Homework 6

CS425/ECE428 Spring 2022

Due: Wednesday, April 27 at 11:59 p.m.

1. Two phase commit and Paxos . . . . 12 points

   In Spanner and similar systems, a combination of two-phase commit (2PC) and Paxos protocols are used. Both the coordinator and participants in 2PC are implemented as *replica groups*, using Paxos to achieve consensus in the group. Each replica group has a leader, so during 2PC, the leader of the coordinator group communicates with the leaders of the participant groups.

   During the execution of 2PC in such a system, there are three points at which a consensus must be achieved within the nodes in a replica group for a transaction to be committed: (i) at each participant group to prepare for a commit, (ii) at the coordinator to decide on a commit after receiving a vote from each participant, and (iii) at each participant again to log the final commit.

   Suppose that there is one coordinator and three participants. Each of these has a Paxos replica group with 5 nodes. The leader of each replica group also acts as the proposer and the distinguished learner for the Paxos protocol, while the remaining four nodes are acceptors. The leaders of the participant and the coordinator replica groups send appropriate messages for 2PC to one another once consensus has been achieved (a decision has been reached) in their respective replica groups. Assume for simplicity that the coordinator replica group only coordinates the transaction and does not participate in processing the transaction (so the coordinator leader need not send prepare and commit messages to itself during 2PC).

   The communication latency between each pair of nodes *within* each group is exactly $10ms$ and the communication latency between any pair of nodes in two different groups is exactly $25ms$. The processing latency at each node is negligible.

   Answer the following questions assuming that there are no failures or lost messages. Further assume that the leader of each replica group has already been elected / pre-configured. All participant groups are willing to commit the transaction, and all nodes within each replica group are completely in sync with one-another.

   (a) (6 points) With this combined 2PC / Paxos protocol,

      (ai) what is the minimum amount of time it would take for each node in the participant group to commit a transaction after the leader of the coordinator group receives the "commit" command from the client? *(3 points)*

      (aii) how many messages are exchanged in the system before all nodes in the participant groups commit the transaction? (Ignore any message that a process may send to itself). *(3 points)*

      [Hint: Think about the message exchanges required by each protocol (2PC and Paxos). Are there messages that can be safely sent in parallel to reduce the commit latency?]

   (b) (2 points) What is the earliest point at which the coordinator group's leader can safely tell the client that the transaction will be successfully committed? Calculate the latency until this point (from the time since the leader of the coordinator group receives the "commit" command from the client).

   (c) (4 points) Suppose we re-configure the system such that the leader of the coordinator group also acts as the leader (proposer and distinguished learner) for the participant Paxos groups. Four nodes in each participant group continue to be acceptors. With this modification, what is the minimum time it takes for each node in the participant group to commit a transaction after the leader of the coordinator group receives the "commit" command from the client?

2. DHT . . . . 16 points

Consider a Chord DHT with a 16-bit address space and the following 100 nodes (hexadecimal values in parentheses).

```
 1127  (467),    2456  (998),    3786  (eca),    4562  (11d2),
 5579  (15cb),   6016  (1780),   6134  (17f6),   6351  (18cf),
 7576  (1d98),   8608  (21a0),   9379  (24a3),   9916  (26bc),
10111  (277f),  10335  (285f),  11967  (2ebf),  12158  (2f7e),
12721  (31b1),  14471  (3887),  15900  (3e1c),  16315  (3fbb),
16419  (4023),  17102  (42ce),  17193  (4329),  17460  (4434),
19257  (4b39),  19857  (4d91),  19963  (4dfb),  20012  (4e2c),
20485  (5005),  20721  (50f1),  21422  (53ae),  22029  (560d),
24052  (5df4),  24335  (5f0f),  25642  (642a),  25963  (656b),
26446  (674e),  26842  (68da),  27477  (6b55),  28481  (6f41),
28926  (70fe),  29112  (71b8),  29408  (72e0),  29548  (736c),
30729  (7809),  31428  (7ac4),  32403  (7e93),  33125  (8165),
33875  (8453),  34871  (8837),  35312  (89f0),  35526  (8ac6),
35600  (8b10),  37641  (9309),  37773  (938d),  41351  (a187),
41463  (a1f7),  42016  (a420),  42200  (a4d8),  42513  (a611),
43590  (aa46),  43934  (ab9e),  43967  (abbf),  45357  (b12d),
46305  (b4e1),  46625  (b621),  46684  (b65c),  47477  (b975),
48441  (bd39),  48679  (be27),  49659  (c1fb),  49844  (c2b4),
50069  (c395),  50135  (c3d7),  50197  (c415),  52086  (cb76),
52325  (cc65),  52368  (cc90),  53171  (cfb3),  53684  (d1b4),
54501  (d4e5),  55037  (d6fd),  55263  (d7df),  56343  (dc17),
56739  (dda3),  57289  (dfc9),  58569  (e4c9),  58640  (e510),
59317  (e7b5),  59453  (e83d),  60596  (ecb4),  60598  (ecb6),
62457  (f3f9),  62794  (f54a),  63816  (f948),  64743  (fce7),
64831  (fd3f),  65010  (fdf2),  65363  (ff53),  65423  (ff8f),
```

For programmatic computations, these numbers have also been made available at:
`https://courses.grainger.illinois.edu/ece428/sp2022/assets/hw/hw6-ids.txt`

(a) (6 points) List the fingers of node 49844.

(b) (6 points) List the nodes that would be encountered on the lookup of the following keys by node 49844:

(i) 12100

(ii) 29200

(c) (4 points) A power outage takes out a few specific nodes: the ones whose numbers are odd. Assume that each node maintains only one successor, and no stabilization algorithm has had a chance to run, so the finger tables have not been updated. When a node in the normal lookup protocol tries to contact a finger entry that is no longer alive (i.e. its attempt to connect with that node fails), it switches to the next best option in its finger table that is alive. List the nodes that would be encountered on the lookup of the key 29200 by node 49844 (include the failed ones).

3. MapReduce...12 points

(a) (6 points) Given four vectors V_1, V_2, V_3 and V_4, each having a dimension of N. Use a map-reduce chain to compute the dot product of $(V_1 + V_2)$ and $(V_3 + V_4)$. The input to the map-reduce chain is in the following key-value format: (k, v), with $k = (i, n)$, where $i \in [1, N]$ is the index of the vector V_n, and v is the corresponding value $(V_n[i])$. The output of your map-reduce chain must of the form (-, final result). Assume there are 50 nodes (servers) in your cluster. Your map-reduce chain must support proper partitioning and load-balancing across these nodes. In particular, assuming a vector dimension of 10000 ensure that a single node is not required to handle more than ≈800 values at any stage. You can assume that, if allowed by your map-reduce semantics, the underlying framework perfectly load-balances how different keys are sent to different nodes.

(b) (6 points) Given a directed graph $G = (V, E)$, use a map-reduce chain to compute the set of vertices that are reachable in *exactly 3 hops* from each vertex. For example, in a graph with vertices $\{a, b, c, d\}$ and the following directed edges, $a \to b \to c \to d$, the vertex d is reachable in exactly three hops from vertex a.

The input to the map-reduce chain is in the following key-value format: (k, v) where k is a graph vertex and v is a list of its out-neighbors; i.e., for each $x \in v, (k, x)$ is a directed edge in E. The output must be key-value pairs (k, v), where k is a graph vertex and v is a list of vertices that are reachable in exactly three hops from k. The list must be empty if there are no vertices reachable in exactly three hops from k. Vertices maybe repeated in the three-hop path and need not be distinct. It is also possible for a vertex to be exactly three hops away from itself, in which case it should be included in the list.

For your assistance, the first map function for an exemplar map-reduce chain has been provided below. You may choose to use the same function, or design your own.

```
function MAP1((k, v)):
    for node in v do
        emit ( ( node , ( “in” , k ) ) )
        emit ( ( k , ( “out” , node ) ) )
    end for
    if v is empty then
        emit ( (k, (“out”, _)))
    end if
end function
```