Fault tolerance in Distributed Systems

Class 10

Distributed Fault-tolerance: How to get it

1. Failure Detection
2. Membership
3. Communication
4. Replication management
5. Resilience
6. Recovery

Membership

- A Process Group: a set of participants cooperating towards some common goal
  - Membership of the group changes over time as participants fail and recover
    - membership service keeps track of current membership, and informs members of the current
    - group view: the subset of the members that is available.
  - Membership can also change deliberately
    - response to environmental or service requirements

What is the “correct” Group View?

- Members’ views must necessarily lag reality
  - What happens if a participant repeatedly leaves and rejoins the group?
- Working definition of correctness:
  - if membership doesn’t change, and links don’t fail, then all members eventually see the same view
- Membership service should be
  - consistent
  - accurate
Membership Service

- What happens if failure detection is:
  - inaccurate?
  - incomplete?
- Notification of changes in membership
  - should arrive everywhere in the same order
  - should be synchronized with respect to the other traffic seen by the group.

Linear Membership Service

- Views are totally ordered
  - system moves from one view to another with every participant in agreement as to the order

Membership Service

- What happens when a partition occurs?
  1. allow participants in the primary partition to proceed, while others are blocked. They can proceed only when the partition is healed.
  2. Force the non-primary participants to crash. They can be recovered and join the system later
- In both cases, the service is degraded.

Partial Membership Service

- Keep delivering (inconsistent) views in both partitions.
  - When partition is healed, state is reconciled.
- No total order on views.
  - Strong partial order: concurrent views don’t intersect
Communication

Reliable delivery in the presence of faults in the channel:

- Omission, timing and value faults

Reliable Delivery

- Mask the fault, by using multiple networks (spatial redundancy)
- Mask the fault, by send multiple copies of a message (temporal redundancy)
  - duplicates discarded at recipient
- Detect and recover (ack and retransmit)
  - acks may be +ve or –ve
- *When* should one mask rather than detect & recover?

Sender Failures in Multicast

- Software multicast: sender might send to some recipients, and then fail.
- Hardware multicast:?

Levels of reliability:

- (a) Unreliable
- (b) Best Effort
- (c) Reliable

Implementing Reliable Multicast

Error Masking and Error Recover

- Masking: all participants re-multicast every message they receive
- Recovery: save messages, and retransmit if the sender is seen to have failed
  - a *stable* message is one that has been received by all recipient
  - stability tracking protocol: when a msg is stable everywhere, it can be deleted from the stash
- All dependent on failure detection
What about Assertion Faults?
1. Convert assertion faults into omission faults by using CRCs, signatures, etc.
   - deals with faults in the channel but not in the sender.
2. Achieve consensus amongst the multiple recipients of a multicast message.

Byzantine Agreement
(Why is this in the section on communication?)
- In the Byzantine Generals problem, some of the participants may be traitors (fail)

- Agreement requires $3f + 1$ participants to tolerate $f$ Byzantine faults
  - even if the channel is perfect (no messenger is captured)
  - tolerating $f$ faults requires $f+1$ rounds of messages

Causal Order despite Communication Failure
- $m_3$ can never be delivered at $q$
- $m_2$ should never become deliverable
  - not enough copies of $m_1$ in the system
**Totally-Ordered Multicast**

- Securing total order is equivalent to securing consensus
  - participants have to agree on the delivery order!

**Replication Management**

Replication is spatial redundancy

- Assume:
  - network does not partition
  - fail-stop: process failures are all crashes
  - all processes are deterministic state machines

**Active replication**

- use atomic multicast to distribute system events (atomic = reliable + totally-ordered)
- run the same state machine in \( n \) places

**Semi-Active Replication**

- What if the programs are non-deterministic?
- Use leader-follower architecture:
  - leader makes all non-deterministic choices, and disseminated the results to the followers.
  - not necessary to use atomic multicast, since execution order can be disseminated too; reliable multicast will do
Other Options

Passive Replication
- replicas log commands, but don't execute them
  - what if processes are non-deterministic …

Lazy Replication
- Ladin’s gossip algorithm
- Causal order

What about Partitions?

Weighted Voting
- Any set of participants with a majority of the votes can proceed
  - \( w = \) write quorum, \( r = \) read quorum, \( n = \) nr of votes
  - require \( 2w > n \) and \( r + w > n \)
- Did you spot the deliberate error?
  - \( n = 7, r = w = 4 \)
  - 4 nodes …

Coteries
- A set \( Q \) of sets, such that each quorum in \( Q \) overlaps with every other quorum
  - \( Q = \{\{a, b\}, \{b, c\}, \{a, c\}\} \) is a coterie of \{a, b, c, d\}
  - Weighted voting majorities are a special case

Resilience

So: we have value redundancy
- What do we do with the multiple (possibly conflicting) values?
- Consumers should reach agreement!
- Sometimes, the inputs are not exactly the same:
  - clock synchronization
  - readings from replicated thermometers
Recovery

After and un-masked, detected failure!

- Recover state from **stable storage**
  - not necessarily disks
- Checkpointing
  - Coordinated at all participants (like consistent cut protocol)
  - Uncoordinated (may cause multiple rollbacks: the *domino effect*).

Logging

- Conceptually similar to checkpointing
  - replaying the log requires that processes are deterministic
  - logging may be pessimistic or optimistic
    - optimistic logging might require roll-back
  - If system is non-deterministic, all non-deterministic choices must be logged too.

Atomic Commitment

- 2PC is the most common protocol

  - **2PC can block**
    - coordinator can fail between *prepare* and *commit/abort*
    - other participants are blocked waiting for decision.
  - 3PC is non-blocking so long as a majority of the processes are correct.
State Transfers

A failed replica must be recovered and re-integrated into the system

- Normally application dependent, since we wish to minimize the network traffic
- The state to be transferred is a moving target!
  - We must ensure that state is transferred faster than it is changed

Totally ordered broadcast can be used to mark the instant at which a replica rejoins