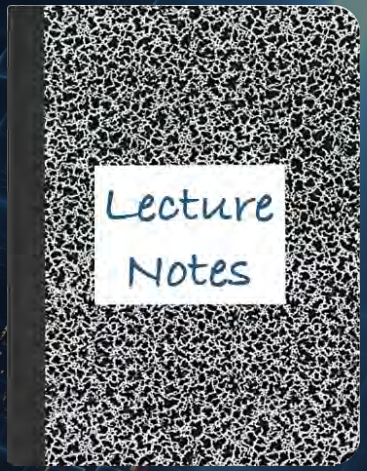
CS 417 – DISTRIBUTED SYSTEMS

Week 14: Exam 3 Review



Paul Krzyzanowski

Question 1

Bigtable stores a collection of tablets. Each *tablet* holds:
(a) A replicated instance of the table.
(b) A single row of a table containing a possibly unlimited number of columns.
(c) All the keys of the table where *hash(key)* maps onto a specific node.
(d) A contiguous, sorted set of complete rows of the table.

- A table is broken up into tablets
- Each tablet contains a contiguous range of rows, sorted by the row key
- Tablets are the unit of distribution and load balancing
- A server manages a collection of tablets
- Picking a good row key enables clients to exploit locality for data access

Question 2

Bigtable does *not* rely on:
(a) SSTable files.
(b) Distributed hash tables.
(c) Google File System (GFS).
(d) Chubby.

- SSTable is a file format used to store tablet data. It maps keys to values, where each can be arbitrary byte strings
- Bigtable tablets and log files are stored in GFS
- Chubby is used to
  - Store the location of the top-level (root) tablet of an instance of Bigtable
  - Discover tablet servers & their deaths
  - Store schema information (column family information)
  - Store access control lists

Question 3

Cassandra is a *wide-column* database. This implies that:
(a) The contents of a column within a row may be distributed among multiple servers.
(b) A column may contain large amounts of data, such as JSON objects or images.
(c) The database contains more columns than rows.
(d) The set of columns can vary from one row to another.

(a) Like Bigtable, all data for a row is stored together.

(b) True, but this applies to regular databases as well.

(c) Not necessarily.

In a wide-column database, the names and data types of columns in a wide-column database can vary from row to row.

Question 4

The Cassandra database distributes data among multiple servers such that:

(a) The hash of a row's partition key determines the server where the row will be stored.

(b) When a table gets too large, it is split and a coordinator tracks which range of rows are on which server.

(c) The columns of each row form a distributed hash table.

(d) Each server holds one or more complete tables and a hash of the table name determines the server.

- The partition key is used for balancing data across the cluster
- The hash of the partition key identifies the assignment of a row to a node.
- (b) Tables don't get split and rows with the same partition key are always located on the same node.
- (c) No – the DHT is based on the partition key; columns in a row stay together.
- (d) Tables can be distributed by partition key and table names have no bearing.

Question 5

A *commit-wait* in Spanner differs from a conventional database commit because the commit-wait waits until:
(a) All earlier transactions commit.
(b) All locks used by the transaction are released.
(c) The commit timestamp is definitely in the past.
(d) All sub-transactions across multiple servers are ready to commit.

- Spanner provides *external consistency* – transaction timestamps are consistent with global time
  - Any observer will not see the results of the transaction until AFTER its timestamp
- To ensure this, a *commit-wait* delays until the transaction timestamp is definitely in the past before releasing locks and completing the transaction.

Question 6

Spanner does *not* employ:
(a) Multiversion concurrency control.
(b) Eventual consistency.
(c) Two-phase locking.
(d) Wound-wait concurrency control.

(b) Spanner provides strong consistency, not eventual consistency

(a) Spanner keeps timestamped versions of data
- This allows transactions to read without blocking future writes
- Writes create new versions with a timestamp of that *write*'s transaction

(c) Spanner uses two-phase commit and two-phase locking to ensure isolation and strong consistency
- It uses replicated state machines (via Paxos) to improve availability.

(d) Spanner uses wound-wait to prevent deadlocks
- Earlier (older) transactions can abort younger ones.
- This ensures that every transaction eventually has a chance to acquire locks

Question 7

Unlike Cassandra, why does Bigtable not replicate tablets?
(a) High performance was the primary design goal.
(b) Bigtable favors consistency over high availability.
(c) Replication is handled by the underlying file system.
(d) The SSTable storage structure uses error-correcting codes.

- Bigtable was built to store data in the Google File System, which provides replication

Question 8

In MapReduce, a *reduce* worker starts processing when:
(a) At least one *map* worker generated some output for that *reduce* worker.
(b) At least one *map* worker has completed its work.
(c) Every *map* worker has completed its work.
(d) Every *map* worker has begun processing data and generating output.

- A *reduce* worker needs to get all the (*key, value*) data that it will need to process
- Until every *map* worker has finished processing data, it is not known if there will be another (*key, value*) pair emitted for that *reduce* worker

Question 9

In MapReduce, the user's *reduce* function is called once for every:
(a) Unique intermediate key, and no two *reduce* workers get the same key.
(b) Unique (*key*, *value*) pair generated by the *map* workers.
(c) Set of keys generated by *map* workers that hashes to the same partition.
(d) Unique key per *reduce* worker, and multiple *reduce* workers may get the same key.

- Map workers process input data and generate (*key, value*) pairs
- The *reduce* function is called once for each unique key and is provided a list of all the values associated with that key.

Question 10

A *map* worker generates output that is:
(a) Written to a temporary local file.
(b) Sent to the appropriate *reduce* worker.
(c) Appended to a shared file for each partition that will be read by the *reduce* worker for that partition.
(d) Appended to a single shared file that will be read by every *reduce* worker.

- *Map* workers store (*key, value*) data in files on the local disk.
- The locations are passed to the *master*, who then tells the *reduce* workers
- *Reduce* workers use RPCs to read the data from the *map* workers.

Question 11

A job in Pregel *terminates* when:

(a) No vertex has sent out any messages in the previous superstep.

(b) Each vertex completed its computation and no edges have been modified.

(c) Every vertex votes to halt.

(d) Every vertex votes to halt and no messages were sent in the previous superstep.

- Each vertex can vote to halt when it has no more work to do.
- The job terminates when all vertices have halted.
- However, if a vertex receives a message from another vertex, it is reactivated and runs again. It has to explicitly vote to halt again.

Question 12

If a worker in Pregel fails and the master reassigns its vertices to new workers:
(a) All workers are paused while the new worker resumes from the last checkpoint.
(b) All workers continue to run and the new worker resumes from the last checkpoint.
(c) All workers resume from the previous superstep and the new workers resume from the last checkpoint.
(d) All workers must restart from the last checkpoint.

- A checkpoint is performed every N supersteps
  - It means saving the state of each vertex computation and all messages.
- Since the recovered vertices may generate messages to vertices on workers that didn't fail, ALL vertices must be restarted from the checkpointed superstep.

Question 13

What is meant by Spark *transformations* being *lazy*?

(a) Transformations create a new dataset from an existing one.

(b) A transformation is executed only when its result is needed.

(c) A transformation is executed in parallel because an RDD is partitioned across multiple workers.

(d) Transformations only process the subset of data that is needed to generate the RDD.

- A transformation is not executed immediately
- Spark builds a graph (DAG – directed acyclic graph) of the sequence of transformations
- Spark works backwards from actions, invoking transformations when it needs to generate the necessary RDDs

Question 14

Which of the following is *not* a feature of Spark RDDs (resilient distributed datasets) created by transformations?

(a) They are replicated for fault tolerance.

(b) A single RDD can be read by multiple future transformations or actions.

(c) They may be cached in memory.

(d) They can be partitioned across a set of machines.

- Fault tolerant storage of RDDs is not needed
  - Except for the initial input data, intermediate RDDs can always be regenerated by applying the necessary transformations

Question 15

In Kafka, a *consumer group* enables:

(a) A message queuing model where multiple consumers read unique messages from a queue.

(b) A publish-subscribe model where multiple consumers share the same messages.

(c) High reliability by replicating partitions of message queues across servers.

(d) Support for multiple message queues, each identified by a different topic.

- Without consumer groups, all consumers (subscribers) will read the same sequence of messages (this is choice b)
- Consumers in the same consumer group will get unique messages from their subscribed topics
  - Consumer groups allow the message processing to be load balanced among multiple nodes

Question 16

Akamai's overlay network is designed to:
(a) Route user requests to the correct content servers at the edge.
(b) Handle broadcast queries to locate servers that have the desired cached content.
(c) Reach origin servers more efficiently than relying on Internet routing.
(d) Serve as a backup in case the Internet fails.

- Dynamic DNS is used to identify the nearest edge servers.
- The overlay network is used to identify more efficient routes from a caching server on the edge to the origin server
  - Performance rather than hop count or cost
  - Persistent TCP connections among Akamai server in the overlay network to avoid connection setup overhead

Question 17

Akamai uses dynamic DNS (Domain Name System) servers to:
(a) Provide different results based on where the query is coming from.
(b) Provide an optimal path to the origin server.
(c) Look up the address of origin servers that host the desired content.
(d) Rewrite the user's URL into a URL targeting an Akamai server.

- Akamai's DNS (and most other CDNs too) takes the source IP address into account when resolving a domain name query
- Returns an edge server that is
  - Closest (in network performance) to the user
  - Available
  - Not overly loaded

Question 18

A *symmetric* encryption algorithm:

(a) Produces the original message if the encrypted message is encrypted again with the same key.

(b) Uses the same key to encrypt and decrypt a message.

(c) Encrypts or decrypts a message based on whether an encryption or decryption key is provided.

(d) Results in a bit sequence that is identical in either direction.

- Symmetric ciphers use the same key to encrypt and decrypt a message
- Public key ciphers use one key to encrypt and another key to decrypt a message

Question 19

To send a message securely to Bob, Alice will encrypt it with:
(a) Alice's private key.
(b) Bob's private key.
(c) Alice's public key.
(d) Bob's public key.

- We assume public keys are not secret (that's why they're called *public*)
- Alice and Bob do not share their private keys with anyone.
- Anything encrypted with Bob's public key can *only* be decrypted with Bob's private key.

(a) Then anyone can decrypt the message with Alice's public key
(b) Then anyone can decrypt the message with Bob's public key
(c) Only Alice will be able to decrypt the message with her private key

Question 20

Diffie-Hellman enables:
(a) Efficient encryption of messages using exponents and modulus operations.
(b) The creation of an encryption and a related decryption key.
(c) A process to encrypt a session key using public key cryptography and send it to another process.
(d) Two processes to compute a number that other processes cannot compute.

The Diffie-Hellman algorithm is used to allow two parties to generate a common key

- Each side creates public & private numbers
- The common key for A & B can only be generated by using the {public # of A and private # of B} or the {public # of B and private # of A}
- Diffie-Hellman is NOT an encryption algorithm – it only allows two sides to compute a key (which is just a number)

Question 21

A Message Authentication Code (MAC) differs from a hash because:
(a) It is encrypted with the recipient's public key.
(b) It identifies the sender of the message.
(c) It produces a variable-length result.
(d) It is a function of a secret key as well as the message.

- A MAC is a *hash(message, key)*

Question 22

Which authentication technique does not rely on a shared secret between the client (user) and server?
(a) PAP (Password Authentication Protocol).
(b) CHAP (Challenge Handshake Authentication Protocol).
(c) Passkeys.
(d) TOTP (Time-based One-Time Passwords).

(a) The client and server must both store the password
(in many implementations, the server may store a hash of the password).

(b) In CHAP, the client and server both compute *hash(challenge, secret)*.

(d) In TOTP, both sides compute a *hash(time, secret)*.

(c) Passkeys use public key cryptography. The server stores the public key for the user and sends the user a random challenge string. The user sends back a digital signature for the challenge (hash encrypted with the user's private key).

Question 23

A *digital signature* is:

(a) A hash of a message that can only be created by the signer but can be validated by anyone.

(b) A secure way of associating a public key with a user's identity.

(c) A timestamped record that contains a timestamp, message authentication code, and a user identifier.

(d) A user's name or identification encrypted with their private key.

- A digital signature effectively a hash of a message encrypted with the user's private key.
- With MACs, all parties must know a shared key.

Question 24

An X.509 *digital certificate* can be used to:
(a) Store the certificate owner's public and private keys.
(b) Generate digital signatures.
(c) Verify a digital signature.
(d) Decode messages sent to the owner of the certificate.
(e) All the above.

- A digital certificate is a way to associate an identity with a public key in a way that cannot be forged
  - The data structure contains a digital signature created by the certification authority
- It's not secret
  - (a) It does not contain private keys
  - (b) You need a private key to generate a digital signature
  - (d) You need a private key to decode messages sent to the owner of the certificate

Question 25

OpenID Connect was designed to allow:
(a) Multiple users to share the same login credentials for a service.
(b) A user to use the same login and authentication across multiple services.
(c) A service to call functions on another server using the user's credentials.
(d) Multi-factor authentication to a service.

- OpenID Connect allows a server to delegate authentication to another service
  - E.g., "Sign-In with Google"