Homework 5

CS425/ECE428 Spring 2023

Due: Monday, April 24 at 11:59 p.m.

1. Two phase commit and Paxos .... 14 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 two participants. Each of these has a Paxos replica group with 4 nodes. The leader of each replica group also acts as the proposer and the distinguished learner for the Paxos protocol, while the remaining three nodes are acceptors (the leader sends its prepare and accept messages to all three 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 $4ms$ and the communication latency between any pair of nodes in two different groups is exactly $20ms$. 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,

      (i) 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)*

      (ii) 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?]

   Solution:
   After a leader in a participant group initiates Paxos, 15 messages are exchanged: (3 Prepare messages from leader to acceptors + 3 OK messages from acceptors to leader + 3 Accept requests from leader to acceptors + 3 Accepted responses from acceptors to leader + 3 The leader/learner sends "Decided" messages).

   Here, the leader can determine convergence after step 4, and send the coordinator node a message in parallel with the "Decided" message in its participant group.

   The 2PC/Paxos protocol has the following steps:

   1. Prepare messages sent from coordinator leader to the leaders of participant nodes. Time takes: $20ms$ and messages sent: 2.

2. Paxos consensus at the participant groups. Here, the Paxos 4th step of sending "Deciding" message can be sent in parallel with the following step 3. Time taken: $4 \times 4ms$ and messages sent: 15.
3. Send "Yes" to coordinator leader. Time taken: $20ms$ and messages sent: 2.
4. Coordinator group runs consensus using Paxos. Again, the "Deciding" message is sent in parallel to the following step 5. Time taken: $4 \times 4ms$ and messages sent: 15.
5. Coordinator leader sends "commit" response to two participants. Time taken: $20ms$ and messages sent: 2.
6. Participant groups now run Paxos to commit the messages. Time taken: $5 \times 4ms$ and messages sent: 15.

Solution:

(i) $20 + 4 \times 4 + 20 + 4 \times 4 + 20 + 5 \times 4 = 112ms$

(ii) $2 + 15 \times 2 + 2 + 15 + 2 + 15 \times 2 = 81$ messages

(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).

Solution: At step 4 from the previous solution, once the coordinator leader has received "Yes" from both the participants and also run Paxos within the coordinator group and received "Accepted" messages within the replica group, it can safely tell the client that the transaction will be committed. Time: $20 + 4 \times 4 + 20 + 4 \times 4 = 72ms$.

(c) (6 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. Three nodes in each participant group continue to be acceptors. The original leader within each participant replica group simply acts as a learner (and is no longer the leader/proposer/distinguished learner). With this modification:

(i) 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? (3 points)

(ii) 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)*

Solution: Since the leader of the coordinator group is now the leader that is running Paxos for each replica group, steps 1, 3 and 5 from the Q1a can be ignored and the times for rest of the steps need to be adjusted.

Note that since there are 4 learners (including the three acceptors) in the participant replica groups, the coordinator leader will send 4 "Decided" messages instead of 3 to all nodes in the replica group. Thus, the complete Paxos for participant replica groups will contain 16 messages.

(i) Times cost: $4 \times 20 + 4 \times 4 + 5 \times 20 = 196ms$.

(ii) Message cost: $16 \times 2 + 15 + 16 \times 2 = 79$ messages.

2. DHT . . . . 16 points

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

```
    209  (d1),     1759  (6df),     2233  (8b9),     2434  (982),
   2775  (ad7),    4302  (10ce),    4540  (11bc),    4544  (11c0),
   6112  (17e0),   6769  (1a71),    7590  (1da6),    8027  (1f5b),
   8184  (1ff8),  10146  (27a2),   10636  (298c),   10810  (2a3a),
  11147  (2b8b),  11841  (2e41),   12250  (2fda),   13178  (337a),
  13669  (3565),  14157  (374d),   14551  (38d7),   15221  (3b75),
  15250  (3b92),  15392  (3c20),   15470  (3c6e),   16201  (3f49),
  20094  (4e7e),  21557  (5435),   22474  (57ca),   22716  (58bc),
  22820  (5924),  23935  (5d7f),   24147  (5e53),   24317  (5efd),
  24568  (5ff8),  24613  (6025),   25282  (62c2),   25336  (62f8),
  26674  (6832),  26844  (68dc),   29578  (738a),   29696  (7400),
  31216  (79f0),  31398  (7aa6),   31766  (7c16),   32172  (7dac),
  32214  (7dd6),  32507  (7efb),   32608  (7f60),   33101  (814d),
  33584  (8330),  36073  (8ce9),   36176  (8d50),   36470  (8e76),
  38061  (94ad),  38362  (95da),   38700  (972c),   39140  (98e4),
  39905  (9be1),  40363  (9dab),   40804  (9f64),   41615  (a28f),
  44296  (ad08),  44784  (aef0),   45204  (b094),   47061  (b7d5),
  47825  (bad1),  48410  (bd1a),   49032  (bf88),   49466  (c13a),
  49990  (c346),  50922  (c6ea),   51135  (c7bf),   51946  (caea),
  52019  (cb33),  52775  (ce27),   52943  (cecf),   53326  (d04e),
  53576  (d148),  54199  (d3b7),   56746  (ddaa),   57346  (e002),
  57419  (e04b),  57879  (e217),   58167  (e337),   58587  (e4db),
  58897  (e611),  60500  (ec54),   60756  (ed54),   60813  (ed8d),
  62387  (f3b3),  62801  (f551),   63008  (f620),   63489  (f801),
  64822  (fd36),  65065  (fe29),   65198  (feae),   65376  (ff60),
```

For programmatic computations, these numbers have also been made available at:
`https://courses.grainger.illinois.edu/ece428/sp2023/assets/hw/hw5-ids.txt`

(a) (6 points) List the fingers of node 21557.

Solution:

i	ft[i]
0	22474
1	22474
2	22474
3	22474
4	22474
5	22474
6	22474
7	22474
8	22474
9	22474
10	22716
11	23935
12	26674
13	31216
14	38061
15	56746

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

21557:

(i) 10100

(ii) 52500

Solution:

(i) Querying 10100:

(a) 21557
(b) 56746
(c) 8027
(d) 8184
(e) 10146

(ii) Querying 52500:

(a) 21557
(b) 38061
(c) 47061
(d) 51946
(e) 52019
(f) 52775

(c) (4 points) A power outage takes out a few specific nodes: the ones whose identifiers are perfect multiples of 3. 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 52500 by node 21557 (include the failed ones).

Solution:

1. 21557
2. 38061 (failed)
3. 31216
4. 47825
5. 51946
6. 52019
7. 52775

3. MapReduce . . . . 10 points

You are the IT administrator for a large class, whose records are maintained in the following key-value format: (k, v), with k = *studentID* (a 6-digit integer value), and v is a list of tuple (*level*, *assignmentID*, *score*), where *level* is either *UG* (for an undergraduate student) or *G* (for a graduate student), *assignmentID* is a unique identifier for an assignment, and *score* is the student's score in that assignment. Please answer the following questions in pseudo-code.

(a) (2 points) Write a MapReduce chain to compute the average score across all students for each assignment. Your MapReduce chain must be of the form (`map` $\rightarrow$ `reduce`). The input to the `map` tasks are the key-value pairs as specified above (key is *studentID*, and value is a list of tuple (*level*, *assignmentID*, *score*)). The outputs of `reduce` tasks must be key-value pairs with each *assignmentID* as key, and the average score (averaged across all students for that assignment) as the value.

(i) Define the `map` function.

(ii) Define the `reduce` function.

Solution:

Map:

```
// k: studentID
// v: a list of tuples: (level, assignmentID, score)
map1(k, v):
    for t in v:
        emit(t.assignmentID, t.score)
```

Reduce:

```
// k: assignmentID
// v: list of scores
reduce1(k, v):
    sum = 0
    count = 0
    for t in v:
        sum = sum + t
        count = count + 1
    emit(k, sum/count)
```

(writing `sum` or `mean` function should also work.)

(b) (2 points) You now need to compute the average score for each assignment computed across all students in the same level (i.e. the average score across all undergraduate students, and the average score across all graduate students). Your MapReduce chain must be of the form (`map` $\rightarrow$ `reduce`). The input to the `map` tasks are the key-value pairs as specified above (key is *studentID*, and value is a list of tuple (*level*, *assignmentID*, *score*)). The outputs of `reduce` tasks must be key-value pairs with the tuple (*assignmentID*, *level*) as key, and the average score computed across all students in the same level for that assignment as the value.

(i) Define the `map` function.

(ii) Define the `reduce` function.

Solution:

Map:

```
// k: studentID
// v: a list of tuples: (level, assignmentID, score)
map1(k, v):
    for t in v:
        emit((t.assignmentID, t.level), t.score)
```

Reduce:

```
// k: (assignmentID, level)
// v: list of scores
reduce1(k, v):
    sum = 0
    count = 0
    for t in v:
        sum = sum + t
        count = count + 1
    emit(k, sum/count)
```

(writing `sum` or `mean` function should also work.)

(c) (6 points) Assume there are 80 nodes (or servers) in your MapReduce cluster. The class comprises of 200 undergraduate students and 200 graduate students, and there are 10 assignments in total. Modify and extend the MapReduce chain in Q3b to balance the load across the cluster nodes, such that each node handles no more than ≈50 values at any given stage. Also explain how your modified MapReduce chain satisfies the above requirement.

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. The input to the first map function in your chain, and the output of the last reduce function in your chain must be the same as that specified in Q3b.

Solution:
There are total 4000 entries and based on the solution from previous part, the reduce phase receives input of 20 unique `(assignmentID, level)` keys. When distributed equally, each key would have ≈200 entries thus nodes would have to handle ≈200 entries which is more than what they can handle: 50.

So we add an additional key to help distribute the computation:

```
// k: studentID
// v: a list of tuples: (level, assignmentID, score)
map1(k, v):
    r = random(1, 4) // get a random integer key between [1, 4].
    for t in v:
        emit((t.assignmentID, t.level, r), t.score)

// k: (assignmentID, level, r)
// v: list of scores
reduce1(k, v):
    sum = 0
    count = 0
    for t in v:
        sum = sum + t
        count = count + 1
    emit(k, (sum, count))
```

```
// k: (assignmentID, level, r)
// v: (sum, count)
map2(k, v):
    emit((k.assignmentID, k.level), v)

// k: (assignmentID, level)
// v: list of tuples (sum, count)
reduce2(k, v):
    sum = 0
    count = 0
    for t in v:
        sum = sum + t.sum
        count = count + t.count
    emit(k, sum/count)
```