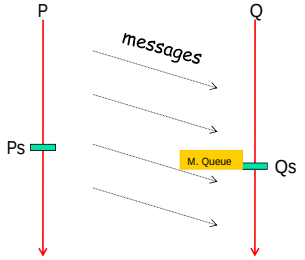
Distributed Systems Fault Tolerance

Kai Shen

Distributed Snapshots

- Snapshot of a distributed system for checkpointing/restart



- Can there be a message sent after Ps and received before Qs?
- Message queue for Qs

Chandy-Lamport Distributed Snapshots

- Any process can initiate snapshot-taking
  - Record own state and broadcast marker
  - Start recording all messages received on all incoming channels
- Marker receiving rule (followed also by the initiator)
  - If I have not yet recorded my own state (first marker being received)
    - Record own state and broadcast marker
    - Start recording all messages received on all incoming channels
  - If I have already recorded my own state (not the first marker)
    - Record state of channel on which maker was received
    - Stop recording that channel
- When does it terminate?

Consensus Problem

- Many distributed system problems are about reaching decisions consistently
  - Whether to commit a transaction in a distributed database?
  - How to order a series of updates in a replicated database?
  - Who gets the lock first in a distributed lock management?
  - Who should be the leader to perform a task on behalf of all of us?
  - … …

Fault Tolerance

- Fault tolerance in a distributed system
  - Nodes may fail, messages may disappear
  - Non-faulty nodes still want to get work done
- Fault-tolerant consensus:
  - Reach agreement on something, e.g., determine whether a bit should be 1 or 0
  - Consistency: all must agree on one value
  - Non-triviality: both 1 and 0 may appear as the agreed result, depending on the system semantics

Is it Difficult?

- Two-generals' problem
  - Two nodes with a faulty communication line
  - Try to reach agreement by proposing an time of coordinated attack and wait for acknowledgement
- Impossibility result for any deterministic protocol
  - Assume a minimal set of successful messages that convince both to attack
  - If the last message was lost, then the receiver would have doubt while the sender would attack
- Non-deterministic protocol
  - Message may be lost but delivery time is bounded; resend if lost
  - Protocol completes when message delivery eventually succeeds

Paxos Algorithm (Lamport)

- Failure modes
  - Nodes fail-stop
  - Messages can be lost, but do not linger forever
- Basic idea
  - Leader gathers majority opinion, makes proposal, waits for majority to accept

Paxos Algorithm

1. Initiate a round, the leader sends "Collect" to everyone.
2. A node, receiving the message, responds with "Last" message of any previously accepted value (if any).
3. When the leader collects >n/2 "Last" messages (info-quorum), it proposes a value through a "Begin" message to everyone.
4. A node, receiving the message, accepts the proposed value and responds with "Accept".
5. The leader (or anyone who wants to know the consensus result) waits for >n/2 "Accept" messages (accept-quorum) to successfully conclude the round.

Paxos Algorithm (Lamport)

- A round may not succeed
  - Failure of nodes (or leader), loss of messages
  - If a round fails, another can be started by the leader or a new leader
- Can two rounds both succeed?
- Can they accept different values?
  - A successful round lead to the acceptance of a value by a majority; all nodes must ever only accept one value eventually
- Non-deterministic protocol; tolerate failures of fewer than half of the nodes

Byzantine Failures

- Node failures:
  - Crash, or fail-stop
  - Byzantine: do arbitrary (maybe malicious) things
- Consensus with fail-stop failures:
  - Non-faulty nodes try to reach a decision
  - Then impose upon the whole system as a majority
  - For k failures, whole system size is at least 2k+1
- Consensus with Byzantine failures:
  - How to guarantee the decision is the majority of non-faulty nodes?
  - For k failures, we need at least 2k+1 good nodes
  - n-node Byzantine system cannot tolerate k failures if n<=3K

Consensus in Asynchronous Systems

- Synchronous systems
  - Messages take bounded delay (operate in steps)
- Asynchronous systems
  - Messages can take arbitrarily long
  - Impossible to distinguish message losses from slow messages
- Impossibility result
  - Not even a single machine failure can be tolerated