Homework 5

CS425/ECE428 Spring 2025

Due: Wednesday, April 30 at 11:59 p.m.

1. Two phase commit and Paxos ..... 14 points

   In many systems, that require both distributed transactions and replication, 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 four participants. Each of these has a Paxos replica group with three nodes. The leader of each replica group also acts as the proposer and the distinguished learner for the Paxos protocol, while the remaining two nodes are acceptors (the leader sends its prepare and accept messages to both acceptors). The leaders of the participants 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 $5ms$ 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) (5 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? *(2 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?]

   (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) (5 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. Two 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? (2 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)*

(d) (2 points) Suppose the communication latency between each pair of nodes within the same replica group is 5ms. Suppose the communication latency between a pair of nodes in two different groups is given by X (where $X > 5$ms). For what range of X would the protocol in part (c) take *less* time to commit a transaction, when compared to the protocol in part (a)?

2. DHT ..... 14 points

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

```
    30  (1e),    1745  (6d1),    1827  (723),    1865  (749),
  2008  (7d8),    2123  (84b),    2842  (b1a),    3227  (c9b),
  4312  (10d8),   4380  (111c),   4744  (1288),   5420  (152c),
  5676  (162c),   6403  (1903),   6511  (196f),   7261  (1c5d),
  7607  (1db7),   7779  (1e63),   7994  (1f3a),   8921  (22d9),
 11357  (2c5d),  11539  (2d13),  12409  (3079),  12732  (31bc),
 12983  (32b7),  13644  (354c),  13865  (3629),  14938  (3a5a),
 15205  (3b65),  15955  (3e53),  18204  (471c),  21242  (52fa),
 21710  (54ce),  23231  (5abf),  24115  (5e33),  24331  (5f0b),
 24611  (6023),  24619  (602b),  25081  (61f9),  26058  (65ca),
 26158  (662e),  26272  (66a0),  26962  (6952),  28286  (6e7e),
 29019  (715b),  29273  (7259),  29916  (74dc),  30166  (75d6),
 30188  (75ec),  31301  (7a45),  33438  (829e),  34062  (850e),
 34564  (8704),  34688  (8780),  34839  (8817),  35792  (8bd0),
 36224  (8d80),  36292  (8dc4),  36553  (8ec9),  36877  (900d),
 37337  (91d9),  37838  (93ce),  38423  (9617),  38495  (965f),
 39038  (987e),  39703  (9b17),  40137  (9cc9),  41858  (a382),
 42327  (a557),  42913  (a7a1),  45244  (b0bc),  46415  (b54f),
 47809  (bac1),  49045  (bf95),  49612  (c1cc),  49671  (c207),
 50143  (c3df),  50284  (c46c),  50442  (c50a),  51045  (c765),
 51236  (c824),  51411  (c8d3),  51517  (c93d),  52236  (cc0c),
 52371  (cc93),  53259  (d00b),  53896  (d288),  54889  (d669),
 58588  (e4dc),  59014  (e686),  59781  (e985),  60378  (ebda),
 61598  (f09e),  61624  (f0b8),  62055  (f267),  63502  (f80e),
 63786  (f92a),  64306  (fb32),  64357  (fb65),  64754  (fcf2),
```

For programmatic computations, these numbers have also been made available at:
`https://courses.grainger.illinois.edu/ece428/sp2025/assets/hw/hw5-ids.txt`

(a) (4 points) List the fingers of node 29273.

(b) (10 points) List the nodes that would be encountered on the lookup of the following keys by node 29273:

(i) 36227

(ii) 17084

3. MapReduce ..... 6 points

(a) (6 points) NetFlicks, a media-service provider, stores the movies watched by its users using a key-value store. Each key-value (k, v) pair is of the form $k = userid$, and $v = (movieId, genrelist)$, indicating the movie that the user has watched, along with a list of genres that the movie belongs to, given as $(genre[1], genre[2], \ldots)$. A movie can belong to one or more genres (e.g. $genre[1]$ = "action", $genre[2]$ = "comedy", etc). Multiple key-value pairs may exist for the same key (userid), for users who have watched multiple movies on NetFlicks.

Assume you are a NetFlicks engineer who has been asked to use a chain of MapReduce tasks to compute the popularity of each genre, where popularity is measured as the number of (non-unique) users that have watched a movie that belongs to that genre. The first map function of your chain must take as input a key-value pair from the datastore described above where $k = userid$, and $v = (movieId, genrelist)$. The final outputs (of your final reduce functions) are key-value pairs with each $genre$ as key, and the number of non-unique users that have watched a movie belonging to that genre as value (the user count is non-unique, in the sense that if a user watches two different movies for the same genre, it must be counted twice for that genre).

Assume there are 1000 nodes (or servers) in your cluster. Your map-reduce chain must support proper load-balancing across these nodes. In particular, assuming that there at most 10 genres, at most 10 million input keys (userID-movieID pairs), and that a given genre appears in at most 1 million of these userID-movieID pairs, your map-reduce chain should be designed so as to ensure that a single node is not required to handle more than $\approx$10000 values at any stage (summed across all tasks that might run on that node in a given 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.

Also mention, for each stage in your map-reduce chain, how many tasks each node runs, and at most how many values each such task handles.

4. Dominant Resource Fair Scheduling ..... 6 points

(a) (6 points) Consider two cloud jobs, Job 1 and Job 2. Each task of Job 1 requires 3 units of CPU and 50MB RAM. Each task of Job 2 requires 15 units of CPU and 10MB RAM. You need to schedule these jobs on a system with 120 units of CPU and 500MB RAM using the dominant resource fairness criteria. How many tasks for each job will you schedule to achieve dominant resource fairness while maximizing resource utilization (to the best extent possible)?

Note that you cannot schedule fractional tasks. The unfairness introduced due to rounding the number of tasks to integer values can be ignored.