Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.814/6.830 Database Systems: Fall 2015 Quiz II

There are 15 questions and 12 pages in this quiz booklet. To receive credit for a question, answer it according to the instructions given. *You can receive partial credit on questions.* You have **80 minutes** to answer the questions.

Write your name on this cover sheet AND at the bottom of each page of this booklet.
Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.
YOU MAY USE A LAPTOP OR CALCULATOR.
YOU MAY NOT ACCESS THE INTERNET.

Do not write in the boxes below

1-5 (xx/29)	6-7 (xx/17)	8-10 (xx/18)	11-15 (xx/36)	Total (xx/100)

Name: Solutions
(Avg = 82, Median = 84, Std. Dev = 11)

I Main Memory Analytics

1. [8 points]: Which of the following are reasons why query compilation (as described in Lecture 12, and the assigned paper "Generating code for holistic query evaluation") can dramatically improve performance versus a tuple-at-a-time iterator model?

(Circle True or False for each item below.)

A. True / False It avoids processor pipeline flushes by eliminating branches.

Answer. True. Branches (jumps) due to function calls can make control flow hard to predict for the CPU.

B. True / False It eliminates additional instructions associated with function calls.

Answer. True. Function calls imply extra jump as well as register save instructions compared to inlined code.

C. True / False It improves opportunities for pipelined operator execution on multiple cores or processors.

Answer. False. Query compilation has little to do with parallel execution on multiple cores or processors. In case it threw you off, the word pipelined here refers to data pipelining across database operators, not with the processor pipeline.

D. True / False It allows more data to fit in memory, since data is compiled rather than interpreted.

Answer. False. Query compilation does not directly affect data layout or data size. Also, compilation systems compile queries not data (whatever that would mean).

II ARIES Recovery

Consider a DBMS running the strict two-phase locking and the ARIES recovery algorithm as described in the paper by C. Mohan. Below we give the state of a transaction table and a dirty page in the memory of the database immediately after the *analysis* phase of the ARIES protocol completes.

Dirty Page Table

Page	RecLSN
A	6
B	5

Transaction Table

LastLSN	TID
6	2
8	3

You are given that there are 3 transactions in the system, 1, 2, and 3, and 2 data pages, A and B.

Your job is to fill in the blank entries of the log below with values that are consistent with an execution of the ARIES logging protocol. There are log record (operation) types, S (start), W (write), CP (checkpoint), and C (commit), and you are told that no transactions aborted. The system crashed after LSN 8, and no transactions were running prior to LSN 0.

Name:

2. [9 points]: Given the partial state of the log shown below, fill in the blank entries with values consistent with the tables given above.

LSN	0	1	2	3	4	5	6	7	8	
Xaction	1		1	2		2		-		CRASH!
Op/Data	S		WB	WA				CP		

Answer. Transaction 2 had to log its start, which implies (LSN 1, Xaction 2, Op Start).

The dirty page table implies position 5 is WA and position 6 is WB. This implies entry (LSN 5, Xaction 2, Op WB).

The transaction table implies the WB in position 6 was done by TID 2. This implies entry (LSN 6, Xaction 2, Op WA).

The transaction table implies the entry in position 8 was by TID 3. The lack of information about TID 1 in the transaction table implies it committed before LSN 6. The only option now is LSN 4. So we have (LSN 4, Xaction 1, Op Commit).

Finally, the first entry by Xaction 3 must be a start. So, the solution is unique:

(LSN 1, Xaction 2, Op Start)
(LSN 4, Xaction 1, Op Commit)
(LSN 5, Xaction 2, Op WB)
(LSN 6, Xaction 2, Op WA)
(LNS 8, Xaction 3, Op Start)

3. [4 points]: At what LSN did the analysis phase begin?

Answer. It begins at LSN 7 if you include the checkpoint LSN or 8 if you don't. We accepted both answers.

4. [4 points]: At what LSN will the REDO phase begin?

Answer. Redo starts at the minimum RecLSN in the dirty page table (LSN 5).

5. [4 points]: Which of the following are possible values of page B on the disk of the database after the analysis phase finishes?

Name:

(Circle 'Yes' or 'No' for each choice.)

A. Yes / No Whatever its state was prior to the start of the log

Answer. No. The dirty page table implies Page B has already written all modifications with LSN < 5. ARIES correctness depends on this: if the contents of the Page B could be whatever B was at the start of the log, then recovery would need to start at LSN 0.

B. Yes / No Whatever was written by LSN 2

Answer. Yes. It could be this LSN was indeed the last thing written to page B.

C. Yes / No Whatever was written by LSN 5.

Answer. Yes. It is also possible that the version of B as of LSN 5 made it to disk at some point, even though the recovered dirty page table does not yet say so. The dirty page table may lag behind the real contents of disk.

D. Yes / No Something else

Answer. No. There are no other log records that write to B in the log.

Name:

III Locking

Ben Bitdiddle is designing a new transactional database system, "LocksDB". He has the T-shirts ready and the website is up and running. All he needs to do now is design and implement the system. That's where you come in.

LocksDB implements a custom lock-based concurrency control mechanism. This mechanism extends the read/write locking you saw in class to include increment operations. The core idea of LocksDB is that while reads and writes are able to express increment (or decrement) operations, one can improve concurrency by treating increments differently than arbitrary writes. *The key observation is that the state of the database is the same regardless of the order in which two concurrent increments run.* To get an idea of what how transactions with increments work, we provide an example below.

In this listing, `RX` stands for a Read of variable `X`, `WX` stands for a Write of some value to variable `X`, and `IX` means to increment `X` by some value. As in the model of writes presented in class, values used in increments may depend on values previously read. Assume numbers do not overflow and that individual increments happen atomically with respect to other increments/writes.

T1	T2
`IX,5`	
`COMMIT`	
	`local_x = RX`
	`WX,local_x + 5`
	`COMMIT`

Figure 1: Both T1 and T2 are doing the same operation on the data, but T1 uses the increment operation.

6. [9 points]: For LocksDB, the definition of conflicting operations needs to be extended to include increment operations. Recall that conflicting operations are those where the state of the system depends on the order in which a pair of operations are executed.

In LocksDB, as in the two-phase locking protocol we studied in class, before a transaction can perform a particular R/W/I operation, it needs to hold the appropriate lock, and locks for conflicting operation types cannot be held simultaneously by two transactions. LocksDB has three types of locks: read-only (R), exclusive (X), and increment (I). R and X locks behave as in a conventional 2PL system. I-locks are new.

Fill in the following *lock compatibility table*, putting a Y if T1 and T2 should be allowed to concurrently hold the indicated lock types on the same data item (i.e., the operations don't conflict), and N otherwise.

(Write 'Y' or 'N' in each cell of the table.)

	T1 R	T1 X	T1 I
T2 R	Y	N	N
T2 X	N	N	N
T2 I	N	N	Y

Name:

Answer. Increment locks aren't compatible with read or write locks, because performing these operations in different orders will cause a different outcome.

7. [8 points]: Consider the four transaction schedules shown below. For each schedule, if it is conflict-serializable (using the definition of conflicts in the previous question), then write an equivalent serial schedule. If it is not, state why not.

T1	T2	Conflict serializable or not?

`IX,5`
 	
`IY,5`
`COMMIT`
	
	`IY,5`
	
	`IZ,10`
	`COMMIT`
		Yes, T1/T2 or T2/T1

`x1 = RX`
`WX,x1+5`
 	
`y1=RY`
 	
`WY,y1+5`
`COMMIT`
	
	`y2= RY`
	
	`WY,y2+5`
	
	`z2=RZ`
	`WZ,z2 + 10`
	`COMMIT`
		No, T1 RY precedes T2 WY, and
		T2 WY precedes T1 WY

`RX`
 	
`IX,5`
 	
`COMMIT`
	
	`RX`
	
	`IX,5`
	
	`COMMIT`
		No, T1 RX precedes T2 IX,
		and T2 RX Precedes T1 IX

`IX,5`
 	
`tmp1x=RX`
`WY,tmp1x`
`COMMIT`
	
	`IX,5`
	
	`COMMIT`
		Yes, T2/T1

Name:

IV 2PC

Consider the two-phase commit (2PC) protocol, running on a single coordinator C and subordinates S1 and S2. Suppose that, unlike the 2PC variant discussed in class, where the network could lose messages, the network between C, S1, and S2 will never lose or reorder messages.

8. [4 points]:

Dana Bass claims that the addition of a reliable network means that the subordinates no longer need to log a commit record to disk, since if the coordinator decides to commit, they will definitely receive the commit message, due to the reliable network. Is she right? Why or why not?

(Write your answer in the space below.)

Answer. No, she's wrong. If the subordinate crashes and recovers, and the coordinator has already committed and forgotten the transaction, the subordinate will come back, see no commit record, ask the coordinator for the outcome, and receive an abort.

In an alternative design, Dana decides to modify her database system to provide a variant two-phase commit protocol that she calls *eventually consistent 2PC* (EC2PC). Her modification to the basic 2PC protocol is simple: *if a subordinate S votes YES for a transaction T then it follows the following protocol*:

- S writes the prepare log record for T
- S releases all locks
- S sends its vote back to the coordinator

This allows other transactions on S to read / write the records that T modified before S learns the outcome of T, increasing concurrency. The protocol is in other respects identical to 2PC: i.e., S waits to hear the outcome of T from the coordinator, and if the outcome is an abort, performs an UNDO, rolling back any records T modified, so that they are reset to their contents prior to T executing.

In the abort case, Dana notes that concurrent transactions may have modified some of the records that T modified while S is waiting to hear the outcome of T. To address this, Dana employs cascading aborts. Specifically, *if some transaction is holding a lock on an object that is being rolled back, that transaction is also aborted and rolled back.*

9. [8 points]: To measure the best-case performance gain of EC2PC, Dana runs many identical transactions that update the same data item D1 on subordinate S1, and the same data item D2 on subordinate S2. Estimate the *maximum* transaction throughput of EC2PC and conventional 2PC (without presumed commit). Your answer should be expressed in terms of transactions per second of the above form that the system could run. To estimate maximum throughput, assume there is an infinite supply of transactions waiting to run as soon as they can acquire the locks they need to proceed. Assume forced log write and one-way network latency are both 10 ms. Also, there are no failures, aborts, or deadlocks, non-forced writes and CPU operations take no time, and the first forced log writes done in processing these transactions are those in the commit protocol.

(Write your estimates in the spaces below.)

Answer. We gave credit for many different answers here as the question is poorly specified. The intended way to model the problem was that each subordinate would have a queue of transactions that have completed and been sent by the coordinator to prepare, and that each transaction has a separate coordiantor running on its behalf in a separate thread. This would mean that in EC2PC, subordinates could start processing the next transaction as soon as the previous one wrote its prepare. In contrast, in conventional 2PC, the next transaction couldn't start until the previous transaction 1) wrote its prepare, 2) sent its vote, 3) waited for the coordianator to write its commit, 4) waited for the coordinator to send the commit, and 5) waited for the commit record to go to disk. Hence, the EC2PC throughput would be 100 tps (10 ms / transaction), and the conventional 2PC would be 20 TPS (50 ms / transaction.)

EC2PC Throughput: _____

Conventional 2PC Throughput: _____

To test the correctness of EC2PC, Dana creates a table with a few hundred records with two fields, `(account, balance)`, where the table is partitioned across two subordinates by hashing on `account`, with a separate third coordinator. She runs transactions that transfer money from one account to another, i.e.,

```
transfer(A,B, amt):
    t1 = Read(A)
    t2 = Read(B)
    if (t1 > amt):
       Write(A,t1-amt)
       Write(B,t2+amt)
```

10. [6 points]: Dana concurrently runs many such transactions that choose random A,B pairs and finds that money is not preserved, i.e., the total sum of all accounts is not the same before and after the workload runs. Explain briefly a situation in which this could occur when running EC2PC.

Name:

(Write your answer in the space below.)

Answer. Suppose a transaction T1 runs and one subordinate prepares, releases locks, but does not commit. Some other transaction T2 then runs and performs writes to the same bank accounts as T1, and completes/commits. If T1 then aborts, T2 will not be rolled back, because it is no longer holding locks, but it has seen T1's dirty data.

V Amazon Dynamo

The follow questions are based on the paper by DeCandia et al, "Dynamo: Amazon's Highly Available Key-value Store", and the in class discussion we had about it.

11. [4 points]: Consider a Dynamo deployment with two replicas R_1 and R_2 storing key K. If R_1 contains version V_1 of K with vector clock $< R_1 : 1, R_2 : 0 >$ and R_2 contains version V_2 of K with vector clock $< R_1 : 1, R_2 : 1 >$ and there are no additional writes to the system, which version(s) will reads from key K eventually return?

(Circle the best answer.)

A. V_1 *Answer.* No

B. V_2 *Answer.* Yes

C. V_1 and V_2 *Answer.* No

D. No writes *Answer.* No

12. [4 points]: With the same replica configuration, what would happen if V_2's vector clock was instead $< R_1 : 0, R_2 : 1 >$? That is, if $V_1 =< R_1 : 1, R_2 : 0 >$ and $V_2 =< R_1 : 0, R_2 : 1 >$ and there are no additional writes to the system, which version(s) will reads from key K eventually return?

(Circle the best answer.)

A. V_1 *Answer.* No

B. V_2 *Answer.* No

C. V_1 and V_2 *Answer.* Yes

D. No writes *Answer.* No

Name:

13. [6 points]: Suppose there are three replicas, R_1, R_2, and R_3. Three writes are performed to key K, resulting in three version clocks: $V_1 =< R_1 : 0, R_2 : 3, R_3 : 2 >$, $V_2 =< R_1 : 1, R_2 : 3, R_3 : 2 >$, $V_3 =< R_1 : 0, R_2 : 0, R_3 : 3 >$. Which of the following are true statements?

(Circle True or False for each item below.)

A. True / False The writer that produced V_1 observed V_2. *Answer.* No

B. True / False The writer that produced V_2 observed V_1. *Answer.* Yes

C. True / False The writer that produced V_3 observed V_1. *Answer.* No

D. True / False V_1 and V_2 are concurrent writes. *Answer.* No

E. True / False V_2 and V_3 are concurrent writes. *Answer.* Yes

F. True / False V_1 and V_3 are concurrent writes. *Answer.* Yes

Name:

VI Distributed Query Processing

Ben Bitdiddle stores data for a restaurant review website in a 4-node distributed database. The tables in the database including their sizes are shown below. Each node has 2 GB of RAM and 2 TB of disk space. Any node can send data to any other node on the network at 10 MB/s and each node has a single disk that can read at 50 MB/sec. In this problem there are no indexes, and network latency, CPU time, and seek times are negligible. Assume that data is uniformly distributed in each table.

```
Restaurants:    (id int primary key,
                 cid int references cities.id,
                 name varchar,
                 cuisine varchar, ...)

Users:          (id int primary key,
                 name varchar, ...)

UserRatings:    (uid int references users.id,
                 rid int references restaurants.id,
                 rating int, ...)

Cities:         (cid int primary key,
                 state varchar, ...)
```

Table	Size	Cardinality
Restaurants	60 GB	10^8
UserRatings	100 GB	10^{10}
Users	20 GB	10^6
Cities	6 GB	10^6
UserSimilarity	2 GB	10^{12}

Figure 2: Table sizes and cardinalities.

The most commonly issued query on this website is the following:

```
SELECT r.id, SUM(ur.rating) as rating
FROM Restaurants r, UserRatings ur, Cities c
WHERE ur.rid = r.id
AND r.cid = c.id AND c.name = 'some city'
GROUP BY r.id
ORDER BY rating DESC LIMIT 20
```

Name:

This query selects the 20 most highly rated restaurants in a given city. Note that *city names are unique.* Ben is assigned the task of distributing the database over 4 nodes to optimize the performance of the above query. Ben chooses the following partitioning for each table.

```
Restaurants: Hash Partitioning on cid
UserRatings: Hash Partitioning on rid
Cities: Hash Partitioning on id
```

14. [12 points]: Draw or describe the best query plan for executing the above query. You should consider both local and distributed aspects of query planning, including how data is repartitioned (if needed), the type and order of local joins and filters on each node, and how the final answer is generated.

(Write your answer in the space below.)

Answer. Cities will first be scanned to select cities with name='some city' and then will be joined with restaurants on r.cid = c.id locally. This will identify restaurants in the given city. Next, we would need to repartition the resulting restaurants based on id. Then each node would compute the join on r.id=ur.rid locally and aggregate ratings. Finally, each node would send aggregate results to the coordinator.

15. [10 points]: How long would the plan you described in the previous problem take to execute?

(Write your answer in the space below.)

Answer. Assume that c.name are randomly distributed. In this case, each node stores 1.5 GB of cities, 25 GB of UserRatings and 15 GB of Restaurants.

Selection on cities requires a single scan of the table, which takes 30s. Also, we know that city names are unique, therefore a single city record is produced as a result. Next, we scan the restaurants table to find restaurants in the city. This would take 300 s. Assuming restaurants are evenly distributed across cities, there are 100 restaurants in each city. Now, we must repartition the 100 restaurants based on id. The data to be transmitted is negligible, so we can ignore network cost. Next, each node locally joins on r.id=ur.rid using a hash join (the sub-setted r relation can fit in memory.) The cost is reading in the UserRating table, which is 500 s. Final step is sending aggregates to the coordinator node, the network cost of which is also negligible. Therefore, resulting latency is 830 s.

End of Quiz II

Name: