Lecture notes on replication

replication is the process of maintaining multiple copies of the same data on different computers
(orthogonal to partitioning)

don't want to become a man w/ two docks

desirable properties
↳ transparency - client should be unaware of interactions w/ multiple copies
↳ consistency - copies may be temporarily inconsistent
↳ correctness depends on application semantics

why replicate data
↳ bring them closer to users - CDNs, game servers, Napster
↳ increase availability
- independent failures of servers located in different data centers

p - prob of server failure
p^N - prob that all N servers fail
$1 - p^N$ - prob of at least one server being up

$p = .05$ → 18 days of downtime in a year
$1 - p^2 = 91.75\%$ → 9 days
$1 - p^5$ → 10 seconds of downtime

(?) What's the difference between caching & replication?
↳ caching maintains previously requested copies of data
↳ cached data may be stale
↳ effectively, replication w/o a consistency protocol

(?) does caching increase availability?
↳ no guarantee on the # of global cache copies
↳ could have one half of data, but not the other

CAP Theorem - Eric Brewer '98

impossible for a distributed system (no clock synch, msg passing) to simultaneously provide:

Consistency - all nodes see the same data all the time
Availability - every request will eventually receive confirm/fail response
Partition Tolerance - system continues to operate in spite of net failures

? how does that relate to what we already know about DS

safety in unreliable systems → consistency
↳ Nothing bad happens if can't reach other processors
↳ Paxos, Raft, 2PC, 3PC

liveness in unreliable systems → availability
↳ eventually something good happens → response to req
↳ some tradeoffs - readers, but no writers
- Raft changes that can be rolled back

reliability - the continued operation under failures of nodes & links, crash or byzantine → partition tolerance

proof sketch:

$\{p_1\}$ $\{p_2, p_3\}$ partition → msgs between p_1 & p_2 are lost

Assume either $\exists\, c_1.\text{write}(x,1) \to p_1$ or $\exists\, c_1.\text{write}(x,2) \to p_1$
followed by $c_1.\text{read}(x) \to p_2$
eventually

p_2 cannot distinguish between the two cases

? what can p_2 do?
↳ delay reply → no availability
↳ reply w/ ? → no consistency if $x \neq 2$
↳ crash → no failure tolerance

? which one would you give up when building a system? Why?

2/3 ain't bad
↳ can build systems that provide 2/3
↳ can still be hard

? what tradeoff does an ATM make?

↳ can't forfeit P in large systems, so must strike a balance
↳ if C & A → best-effort availability in consistent systems
→ best-effort consistency in available systems
↳ but in practice P is rare, so don't need to give up anything

BASE - basically available, soft-state, eventually consistent
↳ as opposed to ACID (atomicity, consistency, isolation, durability)
↳ eventual consistency - eventually get the latest value
↳ liveness, but not safety → can return any set of values
↳ makes DS design more complex
↳ readers + writers $\neq N$
↳ ex. Facebook, nosql