Distributed Systems

28. Fault Tolerance

Paul Krzyzanowski

Rutgers University

Fall 2016

Faults

- Deviation from expected behavior
- Due to a variety of factors:
  - Hardware failure
  - Software bugs
  - Operator errors
  - Network errors/outages

Faults

- Three categories
  - transient faults
  - intermittent faults
  - permanent faults
- Processor / storage faults
  - Fail-silent (fail-stop): stops functioning
  - Fail-recover (fail-restart): stops functioning but then restarts (state lost)
  - Byzantine: runs but produces faulty results
- Network faults
  - Data corruption (Byzantine)
  - Link failure (fail-silent)
  - One-way link failure
  - Network partition
    - Connection between two parts of a network fails

Synchronous vs. Asynchronous systems

- Synchronous system vs. asynchronous system
  - E.g., IP packet versus serial port transmission

- Synchronous: known upper bound on time for data transmission
  - Why is this important?
  - Distinguish a slow network (or processor) from a stopped one

Fault Tolerance

- Fault Avoidance
  - Design a system with minimal faults
- Fault Removal
  - Validate/test a system to remove the presence of faults
- Fault Tolerance
  - Deal with faults!

Achieving fault tolerence

Redundancy

- Information redundancy
  - Hamming codes, parity memory ECC memory
- Time redundancy
  - Timeout & retransmit
- Physical redundancy/replication
  - Triple Modular Redundancy, RAID disks, backup servers

- Replication:
  - Copy information so it can be available on redundant resources
    → State machine replication
    → Consistency (or eventual consistency), message ordering

- Failover: Switch operation from a failed system to a redundant working one

Availability: how much fault tolerance?

100% fault-tolerance cannot be achieved

– The closer we wish to get to 100%,
  the more expensive the system will be

– Availability: % of time that the system is functioning
  - Typically expressed as # of 9's
  - Downtime includes all time when the system is unavailable.

Availability

Class	Level	Annual Downtime
Continuous	100%	0
Six nines (carrier class switches)	99.9999%	30 seconds
Fault Tolerant (carrier-class servers)	99.999%	5 minutes
Fault Resilient	99.99%	53 minutes
High Availability	99.9%	8.3 hours
Normal availability	99-99.5%	44-87 hours

Availability

- At home, component failure is a disruptive event
- In a network of 100,000+ machines, it is a daily issue

Points of failure

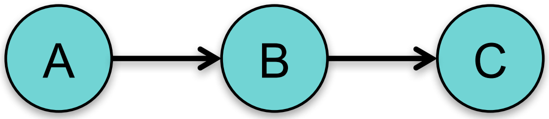
- Goal: avoid single points of failure

- Points of failure: A system is *k-fault tolerant* if it can withstand *k* faults.
  - Need *k+1* components with silent faults
    k can fail and one will still be working
  - Need *2k+1* components with Byzantine faults
    k can generate false replies: *k+1* will provide a majority vote

Active replication

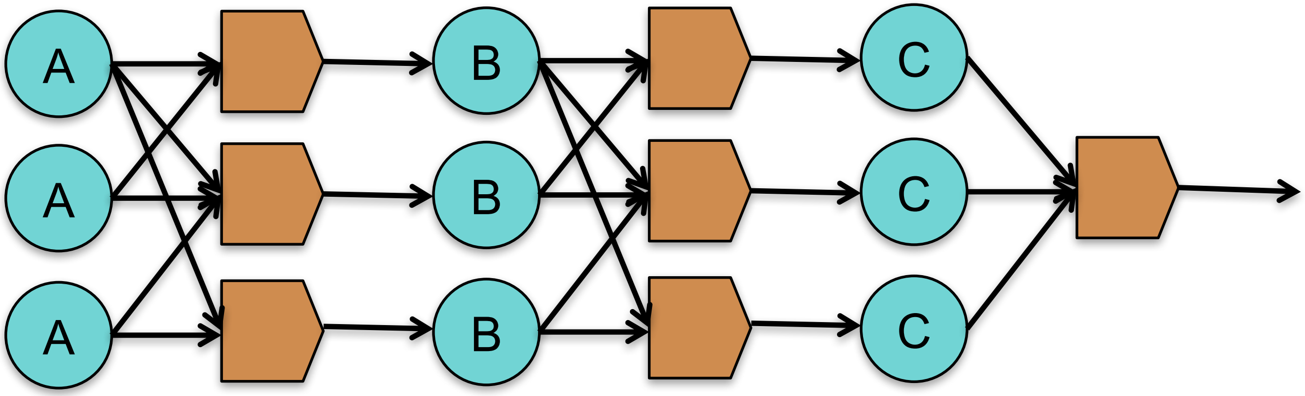
Technique for fault tolerance through physical redundancy

No redundancy:



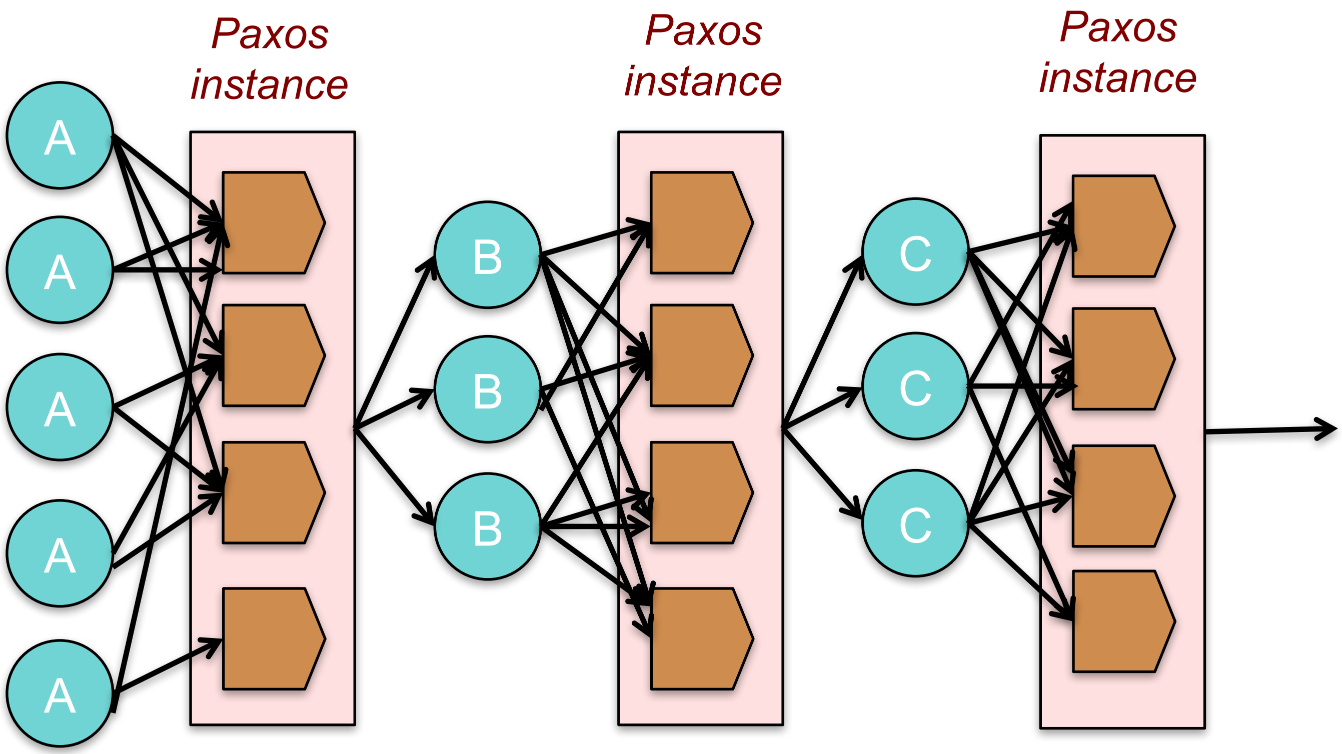
Triple Modular Redundancy (TMR):

Threefold component replication to detect and correct a single component failure – *voting to detect Byzantine failures*



Active replication: Replicated State Machines

Use a distributed consensus algorithm to agree on the order of updates across all replicas.



Active-Active vs. Active-Passive

- Active-Active
  - Any server can handle requests – global state update
  - Usually requires total ordering for updates:
    - Paxos, distributed lock manager, eventual or immediate consistency (Brewer's CAP theorem impacts us)
- Active-Passive = Primary Backup(s)
  - One server does all the work
  - When it fails, backup takes over
    - Backup may ping primary with *are you alive* messages
  - Simpler design
  - Example: Chubby, GFS master, Bigtable master
- Issues
  - Watch out for Byzantine faults
  - Recovery may be time-consuming and/or complex

Agreement in faulty systems

Two army problem

- good processors - faulty communication lines
- coordinated attack
- multiple acknowledgement problem

Agreement in faulty systems

Byzantine Generals problem

- reliable communication lines - faulty processors
- *n* generals head different divisions
- *m* generals are traitors and are trying to prevent others from reaching agreement
  - 4 generals agree to attack
  - 4 generals agree to retreat
  - 1 traitor tells the 1st group that he'll attack and tells the 2nd group that he'll retreat
- can the loyal generals reach agreement?

Agreement in faulty systems

Byzantine Generals problem

- Solutions require:
  - *3m+1* participants for *m* traitors (*2m+1* loyal generals)
  - *m+1* rounds of message exchanges
  - $O(m^2)$ messages
- Costly solution!

Examples of Fault Tolerance

Example: ECC memory

- Memory chips designed with Hamming code logic
- Most implementations *correct* single bit errors in a memory location and *detect* multiple bit errors.
- Example of information redundancy
  - *Why is this not physical redundancy?*

    The extra circuitry is not n-way replication of existing components

Example: Failover via DNS SRV

- Goal: allow multiple machines (with unique IP addresses in possibly different locations) to be represented by one hostname
  - Instead of using DNS to resolve a hostname to one IP address, use DNS to look up SRV records for that name.
    - Each record will have a priority, weight, and server name
    - Use the priority to pick one of several servers
    - Use the weight to pick servers of the same priority (for load balancing)
    - Then, once you picked a server, use DNS to look up its address
  - Commonly used in voice-over-IP systems to pick a SIP server/proxy
  - MX records (mail servers) take the same approach: use DNS to find several mail servers and pick one that works
- Example of physical redundancy

Example: DNS with device monitoring

- Custom DNS server that returns an IP address of an available machine by monitoring the liveness of a set of equivalent machines
  – Akamai approach (Akamai has more criteria than this)

Example: TCP retransmission

- Sender requires *ack* from a receiver
  - Acknowledgement contains next expected byte #
- If the *ack* is not received in a certain amount of time, the sender retransmits the packet
  - If a packet is received but the next expected byte # is unchanged, the sender assumes that the previous packet has not been received
- Example of time redundancy

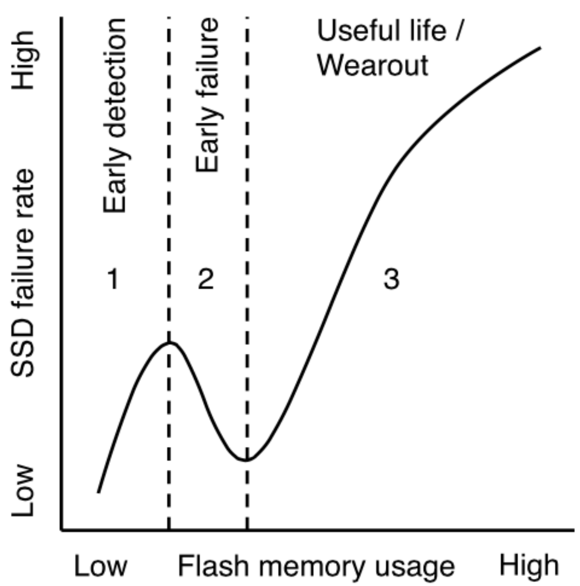
On Windows:

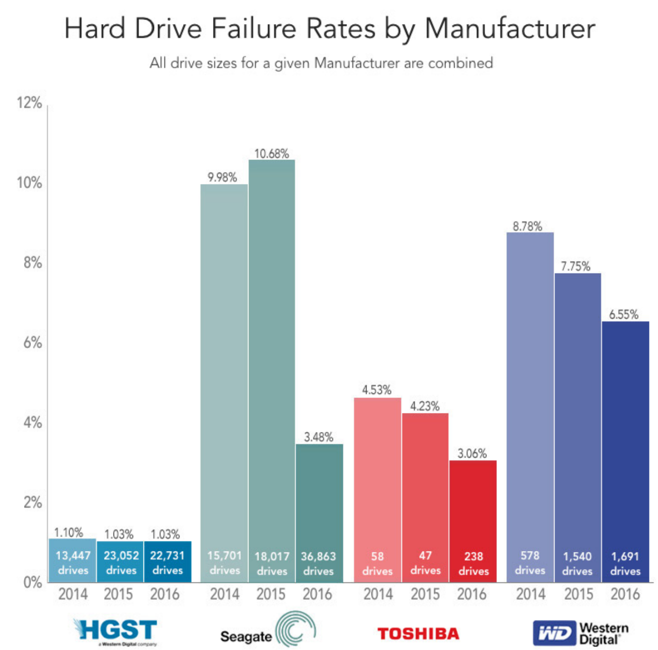
- 3 second timeout for new connections
- Adjusted based on performance for existing connections

See RFC 6298, *Computing TCP's Retransmission Timer*

Disk failure

- Hard disk annual failure rates ~ 5% (1% … 10%+)
  – 80 disks per rack × 100 racks ⇒ >1 failure per day on average
- SSD annual failure rates ~ 1.5%
  – 2–7% develop at least one bad chip in the first four years
  – 30–80% develop at least one bad block





http://www.storagereview.com/node/4966

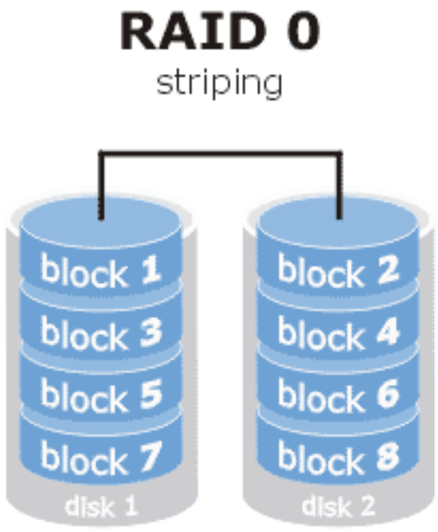
https://www.backblaze.com/blog/hard-drive-reliability-stats-q1-2016/

Example: RAID 1 (disk mirroring)

- RAID = redundant array of independent disks
- RAID 1: disk mirroring
  - All data that is written to one disk is also written to a second disk
  - A block of data can be read from either disk
  - If one disk goes out of service, the remaining disk will still have the data
- Example of physical redundancy

RAID 0: Performance

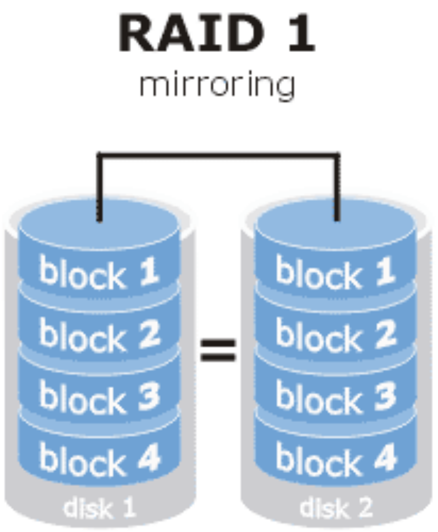
- Striping
- Advantages:
  - Performance
  - All storage capacity can be used
- Disadvantage:
  - Not fault tolerant



RAID 1: HA

- Mirroring
- Advantages:
  - Double read speed
  - No rebuild necessary if a disk fails: just copy
- Disadvantage:
  - Only half the space

Physical Redundancy

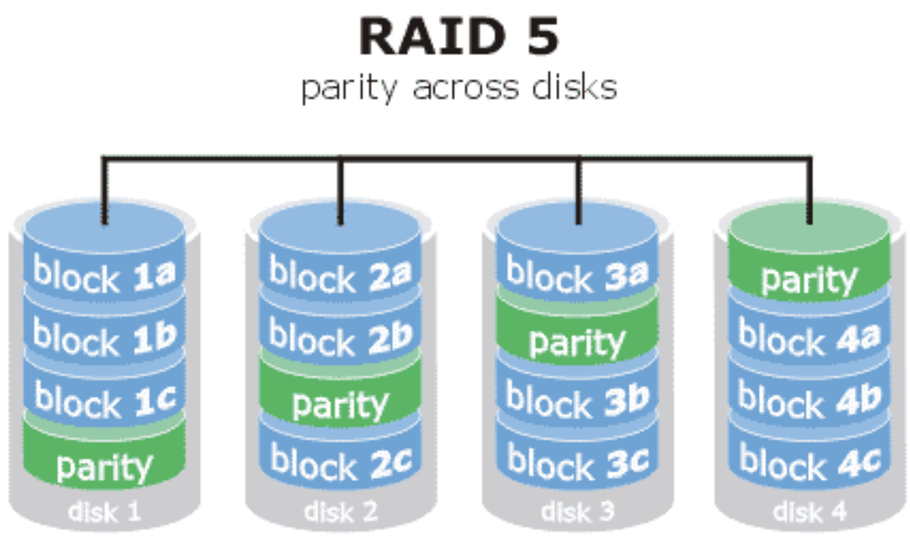


Example: RAID-4/RAID-5

- Block-level striping + parity
- Blocks are spread out across N disks and a parity block is written to disk N+1. The parity is the exclusive-or of the set of blocks in each stripe.
- If one disk fails, its contents are recovered by computing an exclusive-or of all the blocks in that stripe set together with the parity block
- RAID-5: same thing but the parity blocks are distributed among all the disks so that writing parity doesn't become a bottleneck.
- Example of information redundancy

RAID 5

- Interleaved parity
- Advantages:
  – Very fast reads
  – High efficiency: low ratio of parity/data
- Disadvantage:
  – Slower writes
  – Complex controller



Information redundancy

(extra physical components but no data redundancy)

RAID 1+0

- Combine mirroring and striping
  - Striping across a set of disks
  - Mirroring of the entire set onto another set

Fault tolerant techniques we encountered

- **Networking**
  – Ethernet checksums, IP header checksums, TCP & UDP data checksums
  – TCP retransmission, resequencing, congestion control, IP routing
- **Remote procedure calls**
  – Retransmission of requests with time-outs
- **Group communication & virtual synchrony**
  – Retransmission of data
  – Partial and total ordering to ensure replicas are consistent
    - Replicated inputs (replicated state machines)
  – Group management and view changes in virtual synchrony
- **File systems**
  – Replicated servers (Coda, AFS, GFS, Dropbox)
  – Disconnection: Queued changes if a server is not available (Coda)

Fault tolerant techniques we encountered

- **Mutex, Election, Consensus, and Commit algorithms**
  - Leases vs. locks to clean up state after a timeout
  - Leader election (e.g., using Paxos or election algorithms)
  - Mechanisms to agree on data & state of protocol even if processes die
    - Concept of a *quorum* of >50% live processes
    - Writeahead logs
  - Undoing or redoing changes after a failure
    - Writeahead log in commit protocols
    - GFS operation log (file journal)
- **Checkpointing**
  - Pregel's periodic checkpoints to save the state of the computation

The end