**PROJECT REPORT**

**Name**

VEJAY MITUN VENKATACHALAM JAYAGOPAL **UF-ID**: **3106-5997**

MUTHU KUMARAN MANOHARAN **UF-ID: 9718-4490**

**Project details:**

We are implementing random gossip protocol to accomplish two tasks:

1. To transmit rumors across a distributed network of various topologies, and
2. To do a global aggregate calculation i.e. Push-sum across the distributed network.

**Network Topologies implemented:**

• Full Network: Every actor is a neighbor of all other actors. That is, every actor can talk directly to any other actor.

• Line: 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).

• Random 2D Grid: Actors are randomly position at x, y coordinates on a [0-1.0] x [0-1.0] square. Two actors are connected if they are within .1 distance to other actors.

• 3D torus Grid: Actors form a 3D grid. The actors can only talk to the grid neighbors. And, the actors on outer surface are connected to other actors on opposite side, such that degree of each actor is 6.

• Honeycomb: Actors are arranged in form of hexagons. Two actors are connected if they are connected to each other. Each actor has maximum degree 3.

• Honeycomb with a random neighbor: Actors are arranged in form of hexagons (Similar to Honeycomb). The only difference is that every node has one extra connection to a random node in the entire network.

**Implementation of gossip protocol to spread rumor:**

1. We have a GossipNetworkDeployer module which kick starts an actor called GossipNodesMonitor which creates the number of nodes(actors acts as a node) required and connects them based on their topology.
2. There are two types of basic nodes used – one for gossip and another for push-sum. Here the gossip node is used
3. The GossipNodesMonitor starts the initial rumor by selecting the mid node and spreading the rumor to it. This starts the network gossiping and each nodes will keep on gossiping once it has received a gossip. Each nodes will terminate if it receives or hears the same rumor 10 times.
4. When all the nodes in the network has heard the rumor at least once the convergence criteria is reached and we stop the gossip protocol.
5. We measure the difference in the time when the initial gossip was started by the GossipNodesMonitor and the time taken for all the nodes to hear the rumor at least once. We are using System.monotonic\_time(:millisecond) to measure the time difference in milliseconds.

**Implementation of gossip protocol for push sum:**

1. We have a GossipNetworkDeployer module which kick starts an actor called GossipNodesMonitor which creates the number of nodes (actors act as a node) required and connects them based on their topology.
2. There are two types of basic nodes used – one for gossip and another for push-sum. Here the push-sum node is used.
3. The GossipNodesMonitor starts the initial spread of sum by selecting the mid node and sending the {s, w} value which will be both 0. This activates the network.
4. Each node upon a receive of a message, will do the calculation adding it’s own s,w tuple value with the incoming s, w tuple value and retains the half of the sum to its state and randomly selects a neighbor and sends the same retained value to it.
5. When a node s/w ratio doesn’t change more than 10-10 for 3 consecutive rounds, the node will terminate indicating we have reached convergence. Here we have analyzed that once a node reaches this criterion, the average sum approximated is same across all the nodes and denotes the average sum value.
6. We measure the difference in the time when the initial transmit of message was started by the GossipNodesMonitor and the time taken for a node to reach the convergence criteria. We are using System.monotonic\_time(:millisecond) to measure the time difference in milliseconds.

**ANALYSIS:**

**Graphs plotting convergence time vs size of the network for different topologies and**

**Algorithms**

**NOTE :** While Plotting the graph we have taken worst case(Maximum time taken among the 4 recorded readings taken for different number of nodes).

**Gossip Algorithm**

**A close up of a map

Description automatically generated**

**Analysis for rumor spreading gossip protocol:**

1. We ran the gossip protocol for rumor spread for different topology and recorded the reading for nodes from 500 to 10000(readings saved as Readings.xlsx file within the project folder) and plotted the time taken for convergence vs number of nodes as a line graph.
2. We observed that for smaller node values gossip protocol did well for topologies with a greater number of interconnections like full, random honeycomb, torus followed by honeycomb and then line.
3. The random 2D topologies behavior is moderate and random due to its inherent random nature of node interconnections. We observed it to perform better even for a greater number of nodes. This can be attributed to the inherent randomness of the node distribution and connection.
4. But as the number of nodes increased, we observed that the performance of network with major number of interconnections such as full, random honeycomb, 3D torus started to drop and became poor whereas network with limited number of connections such as line topology were increasing gradually and not at an higher rate as the network with many interconnections. This can be attributed to the overhead of sending messages to a randomly selected neighbor from a huge list of neighbors. Also, we observed that in a network with a greater number of interconnections, when a majority of the node has heard the rumor, the time taken for the message to spread to the node which hasn’t heard the rumor increased greatly due to the redundant amount of neighbors. The probability for these nodes to hear the rumor decreased as the spread of rumor across the network increased.
5. The log graph of time taken to convergence vs number of nodes gives more meaningful insight of the asymptotic growth of convergence time over the number of nodes.
6. The logarithmic graph confirms our observations that in a network with a greater number of nodes, line topology, random 2D topology performs than full or random honeycomb topology or 3D torus. Honeycomb topology performed average compared to them.

**Gossip Log Graph**

**A close up of a map

Description automatically generated**

**Push-Sum Algorithm**

**A picture containing text, map

Description automatically generated**

**Analysis for push sum gossip protocol:**

1. We ran the gossip protocol for push sum for different topology and recorded the reading for nodes from 100 to 600 except random 2D for which we took reading from 300 to 600 since for network with nodes less than 300 had a lot of disconnected groups due to the random nature of node placement been done here. We then plotted the time taken for convergence vs number of nodes as a line graph. (Readings are saved as Readings.xlsx file within the project folder)
2. We observed that for higher number of nodes line topology and honeycomb did worse compared to full which did much better here.
3. The best topology for doing aggregate calculation for greater number of nodes were 3D torus, random honeycomb, full and random 2D.
4. Here it is surprising to notice that the topology which did worse in rumor spreading gossip protocol did well in push sum protocol. This can be attributed to the way the protocol is implemented since we are sending a message only when we receive a message. Hence the message transmission is linear. Network with higher number of interconnections benefit much better than network with limited interconnections as the path to transmit message is numerous. This is reflected from our readings.

**Push-Sum Algorithm Log Graph**

**A map of a lot of text

Description automatically generated**

**KEY OBSERVATIONS:**

1. Based on our implementation, the best of both worlds is seen in random 2D protocol, which performs better for a greater number of nodes in both rumors spreading gossip and push sum protocol. 3D torus performs relatively better than full and line in both protocols.
2. Line protocol’s performance is better for larger number of nodes in rumor spreading gossip protocol but worse in push sum.
3. 3D torus, full topology, random honeycomb, is best for larger number of nodes in push sum protocol but performs poor in rumor spreading gossip protocol. Full network being the worse due to the huge interconnection and list of neighbors maintained by each node. We observed that in a network with a greater number of interconnections, when a majority of the node has heard the rumor, the time taken for the message to spread to the node which hasn’t heard the rumor increased greatly due to the redundant amount of neighbors. The probability for these uninformed nodes to hear the rumor decreased as the spread of rumor across the network increased. This explains the poor performance of full network topology and 3D torus in rumor spreading gossip protocol.
4. We can notice that the performance is not always consistent across different topology such as random honeycomb, honeycomb in rumor spreading gossip protocol as it’s a randomized gossip protocol. Sometimes, the best path is chosen via which the message propagates hence we must observe several iterations to come to a meaningful conclusion. When we took the average time taken across these topologies, they all performed almost similar for higher number of nodes and for higher number of iterations.