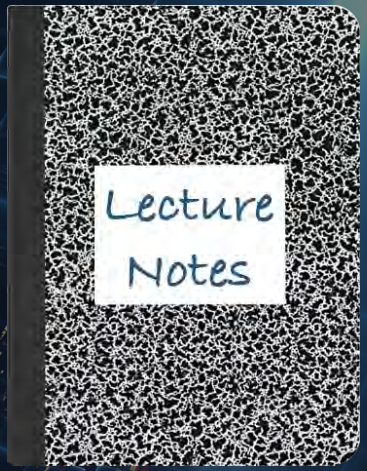
CS 417 – DISTRIBUTED SYSTEMS

Spring 2023 – Exam 2 Review



Paul Krzyzanowski

Question 1

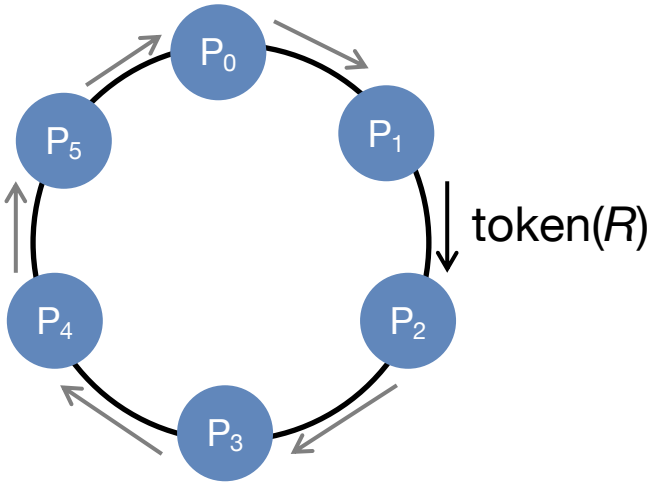
How does the *token ring algorithm* for distributed mutual exclusion work?

(a) A process passes a message to its neighbor that grants access to a resource.

(b) A central coordinator assigns tokens to processes based on their priority.

(c) Processes send requests to all other processes in the system to request a token for a resource.

(d) Processes use timestamps to determine the order in which tokens should be granted.



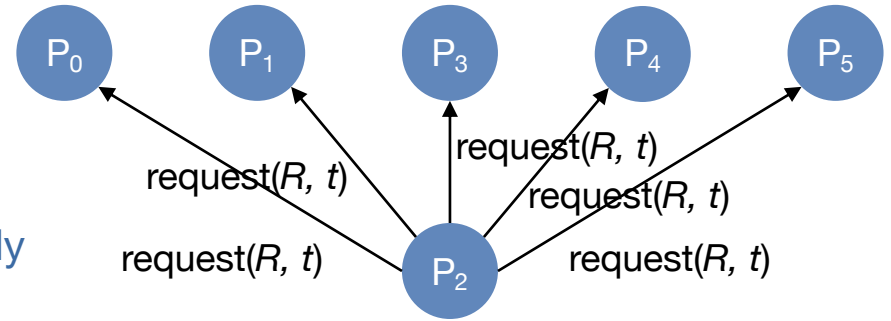
- A message, called a "**token**" for a resource *R* is passed from one system to its neighbor
- When a system receives the token for *R*:
  - If the process needs to access the *R*, it forwards the token when it's done
  - If it doesn't need to access *R* then it forwards the token immediately

Question 2

What is the primary communication pattern used in *Ricart and Agrawala's mutual exclusion algorithm*?

(a) Processes send requests to a central coordinator.
(b) Processes communicate directly only with the process holding the shared resource.
(c) Processes communicate only with their immediate neighbors in the system.
(d) Processes send requests to all other processes in the system.

- Ricart & Agrawala mutual exclusion
  - Send a request for a resource *R* (with a timestamp) to the entire group
  - Wait for everyone to respond with "ok"
  - If a process is using *R*, it will respond only when it is done



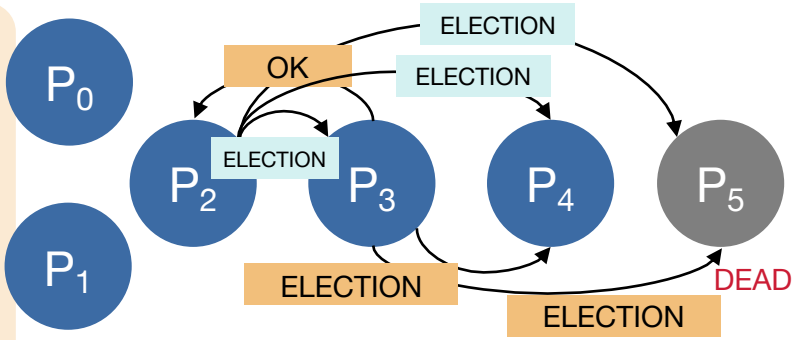
Question 3

Which of the following best describes the *Bully* election algorithm?

(a) Processes with higher IDs win over those with lower IDs to become the leader.
(b) Processes forward election messages to their neighbor to select a leader.
(c) Processes use timestamps to determine the order of election messages.
(d) Processes solicit votes from other processes and the one with the most votes wins.

Bully election algorithm

- If a process P_n detects a dead coordinator:
  - Send a message to all higher-numbered processes
  - If any of them respond, P_n will not be the leader
  - If none respond, that means P_n is the highest-numbered live process and announces itself as the leader
- If a process P_n receives an *ELECTION* message:
  - It holds an election by sending an *ELECTION* message to all higher-numbered processes.



Question 4

The *Chang and Roberts election algorithm* improves the Ring algorithm by:

(a) Having a process to skip over a non-responding neighbor when forwarding a message.

(b) Allowing a process to terminate an election if one is already being held.

(c) Not requiring an election message to circulate through all live processes to elect a leader.

(d) Using a multicast protocol to send an election message to all group members at once.

The *Ring algorithm* builds up a list of all live process IDs as an election message is forwarded from node to node

- When it gets back to the starting node, the node picks the highest process ID as the winner

In Chang & Roberts:

- Election message contains only one process ID (a higher process ID replaces a lower one)
- Avoids multiple elections by tracking if you're participating in an election AND the message you received isn't a higher process ID

Question 5

What is the primary goal of the *Raft consensus* algorithm?

(a) Load balancing requests among a set of processes.
(b) Reliable distributed data storage.
(c) Sending updates to a group of replicas in a fault-tolerant manner.
(d) Reliable data retrieval even in partitioned networks.

Raft is designed for state machine replication.

- It elects a *leader* that accepts all client requests
- The leader assigns a sequence # (log position) to each request and forwards it to all nodes.
- If a *majority* of nodes acknowledge the request, it can be committed (applied to the computation)
- Because of the need for a majority for elections and log updates, a network partition will not result in two leaders

Question 6

What is the minimum number of nodes in a Raft cluster required to tolerate a single node failure?

(a) 2 nodes.
(b) 3 nodes.
(c) 4 nodes.
(d) 5 nodes.

Raft (and other consensus algorithms) relies on ***quorum***: requiring a majority of nodes running for the algorithm to operate.

- For one node to fail, we need at least two to be running or else we cannot have a majority

Question 7

How does the *election protocol* in Raft avoid ties (split votes)?

(a) Requiring an odd number of processes in the system.
(b) Having each process wait a random amount of time before starting an election.
(c) Using logical clocks with totally-ordered timestamps in each message.
(d) Passing all messages through a coordinator.

Raft relies on randomized timers to minimize the chance that a group of nodes will start elections at the same time

- An election timeout is a random value (between 150-300 ms) during which a participant expects to get a heartbeat message from a leader.
- If it doesn't hear from the leader in that time, it starts a new election, hoping to get a majority of votes
- A random timeout increases the chance that one server will time out before the others and start an election, avoiding split votes (where there is no majority)

Question 8

An *AFS* file server must track the following state of clients:

(a) What files each client has open for reading or writing.
(b) Whether a client is running and connected to the network.
(c) What files each client has open for writing.
(d) What files each client downloaded.

AFS supports long-term caching:

A client downloads a file when it opens it and doesn't need to contact the server again unless it has to upload the file after it's been modified and closed.

A server must send *invalidation* callbacks to all clients that have downloaded copies of a file if a new version of the file is uploaded to the server.

Question 9

AFS was designed to scale to bigger environments than NFS. The primary way it did this was by:

(a) Enabling the contents of a file to be distributed across multiple servers.
(b) Using a protocol employing compound RPCs, so that each message may contain multiple operations.
(c) Switching to a TCP-based protocol instead of UDP.
(d) Allowing clients to cache files for a long time without checking for updates.

NFS required contacting the server for additional blocks of data from a file and checking with the server whether cached contents have been modified.

AFS avoided all interactions with the server after downloading a file except:

- Uploading a new version of the file to the server after the file's been closed.
- Receiving cache invalidations from the server if the server has a new version of the file.

Question 10

What is the primary advantage of the *Coda* file system compared to AFS?

(a) Better load balancing.
(b) Enhanced data compression.
(c) Stronger focus on scalability.
(d) Improved fault tolerance.

Coda added support for:

- Replicated read/write volumes
  - This can also help with (a) load balancing and (c) scalability but
    - Load balancing helps only with reading files; writes go to *all* volumes
    - Scalability applies only to load-balancing for read-only files; capacity is not improved
- Disconnected operation
  - If you cannot reach any servers, create a log of which cached files have been changed and will need to be uploaded to the servers

Question 11

Microsoft SMB *oplocks* (and *leases* in later versions of the protocol):

(a) Tell a client what data it can cache and whether it can delay sending updates to the server.

(b) Are a distributed mutual exclusion mechanism to allow a client to get exclusive access to a file.

(c) Allow a client to download a local copy of a file and enable the server to block anyone else from accessing it.

(d) Allow multiple clients to lock different parts of the same file.

SMB oplocks & leases control how a client can cache remote file data and attributes – they have nothing to do with locking

Question 12

Which is not a feature of the *Hadoop Distributed File System* (HDFS)?

(a) It requires a client library to access files.
(b) It uses a distributed hash table (DHT) to locate file contents.
(c) The contents of a single file may be spread across multiple servers.
(d) The protocol for writing file data supports n-way replication.

(b) There is no use of DHTs in GFS or HDFS.

(a) GFS & HDFS access is implemented via a library, not a file system driver.

(c) A file is made up of blocks (chunks), each of which is replicated and which are spread across blocknodes (chunkservers).

(d) Each block (chunk) can be replicated by a user-configured amount.

Question 13

Google's *Chubby* is not suited for providing a way for a group of computers to:

(a) Elect a leader.
(b) Obtain advisory locks.
(c) Subscribe to asynchronous event notifications.
(d) Store large files whose contents span multiple servers.

Chubby supports

- Leader election by having multiple processes try to grab an exclusive lock
- Locking of named resources (all locks are advisory)
- Event notification (file content changes and lock changes)

It is *not* designed for large files

- All data sits within a single server and is cached in memory.

Question 14

Which statement is not true about the differences between the Google File System (GFS) and Amazon Dynamo?

(a) GFS replicates data for fault tolerance while Dynamo does not but writes data to another server if one is down.

(b) GFS relies on a central coordinator while Dynamo is fully distributed.

(c) GFS allows appending data to an object while Dynamo requires updating the entire object.

(d) GFS supports a relatively small number of huge objects while Dynamo supports a huge number of small objects.

(a) Dynamo supports an optional # of replicas but is designed as an *always-writeable* data store. Writes can return without the replication completing.

(b) GFS relies on a GFS *master* – a central server that manages all file metadata.

(c) GFS supports appending to files that may be huge: multi-terabyte. Object stores only support complete overwrites.

(d) Because the GFS master stores all information about the file system & the main benefits of GFS are to have contents of a single file distributed, it is not designed for huge #s of files.

Question 15

Which of the following is a key feature of the *Content-Addressable Network* (CAN)?

(a) It uses a multi-dimensional coordinate space to store and locate objects.

(b) It creates a hierarchical structure for data organization.

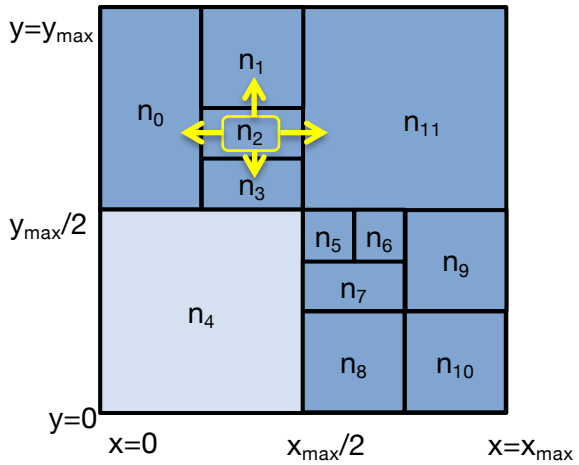
(c) It enables the rapid location of an object by sending a query to all peers on an overlay network.

(d) It uses consistent hashing to distribute the data associated with an object across multiple servers.

The key is hashed with a function for each dimension.

Each node is responsible for a range of values in each dimension.

If a hash(key) is out of range in any dimension, it forwards the request to a neighboring node.



Question 16

Differing from normal hash functions, *consistent hashing*:

(a) Produces a fixed-length value for any input.

(b) Ensures that each key will be hashed to a unique value so that collisions will not occur.

(c) Is a deterministic function that always produces the same result when applied repeatedly to a key.

(d) Reduces the number of keys that must be remapped if the size of the hash table changes.

Consistent hashing means that with *K* keys and *N* buckets, on average only *K*/*N* keys will have to be remapped

(a) All hash functions produce fixed-length results.

(b) Collisions are unlikely but, in theory, unavoidable.

(c) All hash functions are deterministic.

Question 17

Which describes the purpose of *finger tables* in a distributed hash table (DHT) system such as Chord?

(a) To enable each server to store the entire list of nodes in the group.

(b) To create a local hash table for the rapid access of objects stored within the node.

(c) To provide an efficient routing mechanism for key lookups without storing a list of all the nodes.

(d) To store a complete hash table on each node with all the keys and references to nodes where the data is stored.

Finger tables avoid the overhead of forwarding a query from neighbor to neighbor until the correct node is reacted: # of hops to find the node = $O(N)$

Finger table:

Each node stores a list of successor nodes where entry i in the finger table identifies a node that is 2^{i-1} nodes away. # of hops needed to find the node = $O(log_2 N)$

Question 18

How does Amazon Dynamo deal with the possibility of clients making *concurrent updates* to the value of an object?

(a) It uses mutual exclusion to provide an exclusive lock on the object while it is being modified.
(b) It uses state machine replication to ensure all replicas are updated atomically and consistently.
(c) It associates a vector timestamp with an object that is updated whenever a client updates the object.
(d) Client requests are funneled through a central coordinator that dispatches them serially.

Reads and *writes* return a *context*, which contains a vector timestamp.

If a *read* results in values that were updated concurrently, Dynamo gives the option of returning all concurrent versions to the application so the client software can reconcile the differences.

Question 19

What does the *CAP theorem* state about distributed systems?

(a) A system can achieve both consistency and availability, but not partition tolerance.
(b) A system can achieve consistency or availability but not both if partitions occur.
(c) Neither consistency nor availability is possible if partitions occur.
(d) A system can achieve consistency, availability, and partition tolerance, but at the cost of performance.

CAP Theorem: when there is a network partition, you cannot guarantee both availability & consistency

The CAP theorem tells us that we need to choose between designing a highly available system or a system that guarantees all data is consistent.

Question 20

How does the *three-phase commit protocol* (3PC) improve upon the two-phase commit protocol?

(a) By increasing the number of phases needed to get a vote on committing or aborting.

(b) By propagating the results of the vote to provide more opportunities for the protocol to abort rather than wait.

(c) By removing the need for a transaction coordinator.

(d) By enabling transactions to be committed with a majority consensus.

A phase was added to the 2-phase commit protocol to send the results of a commit query (a "prepare") to all participants so everyone will know the unanimous decision. This allows a recovery coordinator to get the state of the protocol by querying any participant and allows for more options to safely abort:

- A participant can abort in phase 1 if it doesn't hear from the coordinator.
- If the coordinator times out waiting to hear from a participant in phase 1, it can send an *abort* to everyone
- If the coordinator times out waiting to get an ACK from a participant in phase 2, it can send an *abort* to everyone because nobody else could have committed yet.

Question 21

Which ACID property ensures that a transaction is either fully completed or rolled back, without leaving the system in an intermediate state?

(a) Atomicity.
(b) Consistency.
(c) Isolation.
(d) Durability.

Atomicity – indivisible – the "all or nothing property"

Consistency – all invariants must hold

Isolation – cannot access intermediate results

Durability – results are made permanent

Question 22

In which scenario is the BASE transaction model typically preferred over the ACID model?

(a) When strong consistency is more important than high availability.
(b) When performance and high availability are more important than consistency.
(c) When the application requires distributed transactions.
(d) When the application requires strict isolation between transactions.

BASE = eventual consistency

Question 23

The purpose of two-phase locking is to:

(a) Ensure serial ordering even if transactions might read uncommitted data.
(b) Control the order in which transactions access resources to prevent deadlock.
(c) Provide a mechanism for rollback in case a transaction must abort.
(d) Get distributed consensus on who gets a lock.

Two phases: growing & shrinking = getting locks & releasing locks.

Ensures serial execution of transactions: one transaction will not modify data that another transaction is using.

BUT … transaction *B* may read data after transaction *A* released locks

If transaction *A* aborts instead of commits, then *B* will have to be aborted: *cascading aborts*

Question 24

What is the primary goal of Multiversion Concurrency Control (MVCC) in a database system?

(a) To improve write performance by allowing multiple transactions to modify the same resource simultaneously.

(b) To increase the database system's fault tolerance by storing backup copies of objects.

(c) To ensure transactions can always read data without blocking.

(d) To ensure that transactions always access the latest version of a resource.

MVCC provides **snapshot isolation**

We store multiple timestamped versions of data – one version for each committed transaction that made changes.

Transactions will read versions of data with timestamps earlier than when the transaction started, even if newer transactions have created new versions.

Question 25

How does the *wait-die* algorithm deal with deadlock?

(a) If a transaction detects a circular wait will occur, it waits, checks again, and aborts if it still cannot get the lock.

(b) A transaction waits on a lock with a timeout and aborts if it cannot get the lock within the allowed time.

(c) If a transaction needs a lock held by an older transaction, it aborts itself even if a deadlock would not have occurred.

(d) Transactions get leases instead of locks and must abort if the lease expires.

Wait-die ensures deadlock cannot occur by making the circular wait condition impossible:

- If an older transaction wants a resource held by a younger transaction, the older transaction waits
- If a younger transaction wants a resource held by an older transaction, the younger transaction aborts

The End