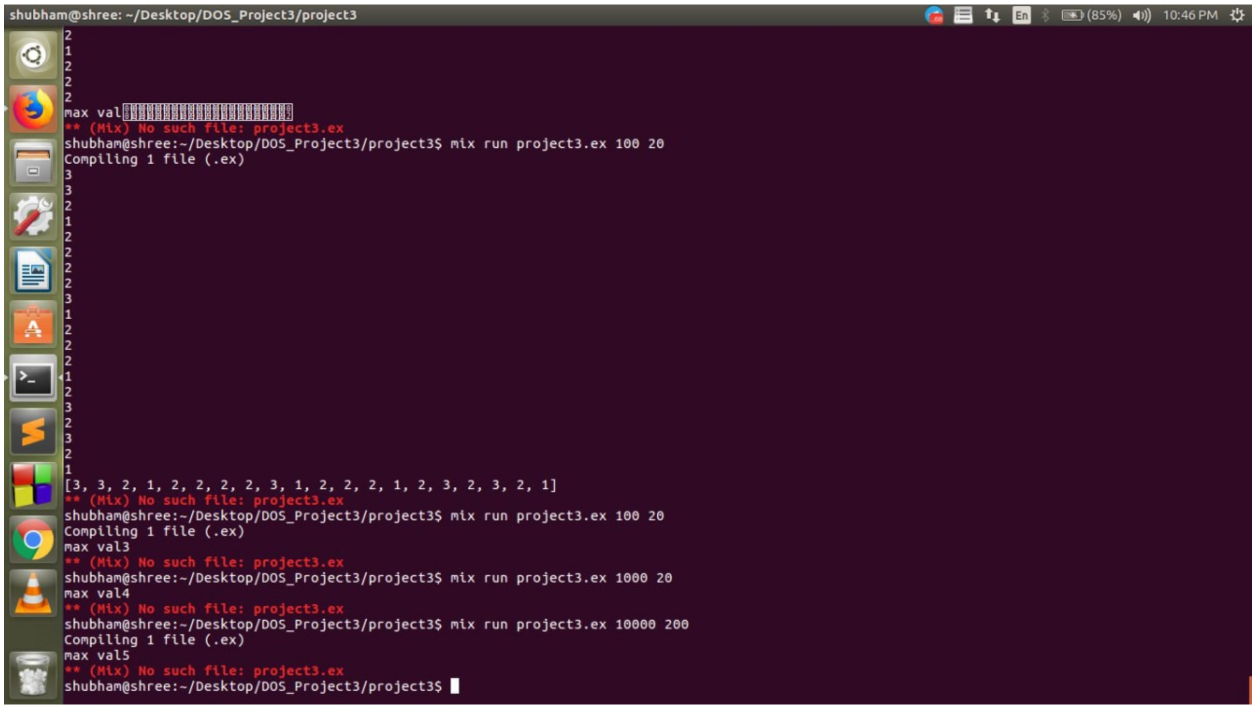
DOS Project 3 Bonus Report

By:
Himavanth Boddu(32451847)
Shubham Saoji(26364957)

Implementation details:
We have implemented network join and routing of **Tapestry Algorithm** using actor model and GenServer in Elixir. We have taken command-line inputs numNodes, numRequests and errorNodePercent. Here numNodes are the total number of nodes that will be present in the network.
Tapestry network implemented for bonus part where fault tolerance of Tapestry algorithm can be observed.



```
shubham@shree: ~/Desktop/DOS_Project3/project3
2
1
2
2
2
max val[illegible]
** (Mix) No such file: project3.ex
shubham@shree:~/Desktop/DOS_Project3/project3$ mix run project3.ex 100 20
Compiling 1 file (.ex)
3
3
2
1
2
2
2
2
3
1
2
2
2
1
2
3
2
3
2
1
[3, 3, 2, 1, 2, 2, 2, 2, 3, 1, 2, 2, 2, 1, 2, 3, 2, 3, 2, 1]
** (Mix) No such file: project3.ex
shubham@shree:~/Desktop/DOS_Project3/project3$ mix run project3.ex 100 20
Compiling 1 file (.ex)
max val3
** (Mix) No such file: project3.ex
shubham@shree:~/Desktop/DOS_Project3/project3$ mix run project3.ex 1000 20
max val4
** (Mix) No such file: project3.ex
shubham@shree:~/Desktop/DOS_Project3/project3$ mix run project3.ex 10000 200
Compiling 1 file (.ex)
max val5
** (Mix) No such file: project3.ex
shubham@shree:~/Desktop/DOS_Project3/project3$ █
```

Above figure to display results for largest tested input(numNodes = 10000, numRequests = 200, max hops = 5), this screen-shot is not for the final version of the project and changes in output display format has been done in the final version.

We have encoded each node number using SHA-1 algorithm which generates 160 bits out of which we use 40 bits(10 characters) and used hex value of it in get_hash(nodeId) function defined by us. We create routing table using ets (The table name for each routing table is an atom in format node id preceded by N and each table is private i.e. read/write function can be performed only by node for which table is created) for each node which contains 10 rows and 16 columns(0-F) where row corresponds to the index of the character matched between source and target and column value corresponds to the first unmatched character node id. If no such match is found then that entry is filled as nil. We create this for 80% of the nodes given in numNodes. For 80% of nodes, routing table is created as soon as process is spawned.

The number of error nodes is calculated as percentage of numNodes and those number of nodes are selected and killed permanently using :kill_node message in handle_info() call.

errorNodes= (errorNodePercent/100)*numNodes

The remaining 20% nodes are added using dynamic node join which computes the routing tables for themselves and send a broadcast message using handle cast to all the nodes except itself so that other nodes are notified about addition of new node in system and they modify/update their routing table accordingly. This is done by comparing the nodeId value with source id and difference is taken, the node with lowest difference is selected while updating the table.

To calculate maximum hops we first select a node as source and numRequest number of other nodes(excluding the source) as destination nodes .We check if the target is active node or error node (by checking using the process-id if it is alive or not). To send request from source to a destination listed in destination list, we first check if the destination node is present in the routing table of the source node. If the final destination node is error node then we update the hop count to 0 for that destination node.

If any intermediate destination node is an error node then we use modify_table_for_inactive_node() function which computer closest node in place of intermediate destination node. In case we find such an intermediate node, we route message to destination through this node. If no such is intermediate node is present, we recompute the route to destination by selecting the closest node by distance to the error intermediate node which is done using get_closest_neighbor().

The function get_closest_neighbor() will compute the nearest possible node by using nearest rows and column value comparison. This will go on until it encounters an active node again(not an error node) and hop count is increased by 1.

If the node is not an error node is present then it will be completed in one hop else it will search for the nearest node to the destination which is available in the routing table using count_hops() function. Whenever a request is forwarded to next node, we increment the hop count in hops_table (an ets table which stores hop count for all numRequest number of destinations). Then we compute hops taken to arrive to the destination node and output the maximum hops taken in this process.

Inference:
In output, we observe no change or a slight change in number of hops for the message to reach the destination when error nodes are introduced compared to model with no error nodes introduced in system. The fault tolerance property of Tapestry algorithm can be observed as message is routed to destination node, might be by a different path(intermediate nodes), requiring more hops but message delivery is ensured.

Since in the case of presence of error nodes we find the nearest possible node to the error node to route forward which may take some extra hops. The resilience of the system is ensured by checking if the message is received by destination node successfully. Success rate in our case is defined by the percentage of requests received by the numRequest number of final destination node successfully. <u>If final destination is error node then we change the final hop count as 0 since it will never find the final destination and keep on searching for it.</u>

Observations:
<u>For input</u>
numNodes = 50
numRequests = 15
errorNodePercentage = 30

<u>Output:</u>
Error list is ["a146d6e3e8", "259d2c3c5b", "2459995b85", "ea9f722a2f", "d7a4a766b8", "8d889e629a", "a0601f5c8d", "340c4889f1", "2b33e1693c", "134c95c6bf", "ba80173f7c", "a370a1c4eb", "1295401b71", "53c0c2973c", "473df39885"]

Source node selected is cf56160616
Destination list is
["103eff4b59", "ea3862ff02", "2b33e1693c", "53c0c2973c", "236b22fc48", "473df39885", "9bfdb05884", "477c95c23d", "28adaf9b00", "8d889e629a", "d7a4a766b8", "ee86906328", "ead86be4ae", "8d61b2828a", "289ceb40a5"]

Destination Node	**Hops**
"103eff4b59"	2
"ea3862ff02"	2
"2b33e1693c"	0
"53c0c2973c"	0
"236b22fc48"	3
"473df39885"	0
"9bfdb05884"	2
"477c95c23d"	1
"28adaf9b00"	2
"8d889e629a"	0
"d7a4a766b8"	0
"ee86906328"	2
"ead86be4ae"	3
"8d61b2828a"	3
"289ceb40a5"	3

Max hops: 3

References:

1. Tapestry: A Resilient Global-Scale Overlay for Service Deployment by Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph and John D. Kubiatowicz. Link to paper- *https://pdos.csail.mit.edu/~strib/docs/tapestry/tapestry_jsac03.pdf*