*Department of Electrical Engineering and Computer Science*
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

# 6.033 Computer Systems Engineering: Spring 2019

# Quiz 2

There are 14 questions and 13 pages in this quiz booklet. Most questions have multiple parts. Answer each question according to the instructions given. You have two hours to answer the questions.

- The questions are organized (roughly) by topic. They are not ordered by difficulty nor by the number of points they are worth.
- **If you find a question ambiguous, write down any assumptions you make.** Be neat and legible.
- Some students will be taking a make-up exam at a later date. Do not discuss this quiz with anyone who has not already taken it.
- You may use the back of the pages for scratch work, but do not write anything on the back that you want graded. We will not look at the backs of the pages.
- Write your name in the space below. Write your initials at the bottom of each page.

This is an open-book, open-notes, open-laptop quiz, but you may NOT use your laptop, or any other device, for communication with any other entity (person or machine).

**Turn all network devices, including your phone, off.**

**Name:** Solutions!

**Email:** **@mit.edu**

*Please use only your MIT email ID.*

## GFS and ZFS

1. [5 points] For each of the following phrases, indicate whether it relates to GFS, ZFS, both, or neither. Pick the BEST choice for each line. Circle only a single choice on each line.

   A. **GFS** / **ZFS** / **Both** / **Neither** Reads may see inconsistent versions of data

   B. **GFS** / **ZFS** / **Both** / **Neither** Multiple different sizes of block/chunk

   C. **GFS** / **ZFS** / **Both** / **Neither** Logging based on blockchain

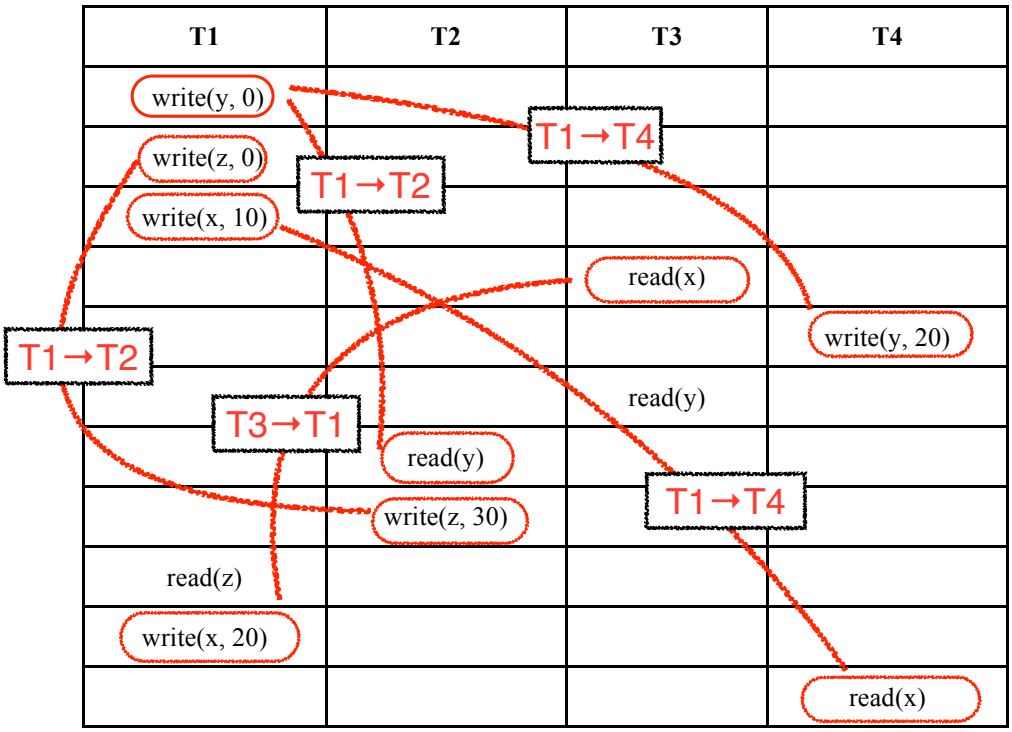
   D. **GFS** / **ZFS** / **Both** / **Neither** Leases

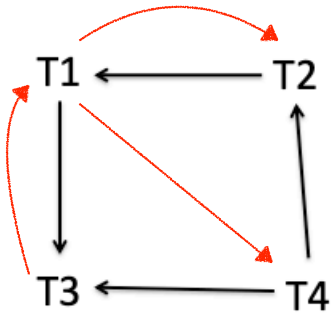
   E. **GFS** / **ZFS** / **Both** / **Neither** Checksum for child block/chunk stored in parent block/chunk

Initials:

## Transactions

2. [8 points] Consider the following schedule and conflict graph for transactions T1, T2, T3, T4. For the schedule, time increases as you move down the page.




A. Which arrows need to be added to the diagram so that it is complete?

a. T1 → T2 and T3 → T1 (Incomplete)
b. T1 → T4 and T4 → T1 (T4 → T1 incorrect)
c. T1 → T2, T3 → T1, and T1 → T4
d. T1 → T2, T3 → T1, T1 → T4, and T4 → T1 (T4 → T1 incorrect)
e. T1 → T2, T3 → T1, and T3 → T2 (T3 → T2 incorrect)
f. No more arrows needed, the diagram is complete as given

B. Identify a SINGLE action in the above schedule that can be removed to make this group of transactions conflict-serializable.

a. T3: read(x)
b. T2: read(y)
c. T1: write(x,10)
d. T4: write(y, 20)
e. T2: write(z, 30)
f. There is no single action that can be removed to produce a conflict-serializable schedule

There are multiple cycles in the conflict diagram. For the schedule to be conflict-serializable, the conflict diagram must be acyclic. Removing any single action does not produce an acyclic diagram.

Initials:

## Replicated State Machines

3. [6 points] Consider a replicated state machine implemented by N+1 replicas. There is one primary P and N backups: $B_1$, $B_2$, .., $B_N$. However, there is no view server.

   Instead, this replicated state machine uses the IDs of the backups as an ordering to determine who becomes the next primary in case of failures. Specifically: if P fails, everyone in the system has agreed that $B_1$ will be the next primary. If $B_1$ fails, $B_2$ will become primary, and so on through $B_N$.

   Assume that there is an efficient heartbeat protocol that allows any replica to check the status of any other replica by sending a request and receiving a corresponding reply. The effect is that every replica knows instantaneously when some other replica is not responding. Further assume that each replica is either "up" (fully functioning, responding to messages) or "down" (not responding to messages), so you do not need to be concerned about partial failure of a replica or recovery of a replica.

   We'll call it a *partition* whenever there is any impairment of replica-to-replica communication. One particular kind of partition is an *asymmetry in message delivery*: that's a partition where some replica A can't send to some other replica B, even though B can send to A.

   Answer True or False to each of the following completions of the claim that "This scheme ensures that there is always **at most one** replica acting as primary **out of the entire set of replicas**…"

   A. **True** / **False** …in any network that is guaranteed to have no partitions.

   B. **True** / **False** …even in a network that allows partitions, as long as there are no asymmetries in message delivery.

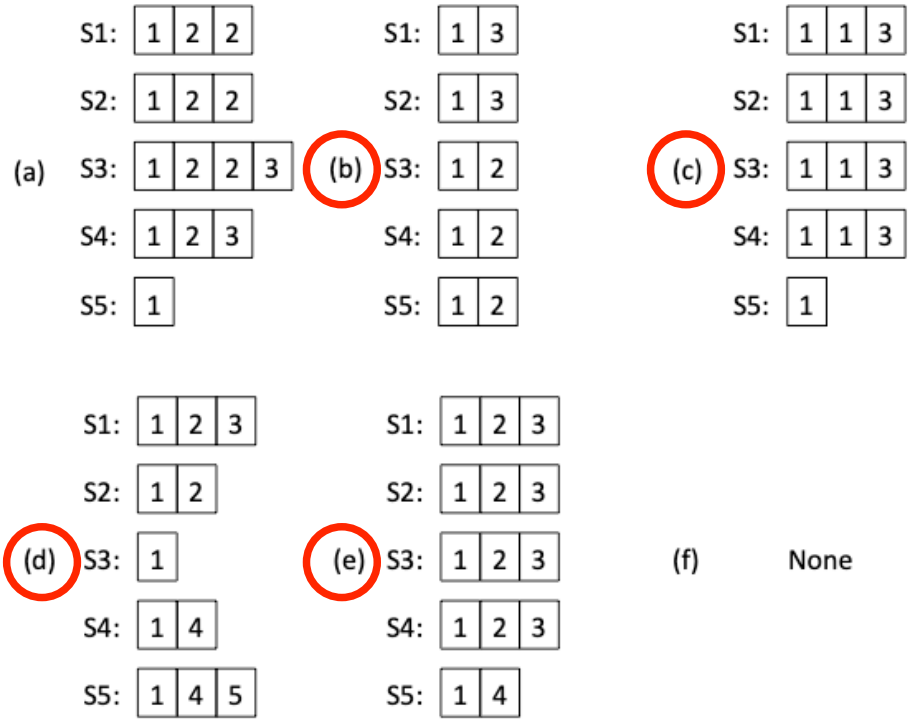
   C. **True** / **False** … even in a network that has asymmetries in message delivery.

   The scheme as described assumes that replicas can communicate. (A) is basically asking whether the scheme works, and we essentially told you that it does. For (B) and (C) we have to consider what happens when replicas may not be able to communicate. Whenever there is a partition, the heartbeat protocol may not work correctly: when a replica doesn't get an expected reply, it assumes the other party has failed — but maybe a message just couldn't get through. It doesn't matter whether there are asymmetries or not, because any kind of problem with the heartbeat protocol means that the scheme can't ensure a single primary in all situations.

## Raft

4. [10 points] Each of the following diagrams shows the logs at five servers S1 through S5. Assume that the five servers are implementing five Raft replicas of a single replicated state. As in the paper, the log entries have been abstracted to show term numbers only.

In (a), S4 is not possible. Before it could log an entry in term 3, it would have had to "catch up" with the second term-2 entry at S1, S2, S3




Identify ALL of the diagrams that are legal in Raft (if any). Clearly indicate your choice(s) by circling the corresponding letter(s).

If you decide that none of them are legal, circle choice (f) above instead.

In all scenarios, all nodes are up and functioning for term 1 and all successfully log first entry. Some node(s) crash(es) after that:

(b) S1,S2 crash after entry for term 1. S5 elected leader with S3, S4, logs entry for term 2. S3 crashes before receiving entry for term 2. S4, S5 crash. S1, S2, S3 recover, S1 elected leader. S1, S2 log entry for term 3, S3 crashes before receiving entry for term 3. S1, S2 crash; S3, S4, S5 recover. S5 elected leader (note that it's now term 4!). S5 updates S3 log to match S5's.

(c) S5 crashes after receiving first entry. Some subset of remaining nodes elect a leader for term 2, but the leader fails before any updates are logged. All four non-S5 nodes are up and elect a leader for term 3, successfully logging an entry.

(d) S4, S5 crash after entry for term 1. S1, S2, S3 elect S1 leader. Entry for term 2 does not reach S3 before it crashes. S3 recovers; S1,S2,S3 elect S1 leader for term 3. S1 logs entry for term 3, but both followers crash before receiving it. S1 crashes. S3, S4,S5 recover and elect S4 leader for term 4. S4 and S5 log entry for term 4, but S3 crashes before receiving it. S3 recovers; S3, S4, S5 elect S5 leader. S5 has logged an entry for term 5, but S4 and S5 have crashed before receiving their updates.

(e) S4, S5 crash after entry for term 1. S1, S2, S3 elect S1 leader. Entry for term 2 does not reach S3 before it crashes. S3 recovers; S1,S2,S3 elect S1 leader for term 3. S1 and S2 log entry for term 3, but log updates do not reach S3 before it crashes. S1, S2 crash. S3, S4, S5 recover and elect S5 leader for term 4 (all three replicas have only the entry for term 1 at the time of election!) S5 logs entry for term 4 and crashes. S1, S2 recover. S1, S2, S3, S4 elect S1 leader. S1 forces all followers to match its log.

## Concurrency control, logging, and recovery

5. [10 points] Answer True or False for these statements:

   A. **True** / **False** A cache is an example of volatile storage, while main memory is an example of non-volatile storage. Both are volatile

   B. **True** / **False** A NO-FORCE policy causes more work during REDO than a FORCE policy. NO-FORCE means that committed writes may not be on nonvolatile storage

   C. **True** / **False** Pessimistic concurrency control gets its name because it causes more aborted transactions. "Pessimistic" because it assumes conflicts

   D. **True** / **False** Two-phase locking is required when using write-ahead logging.

   E. **True** / **False** Hierarchical locking schemes are incompatible with ARIES.

6. [6 points] Consider the following excerpt from a simple write-ahead log. UPDATE records are of the form: UPDATE <var name>=<old value>; <var name>=<new value>. After entry 41, the system crashed.

| Entry # | Trans ID | Entry |
|---|---|---|
| 31 | 111 | UPDATE A=0; A=1 |
| 32 | 111 | UPDATE B=0; B=1 |
| 33 | 111 | COMMIT |
| 34 | 112 | BEGIN |
| 35 | 112 | UPDATE A=1; A=2 |
| 36 | 112 | UPDATE B=1; B=3 |
| 37 | 113 | BEGIN |
| 38 | 113 | UPDATE C=0; C=1 |
| 39 | 113 | UPDATE A=2; A=0 |
| 40 | 113 | COMMIT |
| 41 | 112 | UPDATE B=3; B=4 |

Line 39 shows transaction 113 modifying A, previously modified by uncommitted transaction 112 in line 35

   A. During the database's recovery process, which transaction(s) get undone? Circle one or more of the transaction IDs, or circle None if you believe none are undone.

   111 and 113 have committed by the time of the crash, but 112 is still in progress

   None 111 112 113

   B. **True** / **False** This log excerpt is consistent with a system that uses strict 2-phase locking (holds all locks until the end of a transaction).

   C. **True** / **False** The commit status of a transaction is determined by whether there is a COMMIT entry in the log, even in a system with a NO-FORCE policy

Initials:

## Two-phase commit

7. [6 points] Two servers A, C have modified data on behalf of distributed transaction T. Server B is acting as the coordinator for the transaction commit. The servers may fail, and they communicate via a network where both network failures and partitions are possible.

   In contrast to the system described in lecture, there are no timeouts. Accordingly, when a remote server is not responding there is no reliable way of determining whether it has crashed or is up but unreachable. The transaction T has reached its end and is trying to commit. The following actions have happened as part of the two-phase commit:

   i. B writes "prepare" to its log
   ii. B tells A to prepare
   iii. B tells C to prepare
   iv. C writes changes and "prepared" to its log
   v. A writes changes and "prepared" to its log
   vi. A tells B that A has prepared
   vii. C tells B that C has prepared
   viii. B writes "commit" to its log

Immediately after step viii, B crashes. Answer the next two questions assuming that B has not yet recovered. Pick the BEST response.

A. Can server A commit?

   a. Yes, because B has committed
   b. No, because A does not know the outcome
   c. Yes, because both A and C have prepared
   d. No, because any failure causes an abort

The transaction has committed as soon as B writes "commit." That remains true even if some or all participants are unable to communicate with B.

B. Can server C abort?

   a. Yes, because B has crashed
   b. No, because C does not know the outcome
   c. No, because both A and C have prepared
   d. Yes, because any failure causes an abort

This situation (participants who have prepared but can't contact the coordinator) is called the "window of vulnerability" in 2-phase commit.

C. Now server B recovers and re-establishes communication with servers A and C. Can server A commit now?

   a. Yes, after B informs A that the outcome is commit
   b. No, because the transaction already committed while B was crashed
   c. No, because the transaction already aborted while B was crashed
   d. No, because now the transaction aborts although it couldn't abort before

Initials:

## Passwords

8. [6 points] Consider the following schemes for storing username and password information. We consider a single line containing the fields that are needed for a single user's login lookup; the table as a whole consists of a large number of these lines.

   The different fields in the line are separated by commas (field1, field2, field3). "salt" is a randomly generated 10-digit number. Assume there are no issues with the random number generator. "|" means concatenate. The hash function is "slow" – it takes a substantial amount of computation.

   We want to defend against a so-called "rainbow table" attack, in which the adversary has already computed hashes of many common passwords.

   For each scheme, circle the correct answer to the question: Is this scheme resistant to rainbow table attacks?

   A. **Yes** / **No** username, salt, hash(password) | hash(salt)

   B. **Yes** / **No** username, salt, hash(password) | hash(password | salt)

   C. **Yes** / **No** username, salt, hash(hash(password) | salt)

   A rainbow table contains precomputed hashes for common passwords, so any field that consists of only "hash(password)" is vulnerable to matching.

   The scheme in A is obviously vulnerable — the hash(salt) is useless, since it's just concatenated. (Also, since the salt is provided in the entry, hash(salt) is known to the attacker)

   The scheme in B is vulnerable because the rainbow table allows the attacker to match on the first part of the stored value. Because only a small number of entries in the rainbow table will match the prefix of the stored value, the attacker has a much smaller set of candidate password values that can then be tested. Again, the salt is not providing much value.

   The scheme in C is less vulnerable, because no part of the stored value can be easily matched against hash(password). The stored value does not contain any part that matches hash(password). Although the stored value is derived from hash(password), the use of a salt and a second hash means that the attacker's rainbow table is of little value.

   [One common error in analyzing these schemes is to pay too much attention to how hard it is to generate something by brute force. Hash(salt) is infeasible to generate for every salt; likewise for hash(password I salt). But you don't have to be able to replicate every element of an entry to be able to attack the scheme.]

## Secure Channels

9. [10 points] Bob has built his own secure channel between his two servers A and B. He has an efficient, reliable, and secure encryption engine that can take 512 cleartext bytes and a secret key to produce 512 encrypted bytes. It can also take 512 encrypted bytes and a (matching) secret key to produce the original 512 cleartext bytes.

   Bob's applications always send each other messages that are exactly 512 bytes long, via some sensible and workable but otherwise-unremarkable transport protocol. His implementation has one secret key X that's used for traffic going from A to B, and a different secret key Y for traffic going from B to A. The secret keys X and Y are both "good enough" (long enough, random enough) and are simply hardcoded into the servers A and B.

   Assume a threat model in which the adversary can see any traffic on the network, can create messages on the network, and can destroy messages on the network. However, the adversary cannot compromise server A or server B, and cannot determine any secret key (X or Y) from any number of collected messages.

   A. **True** / **False** Bob's channel may be vulnerable to a replay attack by the adversary
   Neither side can tell the difference between an "old" message and a "new" message

   B. **True** / **False** Bob's channel may be vulnerable to a reflection attack by the adversary
   Different keys are used for the two different directions

   C. **True** / **False** Bob's channel may be vulnerable to the adversary pretending to be A or B.
   Adversary can't get secret keys, so can't spoof either side

   D. **True** / **False** Bob's channel is vulnerable to the adversary reading the application data (cleartext) of messages being sent between A and B.
   Adversary can't get secret keys, so can't read traffic

   E. **True** / **False** The adversary may be able to infer something about Bob's application from the communication pattern between A and B.

   Don't have to be able to read the traffic to be able to distinguish quantity of traffic or timing of traffic, and possibly correlate those with other observations (of Bob or his world)

Initials:

## DNSSEC

10. [6 points] You're talking with someone who read the DNSSEC paper but missed a couple of important ideas. Pick the SINGLE BEST correction for their not-quite-right claims.

    A. "With DNSSEC, there's no reason to be concerned about getting faked domain information when you're working in a coffee shop. The connection between your browser and the DNS server is protected, so you are safe from spoofing and interception attacks on DNS."

       a. Actually, TLS is secured by a completely different mechanism.
       b. Unfortunately, DNSSEC doesn't secure the browser-to-DNS communication.
       c. Yes, but first you have to take part in an elaborate key-signing ceremony.
       d. DNS contains public information, so spoofing and interception are not real threats.

    B. "DNSSEC uses encryption to hide server addresses from hackers."

       a. DNSSEC actually uses signatures for integrity
       b. DNSSEC actually hashes server addresses
       c. DNSSEC only encrypts the roots, not all servers
       d. Zone walking is actually how servers are protected

## Cryptosystem failure

11. [8 points] Circle true or false for the following statements:

    A. **True** / **False** Weak cryptographic algorithms are the primary problem with the security of systems deployed in the real world.

    B. **True** / **False** Back-door capabilities are a risk to security.

    C. **True** / **False** Secret algorithms and secret implementations strengthen the security of a system.

    D. **True** / **False** Trustworthy components assembled together guarantee a secure system.

Initials:

## Botnets, Blockchain, Bitcoin

12. [10 points] Marie has come up with an idea to combine botnets and blockchain. She figures that the current running Bitcoin system can be used as a coordination mechanism between bots and her C&C server. Each new bot will post its availability in a transaction on the Bitcoin blockchain so the C&C server can see it. Likewise, the C&C server will post commands for each of the bots in the same blockchain.

    Assume that Marie has written all of the code for the bots and the C&C server.

    Answer True or False for each of these claims about the resulting system.

    A. **True** / **False** A nonpermissioned blockchain (like Bitcoin) is unnecessary for this application, since Marie controls all of the potential writers.

    B. **True** / **False** If Marie's coordination scheme depends on her bots adding blocks directly (mining), it may be slow and unpredictable.

    C. **True** / **False** If Marie's scheme depends on sending transactions via other miners, she will have to pay them.

    D. **True** / **False** A long consistent public history is important whenever a bot is trying to coordinate with a C&C server.

    E. **True** / **False** Regardless of how well the coordination works, Marie's bots are likely to make her rich via Bitcoin mining.

Initials:

## Meltdown

13. [8 points] A colleague is proposing mitigation strategies for Meltdown in an architecture they are designing. For each proposal, choose the BEST response.

A. "We could eliminate speculative execution."

a. No Meltdown attacks would be affected by that change
b. That could help against Meltdown, but cost too much performance
c. That is a mitigating technique recommended in the published paper
d. That would fix the problem, and eliminate stack smashing too

B. "We could eliminate caching."

a. No Meltdown attacks would be affected by that change
b. That could help against Meltdown, but cost too much performance
c. That is a mitigating technique recommended in the published paper
d. That would fix the problem, and eliminate stack smashing too

C. "We could add a mode that splits the address space."

a. No Meltdown attacks would be affected by that change
b. That could help against Meltdown, but cost too much performance
c. That is a mitigating technique recommended in the published paper
d. That would fix the problem, and eliminate stack smashing too

D. "We could change the operating system to map in less of the kernel's address space."

a. No Meltdown attacks would be affected by that change
b. That could help against Meltdown, but cost too much performance
c. That is a mitigating technique recommended in the published paper
d. That would fix the problem, and eliminate stack smashing too

## Cheap Point

14. [1 point] Consider this to be our market research (again): We want to know which papers you particularly liked or disliked.

    Please pick papers that we read this term, and describe them in a way that we can recognize. Although the rest of the quiz is about material starting with GFS, you can tell us a paper from the first part of the term if you prefer.

    Feel free to comment further about why you like or don't like a paper, but first be sure you've completed all the previous questions.

    A. What was your favorite paper?

    B. What was your least favorite paper?

# End of Quiz 2

Please double-check that you have written your name and email ID on the first page and initialed all of the pages

Thanks for taking 6.033! Have a great summer!

Initials: