

Design and Engineering of Computer Systems

Lecture 36: Fault tolerance and reliability

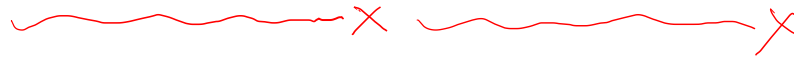
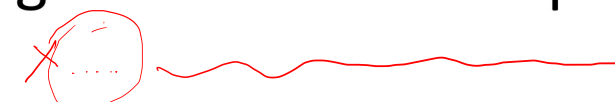
Mythili Vutukuru

IIT Bombay

Fault tolerance and reliability



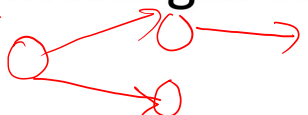
- Computer systems face many kinds of faults
 - Hardware faults due to hardware malfunction, disk failures, ..
 - Software faults due to design flaws, implementation bugs, configuration errors, ..
 - Operational faults due to power failures, network failures, security attacks, ...
 - Faults can be transient, intermittent, permanent
- Most faults go unnoticed and cause no harm, but some faults lead to failures or errors perceived by users of the system
 - Types of failures: fail stop (system stops on failure), fail safe (system fails and falls back to a safe state), Byzantine (system acts maliciously after failure)
- Fault tolerance: ability to mask faults before it becomes a failure or error that the user notices, leading to reliable systems
- This week: how to design systems that are robust to faults

Quantifying reliability


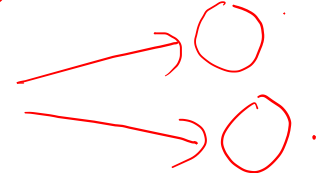
- Mean time to failure (MTTF): average time between failures of a system / component

- Mean time to recovery (MTTR): average time taken to repair the system / component after a failure

- Ideally, we want high MTTF and low MTTR
- Availability (uptime): fraction of time the component is running
 - Approx. $\text{MTTF} / (\text{MTTF} + \text{MTTR})$
 - Downtime = $1 - \text{uptime}$
- Availability also measured in terms of “number of nines”, e.g., a system with 99.99% availability is said to have “four 9s of availability”
 - 86,400 seconds in a day, 99.99% available \rightarrow max 8.64 seconds of downtime per day
0.01% down

Techniques for fault tolerance

- Error detection: how do we detect when a failure has occurred?

- Use sequence numbers and ACKs to detect packet losses (e.g., TCP) 
- Add additional bits (CRC, checksum, ..) to a network packet or data on storage media (e.g., hard disk, DVDs), in order to detect any errors or data corruption 
- Components send heart beat messages to each other periodically, missed heart beat implies failure 

- Error recovery: add redundancy into the system so that fault can be masked and failure avoided

- Retransmission of lost network data (e.g., TCP), application sees no error
- Add additional redundant bits (error correcting codes) to network packets and data on storage media to recover from bit errors 
- Replication of system components and data for redundancy 

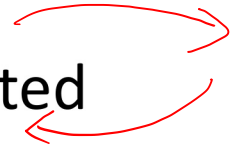
Error detection and correction codes

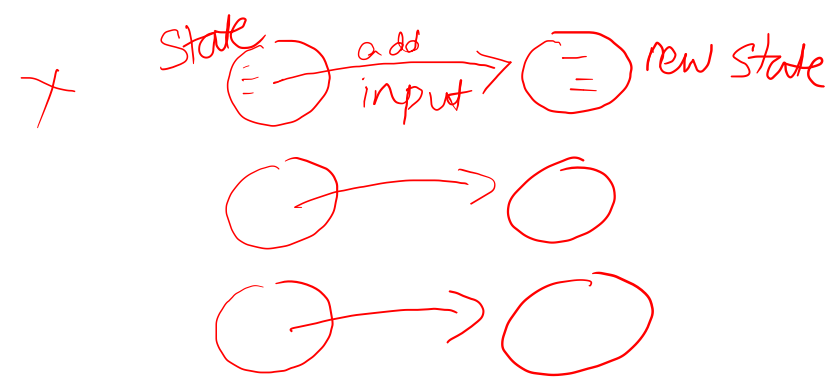
- Error detection / correction codes: additional bits added to messages transmitted over network, or to data stored on persistent storage in a system
 - Detect bit-level corruption of message/data, and even recover original data
- Example: parity bit (add 0 or 1 to make number of 1s odd/even)
 - Even parity: 100 → 1001, 101 → 1010
 - Error detected if parity does not match (e.g., number of 1s is odd, not even) 1011 ✗
- Example: repetition code (repeat a bit multiple times)
 - Example: 0 → 000, 1 → 111, repetition coded data is transmitted/stored
 - One bit errors can be corrected (000 corrupted to 001, can be corrected to 000)
 - Two bit errors can be detected (000 corrupted to 101, can only be detected, not corrected)
- More efficient error correction codes (e.g., Hamming codes) available
 - Add fewer extra bits, but detect/correct a wide range of bit errors

Replication for fault tolerance



- Example: app server maintains shopping cart of all users in e-commerce system
 - Processes user requests to add/delete/view items in shopping cart
 - If server crashes and restarts, shopping cart data lost, user shown incorrect cart
- Solution: replicate shopping cart functionality at multiple server replicas
 - Active-active: User requests to add/delete items replicated and processed (redundantly) by all replicas. If some replicas fail, others can handle the requests correctly.
 - Active-passive: User requests to add/delete items are only handled by one active server replica, which periodically checkpoints shopping cart state in a persistent storage. When active replica fails, one of the passive replicas becomes active, reads state from persistent storage, and resumes.
- Design principle: always send response back to clients (i.e., release any network output) only after replication / check-pointing has been completed
 - Otherwise, client may assume request succeeded (e.g., item added to cart) but replication fails at server (e.g., replicas could not be contacted) and data can be lost

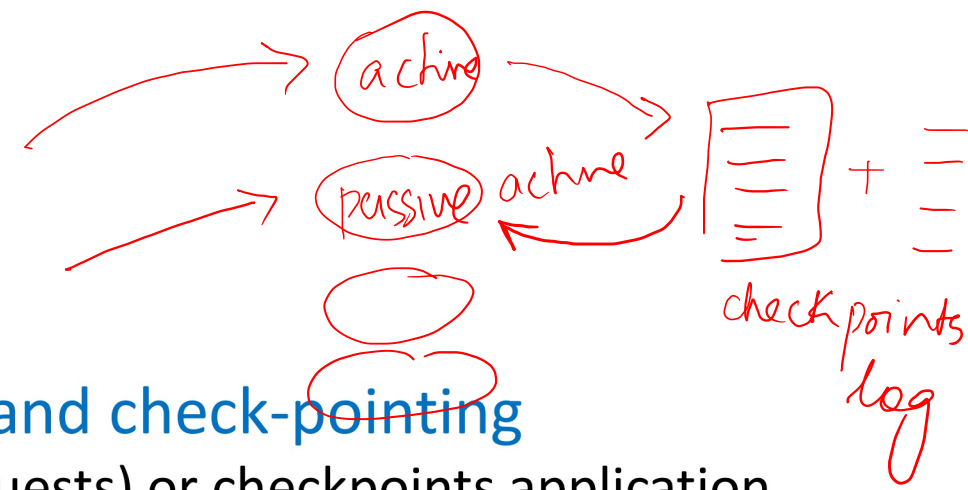




Active-active replication

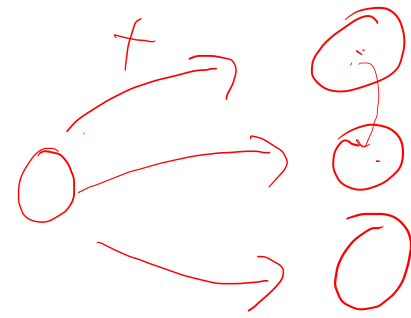
- Active-active replication also known as replicated state machines
 - All replicas maintain the same application state (e.g., shopping cart)
 - All replicas receive same inputs to update state in the same order (e.g., requests to add/delete items)
 - Must ensure no non-determinism due to randomness, so that all replicas processing same inputs have same application state
- **Failover**: if one of the active replicas fails, another can easily take over the requests of the failed replica
 - Quick failover, but higher overhead to keep all replicas in sync all the time
- One of active replicas may be designated as primary/leader and have different responsibilities, e.g., coordinate the replication operation, perform operations that should be done only once
 - E.g., only one of the replicas of a checkout server will bill the credit card

Active-passive replication



- Active-passive replication is also called **logging and check-pointing**
 - Active replica logs all inputs (e.g., all add/delete requests) or checkpoints application state (e.g., contents of shopping cart) in persistent storage
 - Logging inputs is more work than check-pointing final app state
 - Combination may be used: checkpoint periodically, log inputs between checkpoints
 - Full system (not app-specific) replication involves check-pointing all memory of a VM
- **Failover**: if active replica fails, passive replica takes over as active
 - Read latest checkpoint and/or replay inputs from log, start processing requests
 - Longer time for recovery than active-active replication
- Logs and checkpoints stored in reliable persistent storage accessible to both active and passive replicas, e.g., fault-tolerant remote data store
 - Remote data store may also use replication for storing checkpoints reliably

Recovery from failures



- **Failure detection:** replicas use mechanisms like heart beats (periodic messages) to keep track of who is alive and who is not
 - Missed heartbeat indicates failure, triggers recovery/failover procedure
- **Failure recovery / failover:** switch to a backup correctly
 - Active-active: new leader/primary is elected, traffic is no longer sent to failed replica, remaining replicas continue to handle user requests
 - Active-passive: one of the passive replicas becomes active, restores state from checkpoint and/or replays log, starts handling user requests
- Failure recovery does not violate correctness for users/clients of a system
 - Requests for which user received a positive response would have been replicated successfully (response sent only after replication), preserved after failover
 - Any user requests that were in the middle of processing and not replicated correctly will not have received response, must be retried by the user

Challenges with replication

- Replication should be transparent to end user, should provide a consistent view of system as if there is only one replica. How?
- Must ensure consistency / consensus across replicas, in spite of failures
 - Inputs should be replayed at all active replicas in the same order, even if some of them are unreachable sometimes, so that all of them have the same app state
 - All replicas need to agree on certain decisions, e.g., which replica will take over next
- Must ensure atomicity of transactions in case of failures
 - A set of actions should appear like one unit or transaction to end user, irrespective of failures, i.e., all actions performed together or all fail together
 - E.g., checkout server replica accepts user payment, but fails before shipping product. The backup replica should either complete shipping product, or cancel payment

Summary

- In this lecture:
 - Fault tolerance and availability
 - Replication for fault tolerance
 - Active-active and active-passive models of replication
- We will see solutions to the problems of consistency and atomicity in the next lectures, but before that, try to see if you can work out some preliminary solutions yourself. For example, how do you ensure consistency across replicas, when replicas can fail sometimes?