Distributed Systems

20. Exam 2 Review

Paul Krzyzanowski

Rutgers University

Fall 2014

Question 1

NFS was initially designed as a stateless file system. The design principle of many REST-based web services today is to keep the servers stateless. Explain why server state is useful in a network file system.

Server state allows:

- Support for file locking (server keeps track of client locks)
- Client to keep a file open so it doesn’t get get deleted on the server (server keeps state that a given file is in use).
- Server keeping track of which clients need to be informed for cache invalidations.

NOT – *server state allows caching*
NFS did client caching without state, using validation. However, it wasn’t always consistent. State allows for server-initiated cache invalidations

No broad answers, such as “shorter requests”, “faster performance” or “improved fault tolerance” without explaining your answers.

Question 2

What is a false deadlock and under what conditions could a false deadlock arise?

Improper message ordering when reporting a wait-for graph can result in a coordinator thinking that a circular dependency, and hence deadlock, exists when that is really not the case.

Not: a host with a lock dies and then recovers.

Question 3

Explain what happens in the first phase of a three-phase commit protocol.

The coordinator gets everyone's decision on whether they can commit or have to abort.

More precisely:

(a) Coordinator sends *canCommit?* queries to all participants

(b) Waits for responses from everyone

(c) If timeout on waiting for a response

- Coordinator decides that the transaction must abort.

Question 4

You have a collection of stock market summary data in the form of:

{ *date, company, start_price, end_price* }

where the *start_price* is the price of the stock at the start of the trading day and the end_price is the price of the stock at the end of that day.

For example: { 2014-10-29, AAPL, 106.83, 107.34 }.

You want to find out the average daily gain (or loss) of each company from January 1, 2000 through December 31, 2014.

Explain the MapReduce operations needed to accomplish this.
Use C/Java style pseudocode. Feel free to use <= and >= comparisons for dates and assume that functions such as *average()* are available.

Assume that the *map* function is called once for each parsed record of data and a function called *emit()* exists in the form *emit(key, value)*. For *reduce*, assume that a *print* function exists that prints results. For example, *print(name, result)*.

```
map(date, company, start_price, end_price) {
    if (date >= "2000-01-01" && date <= "2014-12-31")
            emit(company, end_price –start_price)
}
```

```
reduce(key, list) {
        /* key == company, list == gain list */
        print(key, average(list))
}
```

Question 5

]Which of the following is an example of an impure name?
(a) Ethernet MAC address.
(b) Email address.
(c) Phone number.
(d) Facebook user name.

- Impure name contains info about the context of a name
  - Path, location, other attributes

Question 6

In virtual synchrony, if process A fails to contact another process, B, in the group:
(a) Process B is removed from the group but only if it is truly dead.
(b) Process B is removed from the group even if it is alive.
(c) Process A will try to route the message through another group member.
(d) Process A will revoke all copies of messages that were sent to other group members.

- We can't tell for sure whether a process is really dead
  - Two-army problem
- If one process believes that another process is dead,
  It conveys that information to the Group Membership Service (GSM)
- The GSM propagates the view change to all members
- If the process was really alive, it will have to rejoin

Question 7

A flush message in virtual synchrony received by a process causes that process to:
(a) Send any unconfirmed messages to members of the group to be sure they have them.
(b) Delete any messages that have been received but not yet delivered to applications.
(c) Delete any messages that have been queued for sending but not yet sent.
(d) Write in-memory data to disk to create a checkpoint for recovery.

- A *flush* is a barrier
  - All messages where group delivery has not been acknowledged (confirmed; hence marked *stable*) need to be delivered to & acknowledged by all group members
  - Other group members that received a message that is not marked as *stable* may take over the sending in case the original sender may have died before all group members received the message

Question 8

Which of the following is not an ACID property of transactions?
(a) The transaction appears as an indivisible action.
(b) The transaction cannot leave data in an inconsistent state.
(c) Once a transaction commits, the results are made permanent.
(d) The results of a transaction must eventually be made consistent across replicas.

- ACID =
  - Atomic: others see the transaction results as a single indivisible action
  - Consistent: any system invariants must hold after the transaction
  - Durable: Failures after a commit will not cause a transaction to revert
- If replicas use ACID, all updates must be atomic
  - There should never be a case where one replica will hold older results than another one.

Question 9

The three-phase commit protocol improves on the two-phase commit protocol in that it:

(a) Allows each participant to acknowledge the completion of a commit.

(b) Ensures that a majority of participants will agree to commit or abort.

(c) Sends a commit or abort request to every single participant.

(d) Enables the use of a recovery coordinator.

- The 3PC introduced
  - Timeouts to avoid indefinite waiting in some cases
  - Propagation of abort/commit decision to all participants
- This allows a recovery coordinator to query participants
  - If *any* participant received the abort/commit decision, then:
    - ALL participants must have voted
    - SOME may have committed; we can finish the job
  - If *any* participant did not receive an abort/commit decision, then:
    - NO participants have committed
    - SOME may have not received the decision; we can start from phase 1.

Question 10

The three-phase commit protocol is not resilient against:
(a) Synchronous environments.
(b) Asynchronous environments.
(c) Fail-stop environments.
(d) Fail-recover systems.

- Original coordinator recovers & reads its writeahead log
- Resumes the protocol
  - BUT a recovery coordinator may have already completed the protocol!

Question 11

When scaling systems, it is common to give up on:
(a) Availability of the service.
(b) Having the system tolerate partition failures.
(c) Atomicity of changes.
(d) Data consistency across systems.

(a) We want high availability

(b) We need to handle network partitioning

(c) We don't want clients to see intermediate state

BUT
(d) Reading slightly old versions on some servers can be ok

This is the premise of BASE: *basic availability, soft-state, eventual consistency*

Question 12

Which is not a necessary role in Paxos?
(a) Client (or Acceptor)
(b) Proposer
(c) Learner
(d) Leader

- The Leader is just an elected proposer that gets all client requests.
- It's an optimization of Paxos to ensure that we have unique, incrementing sequence numbers and avoids having proposals get rejected because of bad numbering

Question 13

A leader in Paxos:
(a) Is the proposer that processes all incoming requests.
(b) Load balances incoming requests among all proposers.
(c) Assigns roles of proposer, acceptor or learner to Paxos processes.
(d) Propagates the consensus value out to servers.

- The leader is an elected proposer.
- All requests from clients go to the leader
- It sends a ***prepare*** request to the set of acceptors
  - Each acceptor will promise to ignore all smaller requests
  - If a majority (**quorum**) of acceptors promise to accept the request
    - That means they agree that this is truly the next request
    - Leader sends an **accept** message with the value

Question 14

To survive the simultaneous failure of P acceptors, Paxos requires:
(a) P^2 acceptors.
(b) *P+1* acceptors.
(c) *2P* acceptors.
(d) *2P+1* acceptors.

- We need a quorum (majority) of live acceptors
- If P fail, that means we need at least P+1 to be alive
- Total acceptors in system = P + P + 1 = 2P + 1

Question 15

With two-phase locking, a transaction:

(a) Obtains locks in the first phase and releases them in the second.
(b) Uses a two-phase commit protocol to obtain locks.
(c) Must get everyone's consensus to release a lock.
(d) Checks if the lock is available in the first phase and gets it in the second.

- Two-phase locking ensures that transactions are serialized.
- Phase 1: obtain locks
  - Do work
- Phase 2: release locks
  - Do work but you cannot obtain any more locks
  - Commit

Question 16

Two-phase locking:
(a) Ensures that a transaction does not read uncommitted data from another transaction.
(b) Uses separate read locks and write locks.
(c) Ensures that transactions are serialized (isolated).
(d) Uses a two-phase protocol to grab a lock reliably.

- Two-phase locking ensures transactions are serialized
- If a transaction T_2 needs to lock a resource but T_1 has a lock, it will have to wait until the lock is released
  - Any resources that T_2 accesses that T_1 uses will be accessed after T_1 releases the locks
  - BUT 2PL does not guarantee no deadlock
    - T1 may lock R_1; T_2 may lock R_2.
    - Then T_1 may try to get a lock for R_2 and block
    - T_2 may try to get a lock for R_1 and deadlock!
- Also … danger of cascading aborts!

Question 17

Deadlock cannot occur if:
(a) Each resource is given to only one process at a time.
(b) A process is allowed to wait until a resource it needs is available.
(c) Two-phase locking is used.
(d) A process can access resources that another process is using.

- Conditions for deadlock
  - Mutual exclusion: a resource can be held by only one process at a time
  - Hold & wait: processes holding resources can wait for other resources
  - No preemption: if a process holds a resource, the resource cannot be taken away
  - Circular wait: processes are waiting for resources held by one of the other processes
- (d) introduces preemption

Question 18

The Chandy-Misra-Hass algorithm deals with deadlock by having a process:
(a) Send probe messages to processes that hold resources it needs.
(b) Send its resource utilization to a central server to assemble a global wait-for graph.
(c) Not wait on resources that a younger process is using.
(d) Kill a process that is using a resource that it needs.

- This is called Edge Chasing

Question 19

The wait-die algorithm ensures there can be no deadlock by:
(a) Having a process kill itself if it waited more than a certain amount of time.
(b) Testing to see if there is a cycle of resource dependencies.
(c) Disallowing a young process to wait for an old one.
(d) Giving a process the option to wait or kill itself.

- if a younger process is using the resource
  - then the older process (that wants the resource) waits
- If an older process is holding a resource
  - then the younger process that wants the resource kills itself
- Cycles are impossible
  - This forces the resource utilization graph to be directed from older to younger processes

Question 20

NFS's validation refers to:
(a) The server sending invalidation messages when content has changed.
(b) The client comparing received data with a checksum to guard against corruption.
(c) The client comparing file modification timestamps from the server against those of its cached data.
(d) The client authenticating with the server.

(a) NFS was designed as a stateless file system

- Server cannot send invalidation messages – it doesn't know who has what cached

(b) Validation is not for checking for data corruption

(d) Validation is not for authentication

Question 21

Coda improves on AFS by adding:
(a) A stateless server for high performance.
(b) Remote access semantics instead of whole file downloads.
(c) Replicated read/write volumes.
(d) File locking support.

- Coda added support for
  - Disconnected operation
  - Replicated read/write volumes

Question 22

Microsoft SMB's oplocks (opportunistic locks):
(a) Control how a client can cache file data.
(b) Allow clients to lock regions of a remote file.
(c) Ensure that only one client can access a file at a time.
(d) Enable a server to lock client operations if a file is being modified.

- Microsoft's oplocks tell a client how it can cache data to ensure coherency
  - Similar to DFS's tokens
- They do not control file locking
  - They are NOT commands that are send from the client to server
  - You need to send

Question 23

Credit-based flow control:
(a) Enables a server to control the rate of messages from each client.
(b) Provides a billing infrastructure to charge clients for server use.
(c) Is a mechanism to balance load among multiple servers.
(d) Notifies clients of file modifications so they can update their caches.

- Server creates a small number of credits
  - Later increases this number as needed
- Server sends these credits to each client
  - The client needs credits to send a message to the server
  - Each time a message is sent, it decrements its credit balance
- This allows server to control the rate of messages from any client and avoid buffer overflow

Question 24

Notification servers were added to Dropbox to:
(a) Enable clients to notify Dropbox servers that changes have been made.
(b) Have blockservers notify metadata servers of changes.
(c) Allow Dropbox to work with mobile devices.
(d) Reduce database query load by not having clients poll for changes.

Server-initiated notifications alleviate the load of having the client continually poll for updates

Question 25

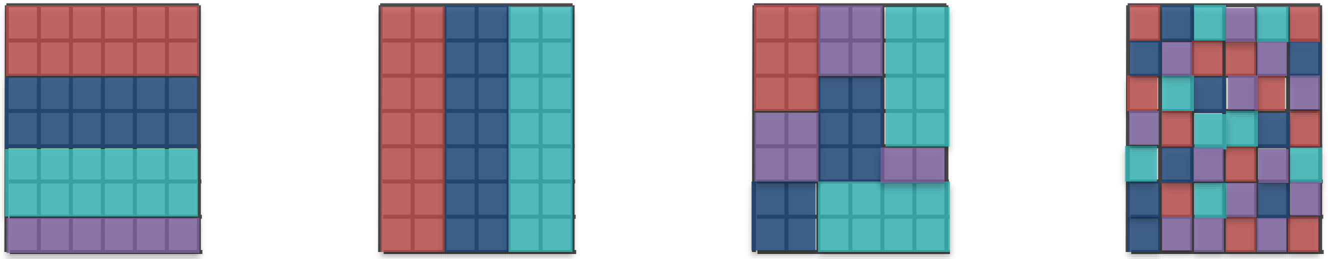
A GFS master:
(a) Is a central server that stores filenames along with their chunk IDs.
(b) Stores all file contents, which are replicated among chunkservers for redundancy.
(c) Does not store any information about files but is responsible for tracking the health of servers.
(d) Does not store any information about files but all operations pass through it to ensure total ordering.

- GFS master
  - Stores file metadata: file names, attributes, and chunk IDs that correspond to a file
  - It is similar to inode data in a file system
- File content is stored on *chunkservers*
  - ON startup, each chunkserver sends the GFS its list of chunk IDs

Question 26

In Bigtable, the table may be split across multiple servers:
(a) By rows, with contiguous ranges of rows placed on different servers.
(b) By columns, with contiguous ranges of columns placed on different servers.
(c) Both rows and columns, with a range of rows and subset of columns on different servers.
(d) With each table cell on an arbitrary server and a master coordinating access.

- A table is broken up by rows into **tablets**
- Each tablet server manages multiple tablets
- A tablet contains a sequence of consecutive rows and all column families



Question 27

A reduce worker starts to process data when:
(a) At least one map worker has completed.
(b) The majority of map workers have completed.
(c) All of the map workers have completed.
(d) At least one map worker has generated a <key, value> pair for that reduce worker.

- ALL map workers must complete first
- Only then do we have all (key, value) data for the reduce workers
- The framework:
  - sorts by key
  - merges all data with the same key to create a single (key, value_list)
  - feeds that to the *reduce* function

The end