14

Replication

* redundancy, replication

MTTF = mean time to fail
MTTR = mean time to repair
avail = MTTF/(MTTF+MTTR)

RAID 1: 2 mirrored disks — O(2N)
RAID 4: one parity disk (XOR'd), but all writes write to parity disk — O(N+1) — T.T
RAID 5: spread that out! — O(N+1)

GFS

Transactions

We can't replicate EVERYTHING...

- atomicity - actions that happen completely or not at all → single sector writes
  - shadow copies - make a copy and then edit the copy → bad performance
  - logs???
- Transaction - BEGIN+END. wb if they run concurrently
  - Isolation - locks??? really bad perf
    - "when multiple transactions run concurrently, but they seem to run serially

Logging

Better atomicity for txns

* log changes whether txn commit/abort - BEGIN, WRITES, READS, UPDATES, COMMITS, ABORTS

reads are hella slow

LOG + INSTALL, write ahead logging

recovery of storage is bad bc whole log. Writes are 2x - if storage is always fully updated, then you have to go back to everything to see whether txn was committed after updated (↑updates)

- Cache writes, then flush - writes aren't as bad bc cache
- log truncation [CHECKPOINT] in log
  - ↳ recovery is better

ISOLATION

We've enforced atomicity. Now how do we make sure concurrent transactions don't fuck up?

Answer: 2PL

* final state serializability - for some seq schedule, O.O leads to same final answer
* conflict serializability - order of all conflicts = order in some seq sched
  - conflict graph: CS iff ∃ cycle conflict graph

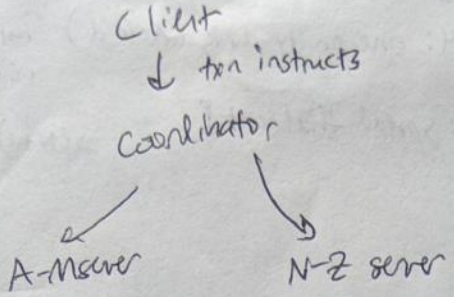
* 2PL: each shared var has a lock
  Before any operation, txn must acquire lock
  After releasing on lock, it can't acquire any new ones

Deadlock? Abort a txn

* reader/writer locks

Lec 18: Multi Site Atomicity

Problem: Maintaining atomicity on multiple machines. Consistency



Problem: What if A-M server commits, but N-Z does not?

*2PC – Two phase commit – PREPARE + COMMIT

Remaining Question – When some is down, some data is unavail? Handle via single copy consistency). Consistency Guarantees?

Lec 19:

Problem: We have multiple machines, but how do we maintain consistency across copies?

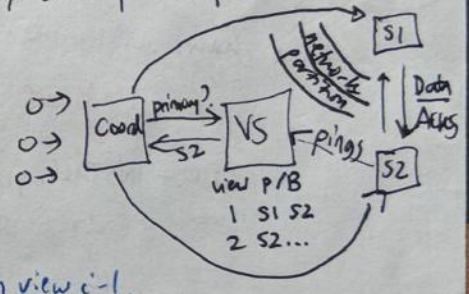
SOL: REPLICATED STATE MACHINES.

Players: clients, coordinators, primary/backup servers

VIEW SERVER – determines which replica server is primary, alerts servers about roles

COORDINATORS – ask view server who is primary/backup

servers – ping VS so VS can detect failures. primary-backup setup

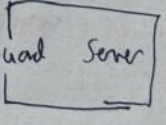


Network partitions create problems!

RULES: 1) Primary must wait for backup to accept each request
2) Backup must reject direct coordinator requests.
3) Primary must reject forwarded requests.
4) Primary in view i must have been primary or backup in view i-1.

Lec 20 Intro to Security

1) Policy – who can r/w data – Confidentiality, Integrity, Availability
2) Threat Model – who are we protecting against

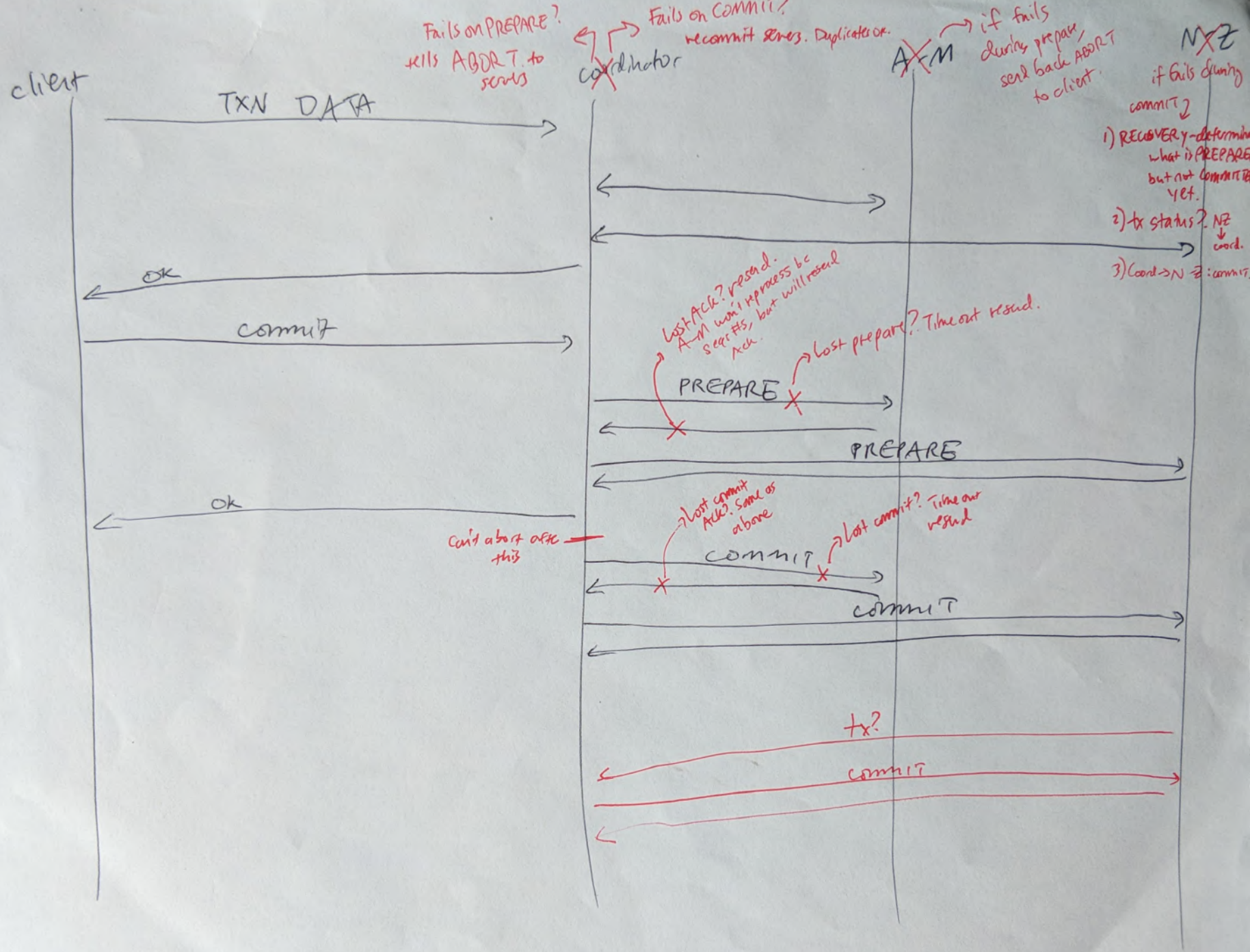


"Principle of least Privilege" – limit # of trusted components in system

Lec 21 Auth

How to verify someone is who they say they are?

1) Auth w/ pws → salt hashes
2) Session Cookies – includes server key ST ppl can't pretend to be someone else and use a forged cookie
3) Phishing – Challenge response


client
coordinator
A
M
N
Z
TXN DATA
Fails on PREPARE? tells ABORT to servers
Fails on COMMIT? recommit servers. Duplicates ok.
if fails during prepare, send back ABORT to client.
if fails during commit
1) RECOVERY - determine what is PREPARED but not committed yet.
2) tx status? NZ → coord.
3) Coord → NZ: commit
ok
commit
Lost ACK? resend. A→M won't reprocess bc seq #s, but will resend ack.
PREPARE
Lost prepare? Timeout resend.
PREPARE
ok
Can't abort after this
Lost commit Ack? Same as above
commit
Lost commit? Timeout resend
commit
tx?
commit

lec 18