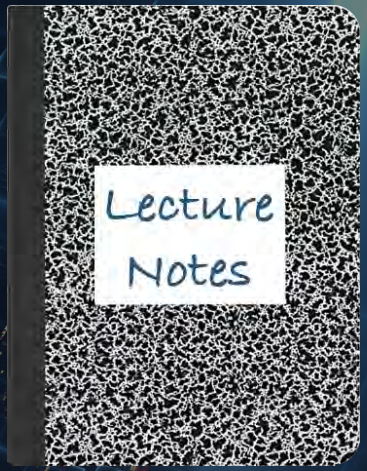
CS 417 – DISTRIBUTED SYSTEMS

Fall 2022 Exam 2 Review



Paul Krzyzanowski

Question 1

On average, which mutual exclusion algorithm creates the least amount of messaging traffic to get and release a lock?

(a) Centralized

(b) Token Ring

(c) Lamport

(d) Ricart & Agrawala

(a) *Request, Grant, Release*

(b) Constant circulation of a token per resource among all group members

On average, the token message will circulate through ½ of the group members before the node that wants it will get it

(c) *Request* to all group members, *Release* to all group members

(d) Request to all group members, ACK (possibly delayed), *Release* to all members

Question 2

Which mutual exclusion algorithm does Google Chubby's locking service implement?

(a) Centralized

(b) Token Ring

(c) Lamport

(d) Ricart & Agrawala

- Central lock manager & simple file system
- Fault tolerant via replicas and state machine replication

Question 3

Unlike the Ricart & Agrawala algorithm, *Lamport's mutual exclusion* algorithm:
(a) Requires the use of unique Lamport timestamps for each message.
(b) Forwards a lock request message from one group member to another until all members get it.
(c) Requires every group member to build a queue of processes waiting for a lock.
(d) Can make progress even if group members that don't hold a lock die.

- Every process
  - Receives timestamped *Request* & *Release* messages
  - Builds an identical queue of requests sorted by timestamps
  - Process ID at the head of the queue gets to access the resource
- In R&A
  - A process not interested in the resource simply sends an ACK to a request
  - A process queues requests only if it's using the resource
  - Competing requests are resolved based on the timestamp

Question 4

A leader is chosen in the *bully election* algorithm when:
(a) Every live process acknowledges an election message.
(b) Some process fails to receive responses to any election messages it sent out.
(c) An election message circulates through every process.
(d) A higher-numbered process responds to an election query from a lower-numbered process.

- A process sends *election* messages to every higher-numbered process
- If none respond, then the process decides it's the highest-numbered live process
  - Announces itself as the leader to the group

Question 5

The *ring election* algorithm terminates when:
(a) A process receives an election message containing a higher-numbered process ID.
(b) Every process accepts the result that was proposed in the election message.
(c) The highest-numbered process ID receives an election message.
(d) A process that started an election receives an election message with its own process ID at the head.

- Each process adds its ID to the list of processes that got an election message
- When a process sees its ID at the head of a received message, it knows that the message circulated through all live processes
- There may be concurrent elections taking place, started by other processes

Question 6

Raft divides its execution into a sequence of numbered terms. The purpose of a *term* is to:
(a) Act like a view in virtual synchrony and ensure that group membership is constant within the term.
(b) Allow the protocol to identify old leaders that recovered.
(c) Give every process a chance to serve as the leader.
(d) Determine that a network partition exists.

- A new term starts with each election
- Term #s make it easy to detect if delayed messages are received or an old leader restarts

Question 7

During an election in Raft, each server votes for:

(a) The candidate with the highest process ID of all the candidates that sent it a vote RPC.

(b) The candidate with the highest term number of all the candidates that sent it a vote RPC

(c) The first candidate that contacts it with a vote RPC.

(d) The last candidate it hears from before the election timeout period.

- A server votes for the first candidate that sends it a *vote* RPC
- BUT… messages with term #s ≤ the current term are rejected

Question 8

If a candidate in Raft failed to get votes from a majority of servers and does not get messages from a new leader, it:

(a) Assumes that the network is partitioned and remains silent.

(b) Queries other servers to find out which one received a majority of votes.

(c) Transfers its votes to another candidate so that the other candidate may emerge as the leader.

(d) Waits and starts a new election.

- If a candidate received votes from a majority of servers, it becomes the leader
  - Starts sending *AppendEntries* RPCs (heartbeats and log updates) with new term #
- If no majority vote & election timeout elapses without receiving *AppendEntries* RPC
  - Increment term #
  - Vote for self
  - Set random election timer
  - Send *ReuqestVote* RPCs to other servers

Question 9

One of the servers in Raft failed but recovered and may have missed some messages. It will:
(a) Tell the leader to pause message delivery to all other servers until it synchronizes and rejoins the group.
(b) Fail any *AppendEntries* RPCs it receives from the leader that contain a message index number that is too high.
(c) Request missing messages from random live servers, which distributes the message traffic across those servers.
(d) Contact the current leader to ask it to send the sequence of messages it is missing.

- Each *AppendEntries* RPC contains the index of the log entry prior to the data in this message
- If a receiver is not up-to-date, it rejects the message (return *false)*
  - Leader backs up and sends an AppendEntries RPC with an earlier entry

Question 10

A stateless file system service *can* correctly support a client:
(a) Locking a remote file so other clients cannot access it.
(b) Caching a remote file's contents consistently.
(c) Reading the latest data from a remote file.
(d) Continuing to access an open remote file after its permissions have changed on the server.

(a) Locking: requires the server to track that a file is locked and reject access from others

(b) Caching consistently: requires getting invalidations if a cache is invalid

- Just having a client check creates extra traffic & has a race condition

(d) Being able to continue accessing a file if the permissions changed means that the server needs to keep state that there's an open file

(c) This can be just a *read* RPC

Question 11

The architecture of AFS makes this not possible:
(a) File locking.
(b) Client caching of file contents.
(c) Multiple clients reading the same file.
(d) Concurrent file modification by multiple clients.

- AFS uses a stateful server & supports long-term client caching of file contents with callbacks from the server if the cached contents change

(a) File locking is supported – the client checks with the server upon opening a file

(b) Long-term caching at the client is central to AFS's architecture

(c) No problem with multiple clients reading – they each have a copy

(d) Concurrent modification is not possible – each client has its own copy and the last one to close overrides everyone else's changes by uploading their copy

Question 12

Coda's client modification log (CML) is used to:

(a) Identify files that need to be uploaded to servers.

(b) Determine which clients need to be sent cache invalidations by the server.

(c) Make it possible to undo changes to files.

(d) Keep track of which clients are modifying which files.

- The CML is used in disconnected operation
  - Log which files have been modified at the client
  - Upon reconnection, the CML is used to determine which files need to be uploaded to the server

(b) This can be an indirect result of a client uploading a file that is cached by others

Question 13

Microsoft's *oplocks* were created to:

(a) Control how clients can cache file data.

(b) Allow clients to lock a file for exclusive access.

(c) Enable servers to track which files are locked by clients.

(d) Keep other clients from reading or writing a file when it is being updated on the server.

- Created to enable caching only when it does not risk inconsistencies

Question 14

The architecture of the Google File System (GFS):
(a) Uses distributed hashing to locate the server hosting the file's metadata, which then points to file data.
(b) Uses a coordinator to assemble a hierarchical file system where each server hosts a node of the file system tree.
(c) Distributes the contents of individual files across many nodes but keeps file information centralized.
(d) Uses a central coordinator to determine which server should store the file's data and metadata.

- Central server for metadata
- Lots of chunkservers for file data
- Clients read and write data directly from/to chunkservers

Question 15

The Google File System (GFS) breaks file writes into two phases, *data flow* and *control flow*:
(a) To ensure the file is completely updated before any replicas are created.
(b) So that the control flow can identify which chunkservers receive the new file data.
(c) Because data flow is the writing of data and control flow is the update of metadata.
(d) To make the best use of network bandwidth when writing data and avoid locking the file for too long.

- Designed to make consistent, atomic file updates low overhead
- Data flow: propagate the data in a pipeline:
  client → replica-0 → replica-1 → replica-2 → …
- Control flow: add the data onto the file
  - requires locking so that all replicas get identical updates
  - Efficient since the data is already at each chunkserver that needs it

Question 16

Implementations of *consistent hashing* are based on:

(a) The hash of a key being an index into a table that contains linked lists of server addresses.

(b) The hash of a key matching the hash of the server ID or address.

(c) A hash of a key being used as an index into a table of server addresses.

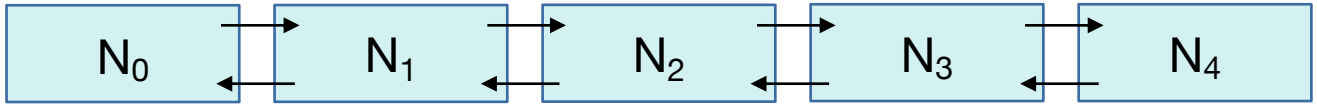
(d) A server being responsible for a range of consecutive hash values.

- Consistent hashing tends to use a large hash space (e.g., cryptographic hash functions)
- Each server is responsible for a range of hashes
- If a new server is added, it takes over part of a range from another server

Question 17

The average number of network hops in a one-dimensional content addressable network (CAN) with *n* nodes is:

(a) O(1)
(b) O($\log_2$ n)
(c) O($\sqrt{n}$)
(d) O(n)



- Each node is responsible for a hash range and knows two neighbors

```
if hash(key) < my_min
  forward query to left neighbor
if hash(key) > my_max
  forward query to right neighbor
```

- Locating the correct node is a linear search

Question 18

Suppose a Chord distributed hash table has 10 nodes and holds 100,000 items. On average, how many items will have to move when an additional node is added?

(a) 0.

(b) 5,000.

(c) 10,000.

(d) 50,000.

- With a good hash function, we expect uniform distribution across nodes
  - 10,000 items per node
- When a new node is added, we expect ½ of those items will have to move

Question 19

Virtual nodes in Amazon Dynamo:

(a) Allow more powerful computers to be configured to handle more data.

(b) Are replicas of nodes to provide fault tolerance in the case of node failure.

(c) Are placeholders for positions in the ring that do not have physical nodes assigned to them.

(d) Allow the query load of a single node to be distributed among multiple computers.

Virtual nodes have several advantages

- If a node fails, load handled by that node is evenly distributed among the remaining nodes
- If a new node is added, it can accept load from each of the available nodes
- The # of virtual nodes assigned per physical node can be based on the machine's capacity

Question 20

Which of the following is not an ACID property of transactions?
(a) Atomic
(b) Consistent
(c) Isolated
(d) Deadlock-free

- Atomic – Consistent – Isolated – Durable

Question 21

A problem with the two-phase commit protocol is that:
(a) A coordinator must contact a majority of participants to reach consensus.
(b) It does not support fail-recovery for participants or the coordinator.
(c) Every participant must wait if the coordinator dies during the protocol.
(d) If a participant changes its mind after submitting its vote, the protocol must restart.

(a) No. The coordinator must contact *all* participants

(b) No. The algorithm requires fail-recovery behavior

(d) A participant is not allowed to change its mind after submitting its vote

(c) The two-phase commit protocol is a blocking protocol. There is no provision for participants to abort rather than keep waiting even if it might be safe to do so.

Question 22

The CAP theorem tells us that:

(a) A system can tolerate partitions, be highly available, or provide consistent data, but not any combination of these.

(b) Consistency cannot be achieved reliably if a system is vulnerable to network partitioning.

(c) Systems must choose between consistency and high availability since partitions can always occur.

(d) A consensus algorithm can allow a system to achieve consistency and high availability even if partitions occur.

- The CAP theorem states that, in the event of a network failure (partition), a distributed data store can provide *either* consistency or availability — but not both

Question 23

Differing from the two-phase commit protocol, with the *three-phase commit protocol*, a recovery coordinator can:

(a) Query just one participant to know if everyone agreed to commit or whether no participant has yet committed.

(b) Query a majority of participants to determine whether there was agreement to commit.

(c) Abort the transaction without querying participants.

(d) Query every participant to find out if they agreed to commit, have committed, or need to abort.

- A recovery coordinator in a 2PC protocol must get the state from all participants
  - One might have received a commit/abort directive while others did not
- In 3PC, the second phase propagates the result of the vote to participants
  - If ANY participant says it received the vote results then ALL participants voted and SOME may have committed/aborted – continue with the protocol
  - If ANY participant says it did not receive the vote results then NO participants have received a commit/abort directive – it's safe to abort or re-run the protocol

Question 24

Strong strict two-phase locking differs from two-phase locking because it:
(a) Avoids the possibility of having other transactions read uncommitted data.
(b) Maximizes the amount of concurrency by reducing the duration that locks are held.
(c) Grabs all locks at the very start of a transaction to avoid deadlock.
(d) Uses mandatory locks instead of leases.

- SS2PL holds locks until the end of the transaction.
- 2PL guarantees a serial schedule but
  - If the transaction aborts after releasing its locks
  - Any transaction that read data created by the aborted transaction will have to abort

  ⇒ **cascading abort**

Question 25

The *wait-die* algorithm avoids deadlock by:

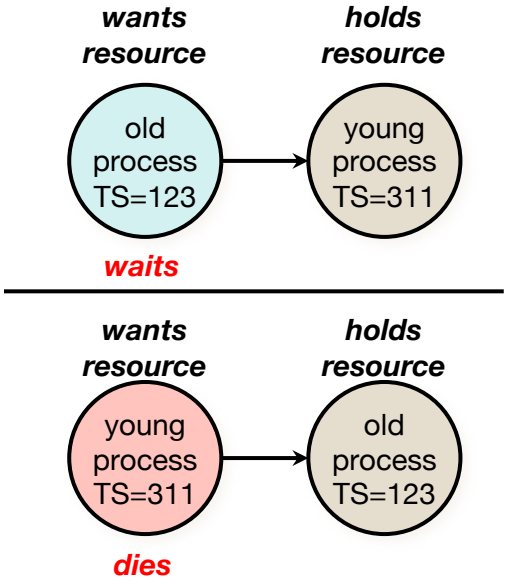
(a) Having a transaction wait for a lock or aborting if it detects a circular wait.

(b) Allowing only older transactions to wait for resources used by newer ones.

(c) Running one transaction at a time and having others wait in a queue until the transaction commits or aborts.

(d) Having a transaction set a time limit for how long it is willing to wait for a lock.

- If an older transaction wants a resource held by a younger one
  - Old transaction waits
- If a younger transaction wants a resource held by an older transaction
  - The young transaction kills itself



The end.