (in milliseconds)**
100	1292
200	2924
300	4328
400	5624

500	7536

- Second approach is called 'parallel gossip' (This is our main approach for this project) where the main process transmits a message to one Actor. On receiving a message, every actor picks a random neighbor and transmits the message and without waiting for any other actors ,it selects another random neighbor and transmits the message. Hence,the process of gossip continues in a parallel manner. When all the actors reaches a threshold value of '11' (arbitrary number), that is, when all the actors have received the message for 11 times, the application terminates. The below table shows data for Full Topology and we can observe that the convergence time is way better than the first approach as there are no wait times involved.

Number of Nodes	**Time to converge (in Milliseconds)**
100	1848
200	1891
300	2114
400	2287
500	2701

Note: While executing Gossip algorithm, since it is asynchronous, we have used :timer.sleep for the functionality to work properly. The program does not exit immediately after the termination condition is reached. You need to abort by pressing ctrl+c to come out of the application, if you don't wish to wait till the Process comes out of the sleep time. [Currently, we have given the sleep time as 2 minutes, which will work perfectly for 50k actors.]

Push-sum Calculation:
Each actor A(i) maintains two quantities: s and w. Initially, s =x(i)= i (that is actor number i has value I) and w = 1. This is set in the GenServer init method. The main module starts this algorithm by sending half of the first actor's [s,w] values to a randomly picked neighbor. The neighbor is picked using 'pickNextNodePushSum' method defined in Actor.ex. This random neighbor adds the [s,w] value it receives and sends half of its updated [s,w] value to anther randomly picked neighbor (This is implemented as a Genserver handle_call method). This process continues until the termination condition is reached. If an actors s/w ratio does not change more than 10 ^−10 in 3 consecutive rounds, the actor terminates from the propagation. The program converges when all the actors stop propagating the [s,w] values.

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

GOSSIP ALGORITHM

	Full	2D	Imp2D	Line
Number of Nodes	50000	50000	90000	10000
Time to Converge (In milliseconds)	200762	26460	47082	296149

PUSH-SUM ALGORITHM

	Full	2D	Imp2D	Line
Number of Nodes	10000	10000	10000	1000
Time to Converge (In milliseconds)	682304	52367945	643039	31234234

One thing we noticed in calculating the largest number of nodes for each of the topology is that the application has that ability to converge for larger number of nodes than what is tabulated. It's taking longer time to create actors and build topology due to which we were not able to collect data beyond what is tabulated. Our speculation is that Full,2D and Improper 2D can converge for larger nodes, that is, beyond 100k (we have tested for 90k nodes for imp2D, it took real time of about 2hours for the program to converge). Line would take the greatest amount of time to converge.

Project 2 - Bonus
Gossip Simulator – Fault Tolerant Implementation

Steps to run the application:
1. In shell, navigate to the directory where mix.exs exists
2. Run command
>mix escript.build
[Ignore any warnings that you notice]
>./project2_bonus <numNodes> <Topology> <Algorithm>
<numNodes> can be any number
<Topology> can be any of full,2D,imp2D,line
<Algorithm> can be any of push-sum or gossip
Example: ./project2_bonus 10 full push-sum
After step2, the program will start running based on the topology and algorithm chosen and terminates when the convergence is reached
3. The time taken to achieve convergence is printed on the screen before the program exits.

Application Details:
Our application consists of two ex files – project2_bonus.ex and Actor.ex

- project2_bonus.ex contains the main module which is used for getting the required inputs from the user, build topologies for the created actors, bind neighbor actors to each actors and call Gossip or Push-Sum algorithm with failure-handling code.
- Actor.ex contains the actual implementation for creating the actors, bind neighbors with actors, gossip propagation and push-sum calculation with failure-handling

What is working?

For implementing the failure model, we have killed some process in the middle of the application execution and measured the performance. We made changes in the application in such a way that, our gossip or push sum implementation for each actor gets called only if that particular actor is alive. If no live actor is found, we search for a live actor and continue with the execution. The data collected for benchmarking cannot be compared with that of the data collected in program2. This is because our implementation checks recursively for live actors and loses large amount of time in each iteration. Moreover, the killing of actors has been introduced asynchronously due to which we incorporated :timer.sleep at certain checkpoints. So, in order to have a comparable data, we have collected and tested the performance with zero killed actor(Benchmark) and different percentages of killed actors(say 10%,25%,50%,75%)

Currently we have set the value for number of process to be killed to 5. While testing the application, you need to give number of nodes as greater than 6 for the application to properly converge.

Parameter used for benchmarking

We based our failure model by using number of killed actors as a parameter to measure the performance.
[Data,Plots and Interesting Observations can be found in Report_Bonus.pdf]