RPCs and Failure

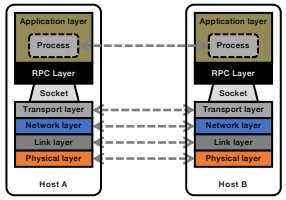
COS 418: Distributed Systems
Lecture 4

Mike Freedman

1

Last Time: RPCs and Network Comm.

- Layers are our friends!
- RPCs are everywhere
- Necessary issues surrounding machine heterogeneity



- Subtle issues around failures
  - … this time!!!

2

2

What could possibly go wrong?

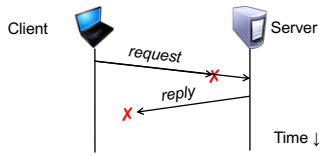
1. Client may crash and reboot
2. Packets may be dropped
   - Some individual packet loss in the Internet
   - Broken routing results in many lost packets
3. Server may crash and reboot
4. Network or server might just be very slow

All of these may look the same to the client…

3

3

Failures, from client's perspective



The cause of the failure is hidden from the client!

4

4

At-Least-Once scheme

- Simplest scheme for handling failures

1. Client stub waits for a response, for a while
   - Response is an acknowledgement message from the server stub
2. If no response arrives after a fixed timeout time period, then client stub re-sends the request

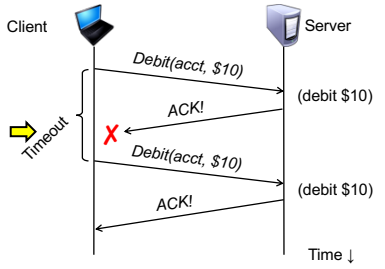
- Repeat the above a few times
  - Still no response? Return an error to the application

5

5

At-Least-Once and side effects

- Client sends a "debit $10 from bank account" RPC

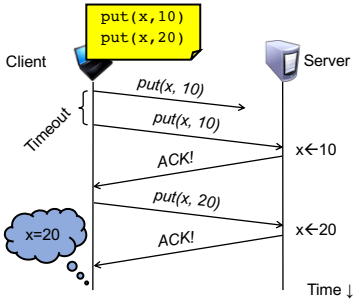


6

6

At-Least-Once and writes

- Consider a client storing key-value pairs in a database
  - put(x, value), then get(x): expect answer to be value

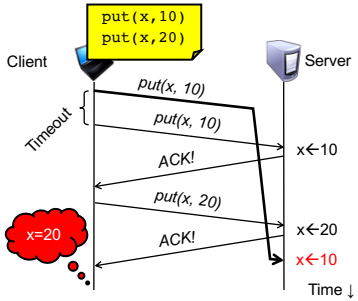


7

7

At-Least-Once and writes

- Consider a client storing key-value pairs in a database
  - put(x, value), then get(x): expect answer to be value



8

8

So is At-Least-Once ever okay?

- Yes: If they are read-only operations with no side effects
  - e.g., read a key's value in a database
- Yes: If the application has its own functionality to cope with duplication and reordering
  - You will need this in Assignments 3 onwards

9

9

At-Most-Once scheme

- Idea: server RPC stub detects duplicate requests
  - Returns previous reply instead of re-running handler
- How to detect a duplicate request?
  - Test: Server stub sees same function, same arguments twice
    - No! Sometimes applications legitimately submit the same function with same augments, twice in a row

10

10

At-Most-Once scheme

- How to detect a duplicate request?
  - Client stub includes unique transaction ID (xid) with each RPC request
  - Client stub uses same xid for retransmitted requests

```
At-Most-Once Server Stub
if seen[xid]:
       retval = old[xid]
else:
       retval = handler()
       old[xid] = retval
       seen[xid] = true
return retval
```

11

11

At-Most-Once: Providing unique XIDs

1. Combine a unique client ID (e.g., IP address) with the current time of day
2. Combine unique client ID with a sequence number
   - Suppose client crashes and restarts. Can it reuse the same client ID?
3. Big random number (probabilistic, not certain guarantee)

12

12

At-Most-Once: Discarding server state

- Problem: seen and old arrays will grow without bound
- Observation: By construction, when the client gets a response to a particular xid, it will never re-send it
- Client could tell server "I'm done with xid x – delete it"
  - Have to tell the server about each and every retired xid
    - Could piggyback on subsequent requests

Significant overhead if many RPCs are in flight, in parallel

13

13

At-Most-Once: Discarding server state

- Problem: seen and old arrays will grow without bound
- Suppose xid = ⟨unique client id, sequence no.⟩
  - e.g., ⟨42, 1000⟩, ⟨42, 1001⟩, ⟨42, 1002⟩
- Client includes "seen all replies ≤ X" with every RPC
  - Much like TCP sequence numbers, acks
- How does client know the server received the info about retired RPCs?
  - Each one of these is cumulative: later seen messages subsume earlier ones

14

14

At-Most-Once: Concurrent requests

- Problem: How to handle a duplicate request while the original is still executing?
  - Server doesn't know reply yet. And we don't want to run procedure twice
- Idea: Add a `pending` flag per executing RPC
  - Server waits for the procedure to finish, or ignores

15

15

At-Most-Once: Server crash and restart

- Problem: Server may crash and restart
- Does server need to write its tables to disk?
- Yes! On server crash and restart:
  - If `old[]`, `seen[]` tables are only in memory:
    - Server will forget, accept duplicate requests

16

16

Exactly-once?

- Need retransmission of at least once scheme
- Plus the duplicate filtering of at most once scheme
  - To survive client crashes, client needs to record pending RPCs on disk
    - So it can replay them with the same unique identifier
- Plus story for making server reliable
  - Even if server fails, it needs to continue with full state
  - To survive server crashes, server should log to disk results of completed RPCs (to suppress duplicates)

17

17

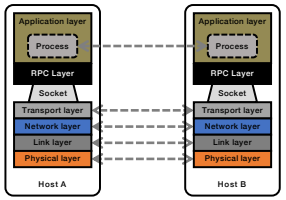
Exactly-once for external actions?

- Imagine that remote operation triggers an external physical thing
  - e.g., dispense $100 from an ATM
- ATM could crash immediately before or after dispensing
  - ATM would lose its state, and
  - Don't know which one happened (although can make window very small)
- Can't achieve exactly-once in general, in presence of external actions

18

Summary: RPCs and Network Comm.

- Layers are our friends!
- RPCs are everywhere
- Help support machine heterogeneity



- Subtle issues around failures
  - At-least-once w/ retransmission
  - At-most-once w/ duplicate filtering
    - Discard server state w/ cumulative acks
  - Exactly-once with:
    - at-least-once + at-most-once + fault tolerance + no external actions

19

19