COP5612 – Fall 2020 / Project 3

TEAM MEMBERS

— Hsiang-Yuan Liao, UFID:4353-5341, hs.liao@ufl.edu
— Tung-Lin Chiang, UFID:9616-8929, t.chiang@ufl.edu

FILES

— Metrics.fs
— program.fs (main file)
— directory file for sharp project (.fsproj)

GOAL

The goal of this project is to implement in F# using the actor model the Pastry protocol and a simple object access service to prove its usefulness. The specification of the Pastry protocol can be found in the paper Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.by A. Rowstron and P. Druschel.

INPUTS

To run the program, we need to provide 2 inputs:

dotnet run numNodes numRequest

- ***numNodes****:* number of nodes
- ***numRequest****:* number of requests

Note that the network stop working when each node has make ***numRequest*** requests.

OUTPUTS

```
[Node94] Lookup done, nodeZ: Node29, hops:2
deliverCount: 98
deliverCount: 99
[Node89] Lookup done, nodeZ: Node16, hops:2
deliverCount: 100




 -------------------- End of Sending Requests --------------------



Averge hops for 100 lookups is 1.680000
hsiang-yuanliao@AlexLiao-MBP pastry % dotnet run 100 10
```

DESIGN OF PASTRY

In a Pastry peer-to-peer overlay network, there is a randomly generated proximity metric, ***proxMetric,*** representing the distance between nodes. We implement the metric using a hash map, and the distance between node ***i*** and ***j*** can be found by the key pair (***i***, ***j***) or (***j***, ***i***). The range of the distances among nodes is controlled by a variable ***maxDistance***. Besides we have a variable ***b*** representing the base for the nodeIDs.

Each node in the network has following properties:

- ***nodeID***: a 128-bit node identifier randomly assigned from the MD5 hashed function
- ***nodeIdx***: a node identifier range from 1 to numNodes, which can access the distance recorded in the ***proxMetric***.
- ***leafSet***:
  - A leaf set contains $|L|/2$ nodes with numerically closest larger and $|L|/2$ nodes with closest smaller nodeIDs
  - $|L| = 2^b = 16,\ if\ b = 4$
- ***neighborSet***
  - A neighbor set contains $|M|$ closest node (by proximity metric).
  - $|M| = 2 \cdot 2^b = 32,\ if\ b = 4$
- ***routingTable***
  - # of rows: $\lceil log_{2^b} N \rceil = \lceil log_{16} N \rceil,\ if\ b = 4$
  - # of a row's entries: $15\ (2^b - 1 = 16 - 1,\ if\ b = 4)$
  - Each entry in the nth row shares a common prefix with the ***nodeID*** in length of n digits, and the next digit ((n+1) th digit) means this entry should be stored in the n+1th column.

The Pastry routing procedure is that if the message key falls within the range of nodeIds covered by the leaf set, the message would be sent to the destination node, whose nodeId is closest to the message key. If the message key is not covered by the leaf set, then the message would be forwarded to a node that shares a common prefix with the key by at least one more digit. In certain cases, the message is forwarded to a node that shares a prefix with the key at least as long as the local node, and is numerically closer to the key than the present node's id.

IMPLEMENTATION

We have two "BOSS" actor, a pastry boss actor and an api boss actor. The pastry boss actor assigns a hashed random ***nodeID*** and ***nodeIdx*** to the new added node and sends a PastryInit messages with ***nodeAIdx***, which is the ***nodeIdx*** of the nearby node A. Then the node A would start routing the ***JOIN*** message util it reaches node Z. Each node has received the ***JOIN*** message would send their state back to the new node with messages of the form of ***InitRoutingTable and InitLeafSet*** for the initialization of the new node's state. Also, these nodes would send ***UpdateStates*** messages to themselves to update their own states.

After the network has built completely, the api boss actor would receive ***SendRequest*** message per 1000 ms (the time is controlled by a variable ***periodicTimer***), and each message makes all the nodes send one request with randomly generated message key. When all the nodes make ***numRequests*** requests, the task is finished.

RESULT

We found that if we would like the Pastry to be strong enough, then we will need more message for update the past network.

We think that it will be better to implement update timestamp cause it is a concurrent network. We have encounter some problem according this. Our lookup message might keep forwarding between a small group of nodes. However, we think we have correctly implement the right algorithm for convergence.

We could even get an average hops of 7.2 for 1000 nodes with 10 requests each without using the routing table. The neighbor table would be the essential use.

#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	1097	25.53	1148	25.67	1098	25.47	1148	25.65	1297	25.63	**1157.6**	**25.59**	25.50	0.09	0.01
100	1250	50.7	1200	50.22	1250	50.63	1250	50.60	1098	50.52	**1209.6**	**50.53**	50.50	0.19	0.03
300	1255	150.16	1192	150.69	1303	150.93	1304	150.84	1204	151.06	**1251.6**	**150.74**	150.50	0.35	0.15
500	1258	251.15	1255	251.11	1159	251.20	1325	250.09	1262	250.27	**1251.8**	**250.76**	250.50	0.54	0.30

#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	2048	23.71	2570	12.64	2447	12.06	2299	18.32	2047	31.95	**2282.2**	**19.74**	25.50	8.31	88
100	2598	28.96	1650	38.95	1997	39.25	1950	32.07	1500	72.04	**1939**	**42.25**	50.50	17.23	306
300	2458	95.17	1999	20.78	2809	14.29	2005	117.32	1948	194.90	**2243.8**	**88.49**	150.50	74.65	8303
500	3463	361.58	2765	472.94	2361	17.71	2354	225.79	2609	418.54	**2710.4**	**299.31**	250.50	182.31	28972

#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	1298	39.11	1966	14.49	1548	28.78	1748	19.19	1599	45.55	**1631.8**	**29.42**	25.50	13.06	152
100	1848	15.16	1832	34.25	1749	32.81	1800	85.02	1649	21.30	**1775.6**	**37.71**	50.50	27.62	774
300	1707	119.08	1808	154.65	1755	83.91	1705	53.66	2003	130.91	**1795.6**	**108.44**	150.50	39.86	3040
500	2020	63.71	1864	348.58	3218	498.85	1759	358.09	2315	322.26	**2235.2**	**318.30**	250.50	158.06	24583

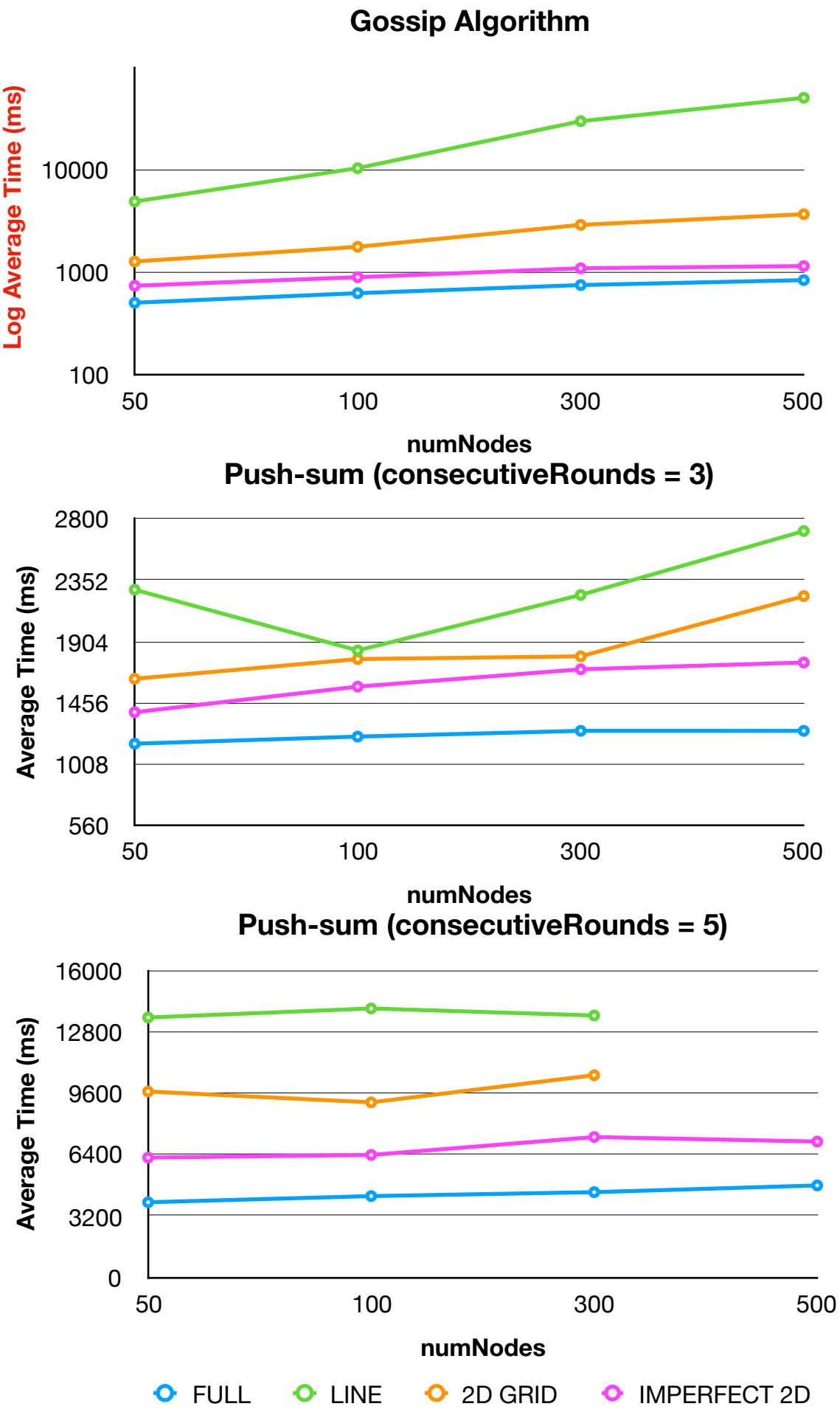
#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	1348	34.74	1247	35.36	1447	36.35	1448	32.33	1448	33.6	**1387.6**	**34.48**	25.50	1.56	83
100	1550	56.8	1548	48.1	1648	50.71	1731	50.0	1395	53.54	**1574.4**	**51.83**	50.50	3.40	11
300	1652	163.61	1645	163.65	1753	149.18	1854	173.55	1600	167.97	**1700.8**	**163.59**	150.50	9.03	237
500	1612	201.56	2014	241.22	1560	252.83	1863	274.87	1704	301.4	**1750.6**	**254.38**	250.50	37.40	1134

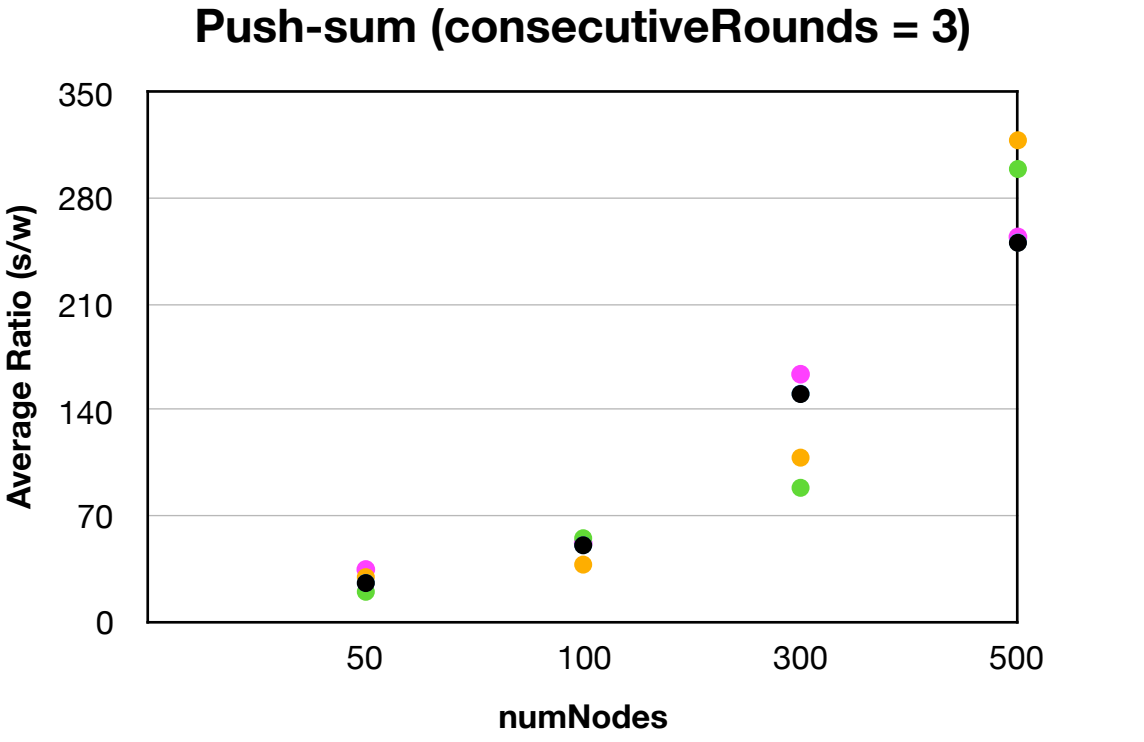
#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	3897	25.49	3877	25.49	3797	25.51	4066	25.49	3857	25.50	3898.8	25.50	25.50	0.01	0.00
100	4302	50.49	4101	50.48	4250	50.53	4103	50.49	4352	50.48	4221.6	50.49	50.50	0.02	0.00
300	4459	150.50	4198	150.48	4545	150.50	4511	150.48	4417	150.52	4426	150.50	150.50	0.02	0.00
500	4648	250.44	4911	250.54	5147	250.50	4527	250.35	4657	250.49	4778	250.46	250.50	0.07	0.01

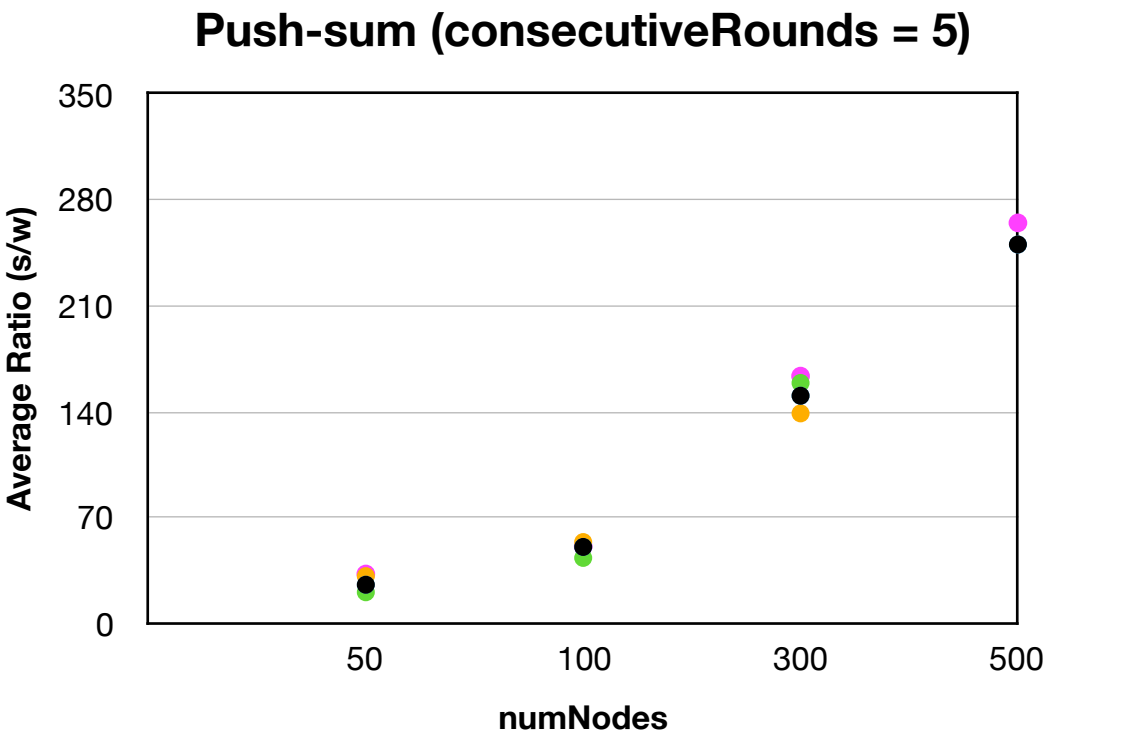
#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	8451	28.46	13714	14.57	10107	9.05	11253	14.88	24152	35.78	13535.4	20.55	25.50	11.12	123
100	11871	15.59	18489	52.87	11149	68.99	13146	53.24	15379	25.64	14006.8	43.27	50.50	21.97	438
300	13628	115.40	13645	168.12	15223	125.82	11712	285.38	13993	99.95	13640.2	158.93	150.50	75.07	4580
500	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	250.50	NA	NA

#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	9903	30.03	9298	39.67	9777	27.58	9201	26.29	10205	32.43	9676.8	31.20	25.50	5.29	55
100	8828	71.66	9242	73.02	8105	40.87	9614	27.35	9758	55.74	9109.4	53.73	50.50	19.74	322
300	10531	214.91	8895	98.52	9810	58.40	12633	101.52	10760	220.84	10525.8	138.84	150.50	74.16	4536
500	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	250.50	NA	NA

#	T1	R1	T2	R2	T3	R3	T4	R4	T5	R5	Avg. T	Avg. R	Ideal	Std. R	MSE
50	6797	33.40	5151	32.43	7254	32.58	6303	32.91	5652	31.85	6231.4	32.63	25.50	0.57	51
100	6652	51.14	7453	50.78	6310	51.16	5810	51.15	5612	49.78	6367.4	50.80	50.50	0.59	0
300	6159	163.34	9268	166.35	7180	162.41	6285	163.79	7610	161.49	7300.4	163.48	150.50	1.83	171
500	7725	263.04	6799	262.79	6866	262.81	7472	267.00	6468	268.37	7066	264.80	250.50	2.68	210


Gossip Algorithm
Log Average Time (ms)
10000
1000
100
50
100
300
500
numNodes
Push-sum (consecutiveRounds = 3)
Average Time (ms)
2800
2352
1904
1456
1008
560
50
100
300
500
numNodes
Push-sum (consecutiveRounds = 5)
Average Time (ms)
16000
12800
9600
6400
3200
0
50
100
300
500
numNodes
FULL
LINE
2D GRID
IMPERFECT 2D





IDEAL FULL LINE 2D GRID IMPERFECT 2D

Gossip Average time (ms): FULL < IMPERFECT 2D < 2D < LINE

Push-sum Average time (ms): FULL < IMPERFECT 2D < 2D < LINE

Std in Average Ratio: FULL < IMPERFECT 2D < 2D < LINE

MSE in Average Ratio: FULL < IMPERFECT 2D < 2D < LINE