Viewstamped Replication to Practical Byzantine Fault Tolerance
ViewStamped Replication: Basics

• What does VR solve?
  – VR supports replicated service
  – Abstraction is replicated state machine (RSM)
    • As opposed to client app performing read/writes and managing concurrency

• Assumptions:
  – Fail-stop model: can support $f$ simultaneous faults with $2f+1$ replicas
  – Asynchronous network model
  – Messages can be corrupt in transit;
    • A replica can determine if a msg is correct or incorrect
Key Ideas

• Requirement
  – Single total ordering of requests
  – Failure tolerance with \( f \) replicas failing simultaneously

• Use a replica group of \( 2f+1 \) replicas for failure tolerance
  – \( f \) replicas may not participate \( \Rightarrow \) slow or crashed
  – Out of remaining \( f+1 \) ones, \( f \) may crash, since other \( f \) are just slow and not crashed (not part of faulty \( f \) set)
  – At least 1 replica will record the correct operation
VR Protocol

- Has 3 operation modes (or replica status)
  - Normal: replica is handling client requests
  - View-change: new replica is being selected → a replica is engaged in the view change protocol
  - Recovery: a crashed node has recovered and running protocol to join the replica group
VR Protocol: Normal operation

VR with configuration f=1
VR Protocol: Normal operation

- Client sends REQ to primary replica
  - REQ, op, c, s, v → to primary (operation with id is op)
- Primary advances op-number; adds request to log; sends PREPARE to all other replicas
  - PREPARE, m, v, n (op-number n, view-number v)
- Non-primary replicas processes PREPARE msg in order
  - Waits for all PREPARE msgs less than n to show up and add to log
    - May need to do state transfer with other replicas
  - Replies with <PREPAREOK v,n,i>
- Primary waits for f PREPAREOK msgs
  - Primary commits the operation; sends <REPLY v,s,x> to others
- Replicas commit operation after receiving REPLY
VR Protocol: View Change

- Goal is to mask failure of primary replica
  - May have crashed; can be slow in response
  - Any replica initiates view change when it has not heard from primary regularly

- Observations:
  - When client does not receive reply from primary, sends request to all replicas
  - f+1 replicas has the record of previous operations
    - Get information from at least f+1 logs for view change
  - One problem:
    - A client request which a primary is processing may get lost
ViewStamps and Protocol

• Viewstamp is a pair <view-number, op-number>
  – Always unique for each operation; different client requests get different viewstamps
  – If a replica gets info about two different operations, with same op-number, pick the larger view-number (i.e. higher viewstamp)

• Steps:
  – DoViewChange: send msg to replica which will be new primary
  – New primary receives f+1 msgs, it updates logs to the most recent values received, updates view-number, and sends StartView msg to other replicas
  – Replicas replace their log with latest log from new primary, and update view-number and status
Recovery Step

• Important step otherwise number of crashed replicas will be greater than f in long running application

• Steps:
  – Send RECOVERY msg to other replicas
  – A replica i responds with RECEOVERY_RESPONSE msg
  – Recovering replica waits for RECEOVERY_RESPONSE msg to build state; then sends PrepareOK msg
Byzantine Fault Tolerance

• Extend VR for preventing malicious attacks and byzantine behavior
  – Service may fail when attacker breaks an assumption

• Practical assumptions
  – Replicas may be compromised ➔ may silently discard an operation
  – Primary can send incorrect messages
  – Clients can be compromised and send incorrect requests
PBFT Protocol

• Mask the possibility that primary can be malicious
  – Introduce additional step to check validity of messages from the primary

• Needs 3f+1 replicas to tolerate up to f faulty nodes (byzantine)
  – Can get replies from 2f+1 replicas
PBFT Protocol : Normal case