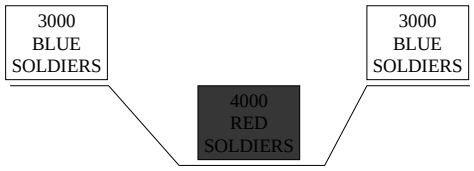
Replication

- previous lectures: replication for performance
- today: replication for availability and fault tolerance
  - availability: providing service despite temporary, short-term failures
  - fault tolerance: providing service despite permanent, catastrophic failures

Fault Models

- fail-stop: broken part doesn't do anything
  - better yet, it tells you it's broken
- Byzantine: broken part can do anything
  - adversary model
    - » playing a game against an evil opponent
    - » opponent knows what you're doing, tries to foil you
    - » opponent controls all broken parts
    - » usually some limit on opponents actions
      - example: at most K failures

Example: Two-Army Problem



Network Partitions

- can't tell the difference between a crashed process and a process that's inaccessible due to network failure
  - "crashed" process might still be running
- network partition: network failure that cuts processes into groups
  - full communication within each group
  - no communication between groups
  - danger: each group will think everybody else is dead

Fault-Tolerance and File Systems

- rest of lecture will focus on file systems
- why?
  - simple case
  - important case
    - » stands in for databases, etc.
  - illustrates important issues

Fault Tolerance and Disks

- disks have nice fault-tolerance behavior
- when you access a disk, either
  - the operation succeeds, or
  - you're notified of a failure
- model disk as fail-stop
  - simplifies fault-tolerance protocols

Mirroring

- goal: survive up to K failures
- approach: keep K+1 copies of everything
- client does operation on "primary" copy
- primary makes sure other copies do the operation too
- advantage: simple
- disadvantages:
  - do every operation K times
  - use K times more storage than necessary

Mirroring: Details

- optimization: contact one replica to read
- what if a replica fails?
  - get up-to-date data from primary after recovering from failure
    - » helpful if primary keeps track of what happened
- what if primary fails?
  - elect a new one
    - » this can be tricky!

Election Problem

- goals
  - when algorithm terminates, all non-failed processes agree on who is the leader
  - algorithm works despite arbitrary failures and recoveries during the election
  - if there are no more failures and recoveries, algorithm must eventually terminate

The Bully Algorithm

- use fixed "pecking order" among processes
  - e.g. use network address
- idea: choose the "biggest" non-failed machine as leader
- correctness proof is difficult

Bully Algorithm: Details

- process starts an election whenever it recovers, or whenver the primary has failed
- to start an election, send *election* messages to all machines bigger than yourself
  - if somebody responds with ACK, give up
  - if nobody ACKs, declare yourself leader
- on receiving election message, reply with an ACK, and start an election yourself (unless you have one going already)

Distributed Parity

- a trick that works for disks only
  - not for the general fault-tolerance case
- idea
  - store N blocks of data on N data servers
  - store parity (bitwise XOR) of the N blocks on an extra server
  - if server crashes, use the other N-1 data blocks, plus the parity, to reconstruct the lost block

Distributed Parity

- survives failures of one server
- after failure, reconstruct and replace the lost server
  - after this is complete, prepared for another failure
- can generalize to survive N failures, with N parity disks
  - fancy coding theory

Distributed Parity

Disk D0	Disk D1	Disk D2	Disk D3	Disk P
0	1	2	3	P(0-3)
4	5	6	7	P(4-7)
8	9	10	11	P(8-11)
12	13	14	15	P(12-15)

Distributed Parity

- to read, just read the appropriate block
- to write
  - read old data block
  - write new data block
  - read old parity block
  - compute new parity block
  - write new parity block
- heavy load on parity disk

Scattered Parity ("RAID 5")

Disk D0	Disk D1	Disk D2	Disk D3	Disk P
0	1	2	3	P(0-3)
4	5	6	P(4-7)	7
8	9	P(8-11)	10	11
12	P(12-15)	13	14	15

Distributed Parity vs. Mirroring

- read performance
  - both good
- write performance
  - mirroring: decent
  - parity: not so good
- space requirement
  - mirroring: use lots of extra space
  - parity: use a little extra space

Mirroring with Quorums

- with mirroring, writes are slowed down, since all replicas must be contacted
- improve this by introducing quorums
  - cost: reads get a little slower
- also helps fault-tolerance, availability

Quorums

- *quorum*: a set of server machines
- define what constitutes a "read quorum" and a "write quorum"
- to write
  - acquire locks on all members of some write quorum
  - do writes on all locked servers
  - release locks
- to read: similar, but use read quorum

Quorums

- correctness requirements
  - any two write quorums must share a member
  - any read quorum and any write quorum must share a member
  - (read quorums need not overlap)
- locking ensures that
  - at most one write happening at a time
  - never have a write and a read happening at the same time

Defining Quorums

- many alternatives
- example
  - write quorum must contain all replicas
  - read quorum may contain any one replica
- consequence
  - writes slow, reads fast
  - can write only if all replicas are available
  - can read if any one replica is available

Defining Quorums

- example: majority quorum
  - write quorum: any set with more than half of the replicas
  - read quorum: any set with more than half of the replicas
- consequence
  - modest performance for read and write
  - can proceed as long as more than half of replicas are available

Quorums and Version Numbers

- write operation writes only a subset of the servers, so some servers are out of date
- remedy
  - put version number stamp on each block in each replica
  - when acquiring locks, get current version number from each replica
  - quorum overlap rules ensure that one member of your quorum has the latest version

Quorums and Version Numbers

- when reading, get the data from the latest version in your quorum
- when writing, set version number of all replicas you wrote equal to 1+(max version number in your quorum beforehand)
- guarantees correctness even if no recovery action is taken when replica recovers from a crash

Fancy Quorum Rules

- example
  - divide replicas into K “colors”
  - write quorum: all replicas of some color, plus at least one of every other color
  - read quorum: one of each color
  - good choice: K colors, K of each color
- consequences
  - pretty good performance for reads and writes
  - very resilient against failures

Quorums and Network Partitions

- on network partition, three cases:
  - one group has a write quorum (and thus usually a read quorum): that group can do anything, other groups are frozen
  - no group has a write quorum, but some groups have read quorums: some groups can read but nobody can write
  - no group contains any quorum: everybody is frozen