IMPORTANT:

- **Upload this PDF** with your answers to Gradescope by 11:59pm on Thursday Dec 4, 2022.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.
- **You have to use this PDF for all of your answers.**

For your information:
- Graded out of **100** points; 4 questions total

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write-Ahead Logging</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Replication</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Two-Phase Commit</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Distributed Query Plan</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
Question 1: Write-Ahead Logging...............................................[25 points]
Consider a DBMS using write-ahead logging with physical log records with the STEAL and NO-FORCE buffer pool management policy. Assume the DBMS executes a non-fuzzy check-point where all dirty pages are written to disk.

Its transaction recovery log contains log records of the following form:

<txnId, objectId, beforeValue, afterValue>

The log also contains checkpoint, transaction begin, and transaction commit records.

The database contains three objects (i.e., A, B, and C).

The DBMS sees records as in Figure 1 in the WAL on disk after a crash.
Assume the DBMS uses ARIES as described in class to recover from failures.

<table>
<thead>
<tr>
<th>LSN</th>
<th>WAL Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;T1 BEGIN&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;T1, B, 6, 7&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;T1, C, 42, 43&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;T2 BEGIN&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;T2, A, 33, 71&gt;</td>
</tr>
<tr>
<td>6</td>
<td>&lt;T1 COMMIT&gt;</td>
</tr>
<tr>
<td>7</td>
<td>&lt;T2, C, 43, 100&gt;</td>
</tr>
<tr>
<td>8</td>
<td>&lt;T3 BEGIN&gt;</td>
</tr>
<tr>
<td>9</td>
<td>&lt;T3, B, 7, 20&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;T2, C, 100, 67&gt;</td>
</tr>
<tr>
<td>11</td>
<td>&lt;CHECKPOINT&gt;</td>
</tr>
<tr>
<td>12</td>
<td>&lt;T3, B, 20, 42&gt;</td>
</tr>
<tr>
<td>13</td>
<td>&lt;T2, A, 71, 13&gt;</td>
</tr>
<tr>
<td>14</td>
<td>&lt;T2 COMMIT&gt;</td>
</tr>
<tr>
<td>15</td>
<td>&lt;T3, B, 42, 66&gt;</td>
</tr>
</tbody>
</table>

Figure 1: WAL

(a) [6 points] What are the values of A, B, and C in the database stored on disk before the DBMS recovers the state of the database?

- □ A=71, B=6, C=100
- □ A=13, B=66, C=67
- □ A=43, B:7, C=Not possible to determine
- □ A=71, B=42, C=42
- □ A=Not possible to determine, B:20, C=43
- □ A=43, B:20, C=Not possible to determine
- □ A:Not possible to determine, B=Not possible to determine, C:67
- □ A=Not possible to determine, B=20, C:Not possible to determine
- □ A:71, B=Not possible to determine C=42

Question 1 continues...
A,B,C: Not possible to determine

(b) [3 points] What should be the correct action on T1 when recovering the database from WAL?
- redo all of T1’s changes
- undo all of T1’s changes
- do nothing to T1

(c) [3 points] What should be the correct action on T2 when recovering the database from WAL?
- redo all of T2’s changes
- undo all of T2’s changes
- do nothing to T2

(d) [3 points] What should be the correct action on T3 when recovering the database from WAL?
- redo all of T3’s changes
- undo all of T3’s changes
- do nothing to T3

(e) [10 points] Assume that the DBMS flushes all dirty pages when the recovery process finishes. What are the values of A, B, and C after the DBMS recovers the state of the database from the WAL in Figure 1?
- A=33, B=6, C=42
- A=13, B=66, C=67
- A=13, B=6, C=100
- A=13, B=7, C=67
- A=71, B=20, C=42
- A=33, B=42, C=100
- A=71, B=7, C=100
- A=13, B=42, C=67
- A=33, B=20, C=43
- A=71, B=66, C=43
- A=13, B=42, C=42
- Not possible to determine

Homework #5 continues...
Question 2: Replication ............................................. [20 points]
Consider a DBMS using active-passive, master-replica replication with multi-versioned concurrency control. All read-write transactions go to the master node (NODE A), while read-only transactions are routed to the replica (NODE B). You can assume that the DBMS has “instant” fail-over and master elections. That is, there is no time gap between when the master goes down and when the replica gets promoted as the new master. For example, if NODE A goes down at timestamp 1 then NODE B will be elected the new master at 2.

The database has a single table foo(id, val) with the following tuples:

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>xx</td>
</tr>
<tr>
<td>2</td>
<td>yy</td>
</tr>
<tr>
<td>3</td>
<td>zz</td>
</tr>
</tbody>
</table>

Table 1: foo(id, val)

For each questions listed below, assume that the following transactions shown in Figure 2 are executing in the DBMS: (1) Transaction #1 on NODE A and (2) Transaction #2 on NODE B. You can assume that the timestamps for each operation is the real physical time of when it was invoked at the DBMS and that the clocks on both nodes are perfectly synchronized.

(a) Transaction #1 – NODE A

1. BEGIN;
2. UPDATE foo SET val = 'x';
3. UPDATE foo SET val = 'yyy' WHERE id = 3;
4. UPDATE foo SET val = 'z' WHERE id = 1;
5. COMMIT;

(b) Transaction #2 – NODE B

2. BEGIN READ ONLY;
3. SELECT val FROM foo WHERE id = 3;
4. SELECT val FROM foo WHERE id = 1;
5. SELECT val FROM foo WHERE id = 1;
6. COMMIT;

(a) Assume that the DBMS is using asynchronous replication with continuous log streaming (i.e., the master node sends log records to the replica in the background after the transaction executes them). Suppose that NODE A crashes at timestamp 5 before it executes the COMMIT operation.

i. [6 points] If Transaction #2 is running under READ COMMITTED, what is the return result of the val attribute for its SELECT query at timestamp 5? Select all that are possible.
- zz
- x
- yy
- yyy
- z

Question 2 continues…
ii. **[7 points]** If Transaction #2 is running under the READ UNCOMMITTED isolation level, what is the return result of the `val` attribute for its `SELECT` query at timestamp 5? Select all that are possible.

- □ zz
- □ x
- □ yy
- □ yyy
- □ z
- □ xx
- □ None of the above

(b) **[7 points]** Assume that the DBMS is using *synchronous* replication with *on commit* propagation. Suppose that both NODE A and NODE B crash at exactly the same time at timestamp 6 after executing Transaction #1’s COMMIT operation. You can assume that the application was notified that the Transaction #1 was committed successfully.

After the crash, you find that NODE A had a major hardware failure and cannot boot. NODE B is able to recover and is elected the new master.

What are the values of the tuples in the database when the system comes back online? Select all that are possible.

- □ { (1,xx), (2,yy), (3,zz) }
- □ { (1,x), (2,x), (3,x) }
- □ { (1,x), (2,x), (3,yyy) }
- □ { (1,z), (2,x), (3,yyy) }
- □ { (1,xx), (2,x), (3,zz) }
- □ { (1,xx), (2,x), (3,x) }
- □ None of the above

Homework #5 continues...
Question 3: Two-Phase Commit...........................................[25 points]
Consider a distributed transaction $T$ operating under the two-phase commit protocol with the early acknowledgement optimization. Let $N_0$ be the coordinator node, and $N_1$, $N_2$, $N_3$ be the participant nodes.

The following messages have been sent:

<table>
<thead>
<tr>
<th>time</th>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$N_0$ to $N_1$: “Phase1:PREPARE”</td>
</tr>
<tr>
<td>2</td>
<td>$N_2$ to $N_0$: “OK”</td>
</tr>
<tr>
<td>3</td>
<td>$N_0$ to $N_2$: “Phase1:PREPARE”</td>
</tr>
<tr>
<td>4</td>
<td>$N_0$ to $N_3$: “Phase1:PREPARE”</td>
</tr>
</tbody>
</table>

Figure 3: Two-Phase Commit messages for transaction $T$

(a) [7 points] Who should send a message next at time 5 in Figure 3? Select all the possible answers.

- $N_0$
- $N_1$
- $N_2$
- $N_3$
- It is not possible to determine

(b) [6 points] To whom? Again, select all the possible answers.

- $N_0$
- $N_1$
- $N_2$
- $N_3$
- It is not possible to determine

(c) [6 points] Suppose that $N_0$ received the “ABORT” response from $N_1$ at time 5 in Figure 3. What should happen under the two-phase commit protocol in this scenario?

- $N_0$ resends “Phase1:PREPARE” to $N_2$
- $N_1$ resends “OK” to $N_0$
- $N_0$ sends “Phase2:COMMIT” all of the participant nodes
- $N_0$ sends “ABORT” all of the participant nodes
- $N_0$ resends “Phase1:PREPARE” to all of the participant nodes
- It is not possible to determine

(d) [6 points] Suppose that $N_0$ successfully receives all of the “OK” messages from the participants from the first phase. It then sends the “Phase2:COMMIT” message to all of the participants but $N_1$ and $N_3$ crash before they receives this message. What is the status of

---

Question 3 continues...
the transaction $T$ when $N_1$ comes back on-line?
- $T$’s status is aborted
- $T$’s status is committed
- It is not possible to determine
Question 4: Distributed Query Plan ........................ [30 points]

The CMUDB group is working on a brand new shared-nothing distributed database system called BusTub*. They are developing the distributed query engine.

Given the following schema:

```
CREATE TABLE t1(PARTITION KEY v1 int, v2 int);
CREATE TABLE t2(PARTITION KEY v3 int, v4 int);
CREATE TABLE t3(PARTITION KEY v5 int, v6 int);
CREATE TABLE t4(PARTITION KEY v7 int, v8 int);
```

The database system partitions the tables by key range. That is to say, each node in the system manages rows of the table within a non-overlapping range of keys.

Given the following query:

```
SELECT * FROM ((t1 INNER JOIN t2 ON v1 = v3)
INNER JOIN t3 ON v3 = v5)
INNER JOIN t4 ON v6 = v7;
```

(a) [5 points] Assume that the query optimizer doesn’t know the ranges of data stored on each node and only knows the tables are sharded by some keys, what are the correct distributed query plans for this query?

□ A)

□ B)

Question 4 continues...
Question 4 continues...
(b) [5 points] Assume there are 3 nodes in the system and the data ranges in each node are as follows:
<table>
<thead>
<tr>
<th>Node 1</th>
<th>0 - 999</th>
<th>0 - 999</th>
<th>1000 - 1999</th>
<th>0 - 999</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 2</td>
<td>1000 - 1999</td>
<td>1000 - 1999</td>
<td>0 - 999</td>
<td>1000 - 1999</td>
</tr>
</tbody>
</table>

Table 2: Data distribution for table $t_1$ to $t_4$

Assume the data are shuffled by range. Which is the best and correct schedule for the query?

- **A)**
  1. HashJoin $v_1 = v_3$: 0-999 on node 1, 1000-1999 on node 3, 2000-2999 on node 2
  2. HashJoin $v_3 = v_5$: 0-999 on node 1, 1000-1999 on node 2, 2000-2999 on node 3
  3. HashJoin $v_6 = v_7$: 0-999 on node 1, 1000-1999 on node 2, 2000-2999 on node 3
  4. Union: on node 1

- **B)**
  1. HashJoin $v_1 = v_3$: 0-999 on node 1, 1000-1999 on node 2, 2000-2999 on node 3
  2. HashJoin $v_3 = v_5$: 0-999 on node 3, 1000-1999 on node 2, 2000-2999 on node 1
  3. HashJoin $v_6 = v_7$: 0-999 on node 1, 1000-1999 on node 2, 2000-2999 on node 3
  4. Union: on node 1, 2, 3

- **C)**
  1. HashJoin $v_1 = v_3$: 0-999 on node 1, 1000-1999 on node 2, 2000-2999 on node 3
  2. HashJoin $v_3 = v_5$: 0-999 on node 1, 1000-1999 on node 2, 2000-2999 on node 3
  3. HashJoin $v_6 = v_7$: 0-999 on node 1, 1000-1999 on node 2, 2000-2999 on node 3
  4. Union: on node 1

(c) **[5 points]** Given the following assumptions:
1. $t_1$ contains 3000 rows and $v_1$ has all values across (0-2999)
2. $t_2$ contains 3000 rows and $v_3$ has all values across (0-2999)
3. $t_3$ contains 3000 rows, $v_5$ has all values across (0-2999) and so as $v_6$
4. $t_4$ contains 300 rows and $v_7$ values are uniformly distributed across 0-2999
5. All joins produce a number of rows equal to $\min\{\text{cardinality of left child, cardinality of right child}\}$.

Question 4 continues...
Using the data distribution of Table 2, how much data is expected to be transferred over the network when executing the plan with the best schedule in question (b)? (Hint: Calculate the answer by summing up all the expected (rows * columns) that would be shuffled before each HashJoin and Union.

- 0 – 5000
- 5000 – 10000
- 15000 – 22000
- 22000 – 40000
- ≥ 40000

(d) [5 points] An engineer realized that t4 is very small so they configured the table to be stored on all nodes. Which of the following plans is correct?

- A)

- B)
Question 4 continues...
(e) **[4 points]** In the correct case of question (d), how much data is expected to be transferred over the network, given the same assumptions in question (c)? *(Hint: Calculate the answer by summing up all the expected (rows * columns) that would be shuffled before each HashJoin and Union.)*

- 0 – 5000
- 5000 – 10000
- 15000 – 22000
- 22000 – 40000
- ≥ 40000

(f) **[6 points]** The BusTub* developers decide to replicate data using multiple groups of Multi-Paxos to ensure high availability. Here is the latest setup of the database (Range = the range of the partition key of a table):

<table>
<thead>
<tr>
<th>Range / Table</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 999</td>
<td>Paxos Group 1</td>
<td>Paxos Group 2</td>
<td>Paxos Group 3</td>
<td>Paxos Group 10</td>
</tr>
<tr>
<td>1000 - 1999</td>
<td>Paxos Group 4</td>
<td>Paxos Group 5</td>
<td>Paxos Group 6</td>
<td>Paxos Group 10</td>
</tr>
<tr>
<td>2000 - 2999</td>
<td>Paxos Group 7</td>
<td>Paxos Group 8</td>
<td>Paxos Group 9</td>
<td>Paxos Group 10</td>
</tr>
</tbody>
</table>

Table 3: Paxos Group IDs

Assuming the following status of the database (L = Leader, A = Acceptor):

What’s the maximum number of nodes that can go down before this query cannot be successfully executed? (Assume reads / writes all go through the leader).

Question 4 continues…
<table>
<thead>
<tr>
<th>Node / Paxos Group</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 1</td>
<td>L</td>
<td></td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 2</td>
<td>A</td>
<td>L</td>
<td></td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 3</td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 4</td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 5</td>
<td></td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 6</td>
<td></td>
<td></td>
<td>A</td>
<td>A</td>
<td>L</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Current Leader / Acceptor nodes of Paxos groups

☐ 1
☐ 2
☐ 3
☐ 4
☐ 5

End of Homework #5