There are 18 questions and 14 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

<table>
<thead>
<tr>
<th>1-4 (xx/30)</th>
<th>5-12 (xx/42)</th>
<th>13-18 (xx/28)</th>
<th>Total (xx/100)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name: SOLUTIONS
Short Answer

1. [9 points]: Ronnie S. Rocker is using Bigtable to store the list of all Rock and Roll Hall of Fame inductees. The row key for an inducted band (or performer) is simply the year it was inducted followed by the band name, e.g. the band “Lynyrd Skynyrd” would have a row key 2006–Lynyrd–Skynyrd (the table has only one row per inducted band). Details about a band are stored using several column families, such as name, members–inducted, albums, photos and so on. The column families name and members–inducted are a part of one locality group called BAND–METADATA.

For each of the following statements, indicate whether it is True or False.

(Circle True or False for each of the following.)

A. True / False  After this year’s Cleveland induction ceremony, Ronnie has decided to update his list to include all the 2012 inductees. Bigtable does not allow Ronnie to insert all the 2012 inductee rows together as a part of a single transaction but if he only includes one locality group (BAND–METADATA) for each row instead of all the column families, he can insert all the rows in a single transaction.

Explanation:
Bigtable only supports single-row transactions. Grouping columns into locality groups leads to more efficient reads but does not allow transactions across multiple row keys.

B. True / False  Ronnie realizes to his horror that he has inserted an incorrect value in the members–inducted column family for the 2012 inductee “Guns N’ Roses”. He decides to delete the row with row-key 2012–Guns–N–Roses before re-inserting a new row with the correct values. Assuming the original insert had propagated from the memtable to one of the SSTables on the tablet to which the Guns N’ Roses row is assigned, Bigtable can process this deletion without doing a major compaction despite SSTables being immutable.

Explanation:
The deletion can be processed by writing a special deletion entry to the memtable that suppresses live data that has already propagated to a SSTable. A major compaction get rids of all deleted data but is not required for deletion processing.

C. True / False  Ronnie is writing a client application to retrieve album data of all bands inducted during the 1990s; this data is spread across three different tablet servers. Locations for two of these tablet servers are already in the client cache but one of these two values is stale (although the client will only realize this after a cache miss). Before beginning to retrieve all the album data from the 1990s inductees, the client library’s table location algorithm will require at least six network round-trips to locate the three tablet servers.

Explanation:
Bigtable uses a three-level hierarchy to store tablet location information. There are three cases:
– Location known and correct: No network roundtrips needed to locate the server.
– Location known but stale: Requires at least two network roundtrips because stale cache entries are only discovered upon misses (at most six are needed).
– Location unknown: At least one network roundtrip is needed (at most three).

In total, at least three network roundtrips are needed to locate the three tablet servers.

Name: Solutions
2. [8 points]:

_FIFA Ballon d’Or_ is football’s highest individual and is awarded to the player who is considered to have performed the best in the previous season. You are a journalist deciding whether to vote for Lionel Messi or Cristiano Ronaldo. To that end, you decide to compute statistics for each player, namely the number of goals scored by Messi and Ronaldo aggregated over each month.

You decide to compute this using MapReduce. Please complete pseudo-code for the map and reduce functions. You’re given an input set of all goals scored during last season whose record schema is [INT goalid, TEXT scorer-name, INT month]. Use the function `emit(key, value)` in your map job to emit a tuple for the reduce job. The reduce job will be called once per distinct key emitted by any mapper, with an iterator over all values corresponding to that key. Your reduce function should also call `emit(key, value)` to produce results; the union of the results from all of the reducers should be the desired set of goal statistics.

```plaintext
function map(input) {
    records = load(input, ...)

    for r in records:
        if r.scorer-name in ('Lionel Messi', 'Cristiano Ronaldo'):
            emit ({r.scorer-name, r.month}, 1)
}

function reduce(key, iterator) {
    emit (key, len(list(iterator)))
}
```

Name: Solutions
3. [10 points]: Suppose you are building a highly available database system, where the primary database server has a replica that can be failed over to in the event of a crash. This database system uses strict two-phase locking for concurrency control. For each of the following replication schemes, indicate whether it is transactionally consistent. By this, we mean that after the replica takes over for the primary the follow conditions are true:

1. Any transaction that had committed on the primary also will have committed on the replica.
2. All committed transactions will have run in the same equivalent serial order on both primary and replica.
3. Any transaction that rolled back or aborted on the primary will also have rolled back or aborted on the replica.

Circle YES or NO to indicate whether the describe approach is transactionally consistent or not, and write a one sentence explanation for your answer.

A. The primary and replica are both deterministic, and the primary ensures replica has received the SQL for each transaction before it commits it.

Transactionally consistent: YES NO
Explanation:

Deterministic execution along with ensuring the replica has received the SQL ensures transactional consistency.

We also accepted NO as an answer for this problem as long as the explanation pointed out at least one of the two reasons below:

– that the primary and replica might both fail, leading to scenarios where property 1 or 3 is violated.
   (The failure model given in the problem implicitly assumes only one server will fail at a time but this is not explicitly stated, so this answer is acceptable.)

– that the replica might crash before logging the statement or executing the transaction.
   (The problem again implicitly assumes that the SQL statement is acknowledged as having been received by the replica only after it is logged. Full points were awarded to students who pointed this out.)

B. The primary and replica are both non-deterministic, and the primary ensures the replica has received the SQL for each transaction before it commits it.

Transactionally consistent: YES NO
Explanation:

Irrespective of server failures, non-deterministic execution means that condition 2 for transactional consistency cannot be satisfied in this case.

(Problem continues on next page.)

Name: Solutions
C. The primary and replica are both non-deterministic, with transactions coordinated using two-phase commit. The primary ensures the replica has executed the SQL for a transaction and is in the \textit{PREPARED} state before it commits/aborts the transaction. The primary does not crash before the secondary learns about the transaction outcome.

Transactionally consistent: \textbf{YES} \hspace{1cm} \textbf{NO}

Explanation:

\textit{Two-phase commit along with strict two phase locking ensures transactional consistency despite execution being non-deterministic.}

D. The primary ships every log record it writes to the replica, and the replica replays log records in order. Before writing a commit record, the primary waits to receive an acknowledgement that replica has received and written the commit record.

Transactionally consistent: \textbf{YES} \hspace{1cm} \textbf{NO}

Explanation:

\textit{In-order replay of log records at the replica site ensures transactional consistency in much the same way as deterministic execution.}

\textit{We also accepted NO as an answer for this problem if the explanation pointed out that the primary and replica might both simultaneously fail, leading to scenarios where property 1 or 3 is violated.}

4. [3 points]: In an ARIES-based logging system, how could you simplify the system if you knew that the database never wrote dirty pages from uncommitted transactions to disk?

\textbf{(Write your answer in the space below.)}

\textit{In this case, a separate UNDO phase is not required during recovery. Undo related logging work such as before-images of pages also need not be done at run-time.}

\textit{[Full points were awarded only if both these points were mentioned in the answer.]}

Name: Solutions
5. [4 points]: Suppose you are designing a database system designed to process thousands of small transactions per second. Your system uses a multi-threaded design, with one thread per transaction, employs strict two-phase locking for concurrency control, and does write-ahead logging with a FORCE of the log tail before each commit for recovery. It has a single disk, which is used for both the database pages and the log.

You find that your system can only process about 100 transactions per second, and observe that the average RAM utilization and CPU load of the system while processing transactions is very low. Assuming you cannot add new hardware, what would you recommend as a way to improve the throughput of the system?

(Write your answer in the space below.)

Based on the information given, the slowdown is either due to flushing of the log on every transaction, or possibly due to lock conflicts/deadlocks between the transactions. Therefore, we accepted two answers: 1) Use group commit, increasing latency of transactions, but allowing multiple log writes to be batched together, or 2) run one transaction at a time, or use some kind of deterministic ordering of transactions to avoid lock contention.

II Pig

Suppose you have three Pig tables, papers, authors, and paperauths, defined as follows:

papers = load 'papers.dat' using PigStorage('	') as (pid:int, pname:chararray );
authors = load 'authors.dat' using PigStorage('	') as (aid:int, aname:chararray);
paperauths = load 'paperauths.dat' using PigStorage('	') as (pa_pid:int, pa_aid:int);

Now, consider the following Pig program (note that this program doesn’t actually use the author’s table).

x = group paperauths by pa_pid;
x2 = foreach x generate flatten(paperauths), COUNT(paperauths) as count;
x3 = cogroup x2 by pa_pid, papers by pid;
x4 = foreach x3 generate flatten(papers), flatten(x2);
x5 = foreach x4 generate pname, count;
dump x5;

Here dump is a Pig command to print a result set.

Name: Solutions
6. **[4 points]**: Assuming that there are relational tables with the same schema as papers and paper-auths, what SQL query is equivalent to the above program?

   (Write your answer in the space below.)

   ```sql
   SELECT pname, count(*)
   FROM papers, paperauths
   WHERE paperauths.pa_pid = papers.pid
   GROUP BY pname;
   ```

7. **[4 points]**: How many map/reduce jobs would this translate into?

   Number of Map Reduce Jobs: 2

8. **[6 points]**: Describe briefly what the first map job does.

   (Write your answer in the space below.)

   It reads the paperauths file, iterating through the records and emitting tuples of the form pid, (paper-auths tuple) (i.e., where the key is pid and the value is the tuple.)

9. **[6 points]**: Describe briefly what the first reduce job does.

   (Write your answer in the space below.)

   For each tuple t in each {paperauth tuple} set S passed into the reduce job, emit t.pa_pid, COUNT(S).
   We also accepted answers that did not do the COUNT in the reduce (presumably deferring it to the next map job.)

---

Name: Solutions
III Logging

You are running an ARIES-based database system that employs strict two phase locking. There are 4 transactions running in the system, T1–T4. The system crashes and recovers, and after recovery you observe that transactions T1 and T2 have committed, and transactions T3 and T4 have aborted or rolled back.

Given this outcome, your job is to reconstruct the state of the log prior to the execution of recovery, given a partial log fragment and database state on disk (also prior to recovery).

Assume that pages may be flushed to disk at any time, and that the only log records that may appears are start of transaction (SOT), COMMIT, ABORT, and update (UP) records (e.g., there are no checkpoint records, CLR records, or END records).

10. [10 points]: The log and database state at the time of crash are given below. Assume that this is the entire log. Fill in the missing rows in the low (rows marked with a *) – there may be multiple valid answers to this question:

<table>
<thead>
<tr>
<th>LSN</th>
<th>Type</th>
<th>TID</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SOT</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>*2</td>
<td>SOT</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>UP</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>UP</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>SOT</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>*6</td>
<td>SOT</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>*7</td>
<td>COMMIT</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>UP</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>UP</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>*10</td>
<td>UP</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>11</td>
<td>UP</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>12</td>
<td>COMMIT</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>UP</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>*14</td>
<td>UP</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>*15</td>
<td>ABORT</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>UP</td>
<td>4</td>
<td>B</td>
</tr>
</tbody>
</table>

Database state:

<table>
<thead>
<tr>
<th>Page</th>
<th>LSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>13</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>14</td>
</tr>
</tbody>
</table>

Records 6 and 7 can appear in the opposite order.

Name: Solutions
11. [4 points]: What were the contents of the dirty page table at the time of the crash?
(Fill in the table below; some entries may be unused.)

<table>
<thead>
<tr>
<th>Page</th>
<th>RecLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>16</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
</tr>
</tbody>
</table>

12. [4 points]:
What were the contents of the transaction table at the time of the crash?
(Fill in the table below; some entries may be unused.)

<table>
<thead>
<tr>
<th>TID</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>

*TID 3 appears because there is no END record for transaction 3 in the log. Answers without transaction 3 were also accepted because the problem setup stated that the log does not contain END records.*
IV Distributed Query Processing / Transactions

The giant retailing chain MaddenMart hires you to build a system to generate monthly sales reports that indicate the average sales in different regions and product categories. You decide to implement this reporting functionality using a distributed database. Because MaddenMart’s sales are so tremendous, you build a distributed database with 3 nodes. Your database has the following tables (this schema is similar to the TPC-H benchmark used for OLAP systems):

orders: (o_id int primary key,
         o_date datetime,
         o_store int references stores.s_id, ...)

products: (p_id int primary key,
           p_desc varchar,
           p_category int,
           p_price float, ...)

stores: (s_id int primary key,
         s_region int,
         s_address varchar, ...)

order_items: (oi_id int primary key,
              oi_o_id int references orders.o_id,
              oi_p_id references products.p_id,
              oi_discount float,
              oi_quantity int, ...)

Here, order_items records the specific products bought as a part of each order. Dates are uniformly distributed between January 1, 2000 and January 1, 2010. The orders.o_id attribute is assigned in monotonically increasing order, as is the o_date attribute. There are a total of 10 regions. Each order has about the same number of order_items.

You decide to store these tables with no indexes and unsorted.

These tables have the following cardinalities and sizes:

<table>
<thead>
<tr>
<th>Table</th>
<th>Size</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>orders</td>
<td>10 GB</td>
<td>$10^8$</td>
</tr>
<tr>
<td>products</td>
<td>10 MB</td>
<td>$10^4$</td>
</tr>
<tr>
<td>stores</td>
<td>1 MB</td>
<td>$10^3$</td>
</tr>
<tr>
<td>order_items</td>
<td>200 GB</td>
<td>$10^9$</td>
</tr>
</tbody>
</table>

Name: Solutions
Suppose you want to run the query:

```sql
SELECT s_region, SUM(p_price * oi_discount * oi_quantity)
FROM orders, products, stores, order_items
WHERE o_date > 'January 1, 2009'
AND oi_o_id = o_id
AND oi_p_id = p_id
AND o_store = s_id
GROUP BY s_region
```

Each of your machines has 5 GB of memory, and each node can send to any other node over the network at 10 MB/sec, and your disks can read at 50 MB/sec.

13. **[4 points]**: What partitioning / replication strategy would you recommend for each of the tables? Assume that tables are updated in batches over night, and that regardless of the partitioning strategy, update performance is not a concern.

Choose between:

1. Hash partitioning on a single attribute,
2. Range partitioning on a single attribute, or
3. Replicating a table on each node.

Assume that when you choose hash or range partitioning, each node receives about 1/3rd of the data.

(For each table, indicate your partitioning scheme, and the partitioning attribute (if any))

**A. orders**
- Partitioning scheme (1–3): __1__
- Partitioning attribute (if not replicated): __o_id__

**B. products**
- Partitioning scheme (1–3): __3__
- Partitioning attribute (if not replicated): _________

**C. stores**
- Partitioning scheme (1–3): __3__
- Partitioning attribute (if not replicated): _________

**D. order_items**
- Partitioning scheme (1–3): __1__
- Partitioning attribute (if not replicated): __oi.o_id__

Name: Solutions
14. [8 points]: Describe or draw the optimal query plan for your query given the physical design you selected above. 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. Be sure to indicate which relation is the inner/outer in your joins.

(Write your answer or draw your plan in the space below.)

Each node locally joins its orders and order_items partitions and computes the answer over its local data. No repartitioning is required. The best local plan is:

```
(((order_items ◦ filter(order_items)) ◦ stores) ◦ products)
```

where the plan is left deep and the relation of the left is the outer relation in the join. For each join, the inner relation fits into memory, so we can perform an in-memory hash joins all the way up. For the first join, we can hash on either table. Each node computes partial aggregates over its and so transmits its (at most) 10 groups to one of the coordinator node, which performs the final summation.

Name: Solutions
15. [3 points]: Estimate the number of bytes read from disk and sent over the network by each node. Assume an integer, date, or float is 4 bytes, and a string is 100 bytes.

<table>
<thead>
<tr>
<th>Node</th>
<th>Bytes Sent Over Network</th>
<th>Bytes Read From Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10 MB + 1 MB + 200/3 GB + 10/3 GB ≈ 70 GB</td>
</tr>
<tr>
<td>2</td>
<td>10 * (4+4) = 80 bytes</td>
<td>10 MB + 1 MB + 200/3 GB + 10/3 GB ≈ 70 GB</td>
</tr>
<tr>
<td>3</td>
<td>10 * (4+4) = 80 bytes</td>
<td>10 MB + 1 MB + 200/3 GB + 10/3 GB ≈ 70 GB</td>
</tr>
</tbody>
</table>

16. [3 points]: Ignoring CPU time, estimate the total runtime of your query plan.

(Write your answer in the space below.)

Around 1400 seconds to perform disk I/O, network I/O time is negligible.

Now that you’ve designed the schemas and chosen a physical design that is good for generating the required reports, you turn your attention to building the loading system that is used to populate these tables. Every night, a single thread takes all of the transactions that were processed that day, transforms them into the appropriate format for your reporting database, and issues INSERT commands to add them.

Initially, you plan to use two-phase commit (2PC) where the coordinator is the node doing the loading, and the workers are your three database nodes. The idea is to ensure that INSERTS that need to go to multiple tables (e.g., tables that are replicated) are processed atomically. You plan to use the basic (non-presumed abort or presumed commit variant of 2PC).

Ben Bitdiddle tells you that he believes two-phase commit will be slow, due to the forced logging of updates and multiple network round trips. Instead of force-writing PREPARE and COMMIT log records on workers, Ben suggests not forcing them (i.e., just allowing them to propagate to disk when file system buffers are flushed).

17. [4 points]: Being a good 6.814/6.830 student, you recognize that Ben’s design could cause problems. List one problem that could occur as a result of Ben’s suggested change?

(Write your answer in the space below.)

Worker could crash after coordinator receives ACKs from all workers and forgets about transaction, but before it has flushed COMMIT. Now when the worker recovers, it will see a PREPARE for the transaction and contact the coordinator. Since the coordinator forgot about the transaction, it will reply with ABORT and the worker will decide to abort the transaction – clearly this is wrong.

Name: Solutions
18. [6 points]: Alyssa P. Hacker points out that since tables are unsorted and uncompressed, updates are simply being appended to the end of each partition on each node. Furthermore, you only have one thread performing load operations at a time, and you never need to rollback inserts. How would you exploit this to avoid the use of 2PC, while still preserving atomicity?

(Write your answer in the space below.)

Once workers have acknowledged / logged inserts, since you know you will never abort / rollback, you don’t need 2PC. So a single phase commit protocol is fine.

Additionally since inserts are processed in-order/one at a time there’s no need to worry about locking at all on the workers, except as it relates to concurrent queries reading the database. These reads can be isolated from inserts by simply not scanning outstanding records during table scans, which doesn’t require fine-grained locking, esp. for the read queries.

End of Quiz II