In order to receive credit you must answer the question as precisely as possible. You have 100 minutes to answer this quiz.

Some questions may be much harder than others. Read them all through first and 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.

<table>
<thead>
<tr>
<th>I (xx/16)</th>
<th>II (xx/14)</th>
<th>III (xx/15)</th>
<th>IV (xx/15)</th>
<th>Total (xx/60)</th>
</tr>
</thead>
</table>

Name :

NYU ID (N*******):
I Multiple choice questions (16 points):

Answer the following multiple-choice questions. Circle all answers that apply. Each problem is worth 4 points. Each missing or wrong answer costs -2 point.

A. Which of the following statements are true about two phase commit (2PC)?

1. 2PC is an essential component in implementing distributed transactions.
2. In traditional 2PC, the coordinator may unilaterally abort the transaction.
3. There is no way to make 2PC fault-tolerant because the failure of the coordinator may block any further progress.
4. A participant of a 2PC may release locks on the transaction’s data objects upon voting "yes".

B. Which of the following statements are true for Google’s Percolator and Spanner systems?

1. Both Percolator and Spanner provide transactions with the same semantics.
2. In Percolator, the client acts as the coordinator of the 2PC.
3. Percolator and Spanner use the same mechanism for assigning timestamps to each transaction.
4. Read-only transactions in Spanner must grab read locks according to the 2-phase locking protocol.

C. Which of the following things are true for Paxos?

1. If there are two concurrent proposers, Paxos will forbid both proposers from getting a majority of prepare-ok replies.
2. Once the proposer has received a majority of prepare-ok replies, it is free to send the accept request with any value that it likes to propose.
3. Each replica in Paxos must log its latest accepted proposal number and the corresponding value to disk if it wants to come back following a crash.
4. It takes a minimum of two network roundtrips for any proposer to have its value committed for a Paxos instance.
5. It takes a minimum of three network roundtrips for any proposer to have its value committed for a Paxos instance.

D. Which of the following statements are true for MapReduce and Piccolo?

1. MapReduce handles node failure by re-executing any tasks assigned to the failed node on other nodes.
2. Piccolo can also handle node failure by re-executing the kernels assigned to the failed node.
3. In MapReduce, a reducer may start if and only if all mappers have finished.
4. In Piccolo, accumulations on the in-memory distributed table may start if and only if all kernels have finished execution.
II Paxos

Ben is using Paxos to maintain the membership views of a changing set of replicas/nodes. To begin with, every node has static knowledge of the first view in the system (with identifier zero and node A only), i.e. \{vid=0, nodes=(A)\}. If a new node joins the system, it would attempt to initiate Paxos with a new view containing itself. If an existing node crashes, any remaining live node would attempt to initiate Paxos with a new view that excludes the crashed node.

1. [4 points]: Ben started the system with node A. Subsequently, he started node B, then C, and then D, in sequence. He then killed node C. (Before starting or killing any node, Ben makes sure that the existing view stabilizes.) Please enumerate the resulting sequence of views that Ben’s system goes through including view numbers and the corresponding replica set.
2. **[10 points]:** In Ben’s current implementation, whenever a new node joins the system, it initiates a Paxos for the new view among the existing set of replica nodes and itself. For example, suppose the current view is \( \{\text{vid}=100, \text{nodes}=(A,B,C,D)\} \) and then node E joins the system. Node E would try to reach consensus on the new view \( \{\text{vid}=101,\text{nodes}=(A,B,C,D,E)\} \) among nodes A,B,C,D,E.

Alyssa P. Hacker thinks Ben’s design is incorrect and that the new node should not participate in the Paxos agreement for the new view, i.e. in the previous example, only nodes A,B,C,D (excluding E) should participate in Paxos to agree on \( \{\text{vid}=101,\text{nodes}=(A,B,C,D,E)\} \).

Is Ben’s design correct? If so, provide an explanation. If not, give a concrete counterexample on how his design would lead to inconsistency (i.e. violation of sequential consistency).
III MapReduce

Randy and Alex have been working on optimizing Hadoop MapReduce for small jobs whose intermediate data can fit in the memory. In their current prototype, mappers send all intermediate key/values to the master node who stores them in an in-memory hash table (keyed by the intermediate keys). The master node may also apply user-defined accumulation functions (also called combiners, see MapReduce paper Sec 4.3) to aggregate multiple values corresponding to the same key. For example, for the word count application, the programmer may specify the \texttt{sum} function to aggregate intermediate values for the same key when mappers update the master node’s hash table. Randy and Alex also changed the reducers to read all or aggregated intermediate key/values from the master node’s hash table.

3. **8 points**: Can Randy and Alex’s optimized system use Hadoop’s existing failure recovery mechanism by re-executing failed mappers or reducers? Why and why not. (Please give concrete examples whenever possible.)
4. [7 points]: Can Randy and Alex’s system use Hadoop’s existing straggler mitigation mechanism (i.e. by speculatively re-executing mappers or reducers whose progress is deemed too slow)? Why and why not. (Please give concrete examples whenever possible.)
IV Isolation and Fault tolerance

As part of the YFS labs, Ben Bitdiddle has implemented Paxos and used it to maintain replica membership in a primary-backup based RSM. Therefore, as long as no more than half of the existing replicas fail, the RSM can maintain data consistency and durability.

Ben has used the fault-tolerant RSM to replicate both the lock server and extent server. Thus, he now believes that the resulting YFS handles both concurrency and failures perfectly.

Alyssa P. Hacker is skeptical of Ben’s claims. Specifically, she thinks Ben is overlooking yfs_client failures. Alyssa identifies yfs_client::unlink as a concrete example to show Ben why client failures are problematic. (In the code snippet blow, all error checking on return values has been omitted for brevity.)

```c++
//remove file named `name' in parent directory with inum `dir'
int yfs_client::unlink(inum dir, std::string name)
{
    ... 
    1: lc->acquire(dir); //grab lock, lc is an instance of lock_client
    2: 
    4: xinum = ilookup(dir, name); //lookup file’s inum from extent service
    5: 
    6: lc->acquire(xinum);
    7: 
    8: //fetch dir content into string `extent’ from extent service
    9: ec->get(dir, extent);
    10: 
    11: //modify `extent’ to exclude the file with name `name’
    12: erase_entry(name, extent);
    13: 
    14: ec->remove(xinum); //remove file extent
    15: 
    16: ec->put(dir, extent); //update modified dir content
    17: 
    18: lc->release(xinum);
    19: lc->release(dir);
}
```

5. [5 points]: What’s the worse that can happen if a yfs_client node fails at line 2? Explain.
6. [10 points]: What’s the worse that can happen if a \texttt{yfs\_client} node fails at line 15? Explain.