Practice questions

416 2017 W2 (Winter 2018)
• True/False: In a practical system that uses proof-of-work, the checking of the proof has to be easy relative to the generation of the proof.
PQ 1

• True/False: In a practical system that uses proof-of-work, the checking of the proof has to be easy relative to the generation of the proof.

• True!

  • (If not, your system’s performance has to scale with the number of CPUs that use it)
PQ 2

- You are designing a protocol stack. You have narrowed down your design to two choices. And, you know that the specification for protocol C is likely to change. Which stack design should you use?

(a) protocol A
(b) protocol B

/-protocol C
  protocol B
  protocol A/-
You are designing a protocol stack. You have narrowed down your design to two choices. And, you know that the specification for protocol C is likely to change. Which stack design should you use?

- If protocol C changes then only protocol B would need to adapt, and not A
PQ 3

- A network element can inspect any of the protocols present in the packet. So, **why not** build e.g., a switch that is aware of HTTP and have it route packets based on HTTP information that it can extract from the packet?
PQ 3

• A network element can inspect any of the protocols present in the packet. So, **why not** build e.g., a switch that is aware of HTTP and have it route packets based on HTTP information that it can extract from the packet?

• Expensive! Line-rate HTTP processing. Also requires interpreting all the protocol below HTTP (e.g., TCP/IP)

• Higher-level protocols change, often more frequently (HTTP 2.0)

• More software can access/manipulate HTTP content (not just your OS TCP/IP stack). Requires more robustness/more security considerations.
PQ 4

• Which of these design scenarios contain elements of fate sharing?

1. Authenticating an ATM card by requiring a retina scan

2. Placing a passenger’s checked baggage on the same flight as the passenger

3. Attaching a spare car key to the inside of the car’s bumper
PQ 4

- Which of these design scenarios contain elements of fate sharing?

1. Authenticating an ATM card by requiring a retina scan

2. Placing a passenger’s checked baggage on the same flight as the passenger

3. Attaching a spare car key to the inside of the car’s bumper

Fate sharing: lose state information for an entity if and only if the entity itself is lost.
Fate sharing: lose state information for an entity if and only if the entity itself is lost.

• Which of these design scenarios contain elements of fate sharing?

1. Authenticating a debit card by requiring a retina scan
   • Not really. People with eye injuries may want to use debit cards. And, debit card loss does not result in losing your retinas!

2. Placing a passenger’s checked baggage on the same flight as the passenger
   • Some! If lose plane, then lose both. If passenger is “lost”, chances are the baggage is lost too. If baggage is lost/destroyed, then probably passenger is lost, too. (But, they don’t fate share when not in the air!)

3. Attaching a spare car key to the inside of the car’s bumper
   • Complete fate sharing. If car is lost, then key is also lost. And, key can’t really be lost without the car (it’s attached). i.e., key is lost, the car is lost, too.
PQ 5

• True/False: remote procedure call systems provide the same semantics as local procedure calls.
  
  • True
  
  • False
PQ 5

• True/False: remote procedure call systems provide the same semantics as local procedure calls.
  
  • True
  
  • False

   LPC provides exactly once semantics; RPC cannot provide this (in the presence of failures).
You decide to extend DFS in A2 with **DWRITE** mode (disconnected writes). *What are the DFS semantics you will have to revisit in your DWRITE design?* (List all that come to mind)
PQ 6.1

• You decide to extend DFS in A2 with DWRITE mode (disconnected writes). What are the DFS semantics you will have to revisit in your DWRITE design?

• Revisit READ mode semantics — what’s the “latest chunk version” if there are multiple identical versions available? Which one should win? How would you solve this?

• Revisit DREAD mode semantics — should reads in DREAD mode observe writes in DWRITE mode?

• Revisit Open file fetch semantics — why require a fetch for a file opened in DWRITE mode (for disconnected writing)?
PQ 6.2

• You decide to extend DFS in A2 with DWRITE mode (disconnected writes). What are the DFS semantics you will have to revisit in your DWRITE design?

• Revisit READ mode semantics — what’s the “latest chunk version” if there are multiple identical versions available? Which one should win? How would you solve this?

• Change READ mode to not be ‘latest’ but “most up to date, or local first if multiple identical versions”?

• Decide latest chunk based on clocks; use clock synchronization to decide versions offline.

• When offline assign non-deterministic versions to writes (random offline writer wins)

• Let server arbitrate multiple offline writes and pick winner

• Return all the writes and let the application figure it out (conflict resolution)
All problems in computer science can be solved by adding another level of indirection. But that will usually create another problem.” -- David Wheeler

• A2 design uses indirection. What does it use it for? (i.e., what advantages do you get from indirection in A2)?
PQ 7

All problems in computer science can be solved by adding another level of **indirection**. But that will usually create another problem.” -- David Wheeler

• A2 design uses indirection. What does it use it for? (i.e., what advantages do you get from indirection in A2)?

• Server interposes on client requests: clients don’t know each other identities, who has which chunks, who has which files open, do not observe client failures
PQ 8

• GlobalFileExists() in A2 is an idempotent operation
  • True
  • False
PQ 8

- GlobalFileExists() in A2 is an idempotent operation
  - True (with 1 client in the system)
  - False (with multiple clients in the system)

Clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request.
PQ 9

- A2 design assumes the server never fails. How would you extend the design to handle server failures?
A2 design assumes the server never fails. How would you extend the design to handle server failures?

- Treat server failures as disconnections at the client!
- All distributed state at the server must be stored durably on disk
- Introduce server restart/recovery procedure
- All operations that modify server state (e.g., write of a chunk generates a new chunk version) must use a logging protocol to ensure durability (same as the suggested write protocol, but server-side)
A2 design assumes the server never fails. How would you extend the design to handle server failures?

- Treat server failures as disconnections at the client!
- All distributed state at the server must be stored durably on disk
- Introduce server restart/recovery procedure
- All operations that modify server state (e.g., write of a chunk generates a new chunk version) must use a logging protocol to ensure durability (same as the suggested write protocol, but server-side).
PQ 10

• An API critically determines the design of the system. Imagine that the DFS API in A2 was changed such that there was no DREAD mode and no LocalFileExists call.

Assuming that file contents would still be stored at clients, how would your design change in response?
PQ 10

• An API critically determines the design of the system. Imagine that the DFS API in A2 was changed such that there was no DREAD mode and no LocalFileExists call.

  Assuming that file contents would still be stored at clients, how would your design change in response?

• Clients no longer fetch content on open. All read/write operations streamed to the remote client replica. No disconnected mode operations means MountDFS does not need to succeed in disconnected mode.
PQ 11

- A2 makes disconnections \textit{visible} to applications. Assume you changed A2 such that disconnections were \textit{invisible} to the application. \textbf{How would you change the DFS API and the DFS API semantics to accomplish this?}
A2 makes disconnections visible to applications. Assume you changed A2 such that disconnections were invisible to the application. How would you change the DFS API and the DFS API semantics to accomplish this?

- Remove DisconnectedError from all calls.
- Remove DREAD mode, LocalFileExists call.
- Either (1) make all calls best-effort — Read always returns local copy in disconnected mode, or (2) make operations block until server becomes available.