Mar 5
Viewstamped Replication

Vijay Chidambaram

Viewstamped Replication

- Replication technique that handles node failures
- Provides consensus among replicas
- Alternative to Paxos, the protocol behind RAFT
- Slightly different goal: replication versus consensus
- Viewstamped Replication uses consensus internally

Assumptions

- Failures are fail-stop
- Does not handled Byzantine failures
- Works in an async environment, similar to Paxos
- Assumes only a minority of replicas fail. To tolerate f failures, total 2f + 1 nodes required

Quorum Intersection Property

- Quorum: f + 1 replicas
- All steps in the protocol executed by a quorum
- Quorum intersection property: any two quorums must intersect in at least one node
- We saw this in Paxos as well

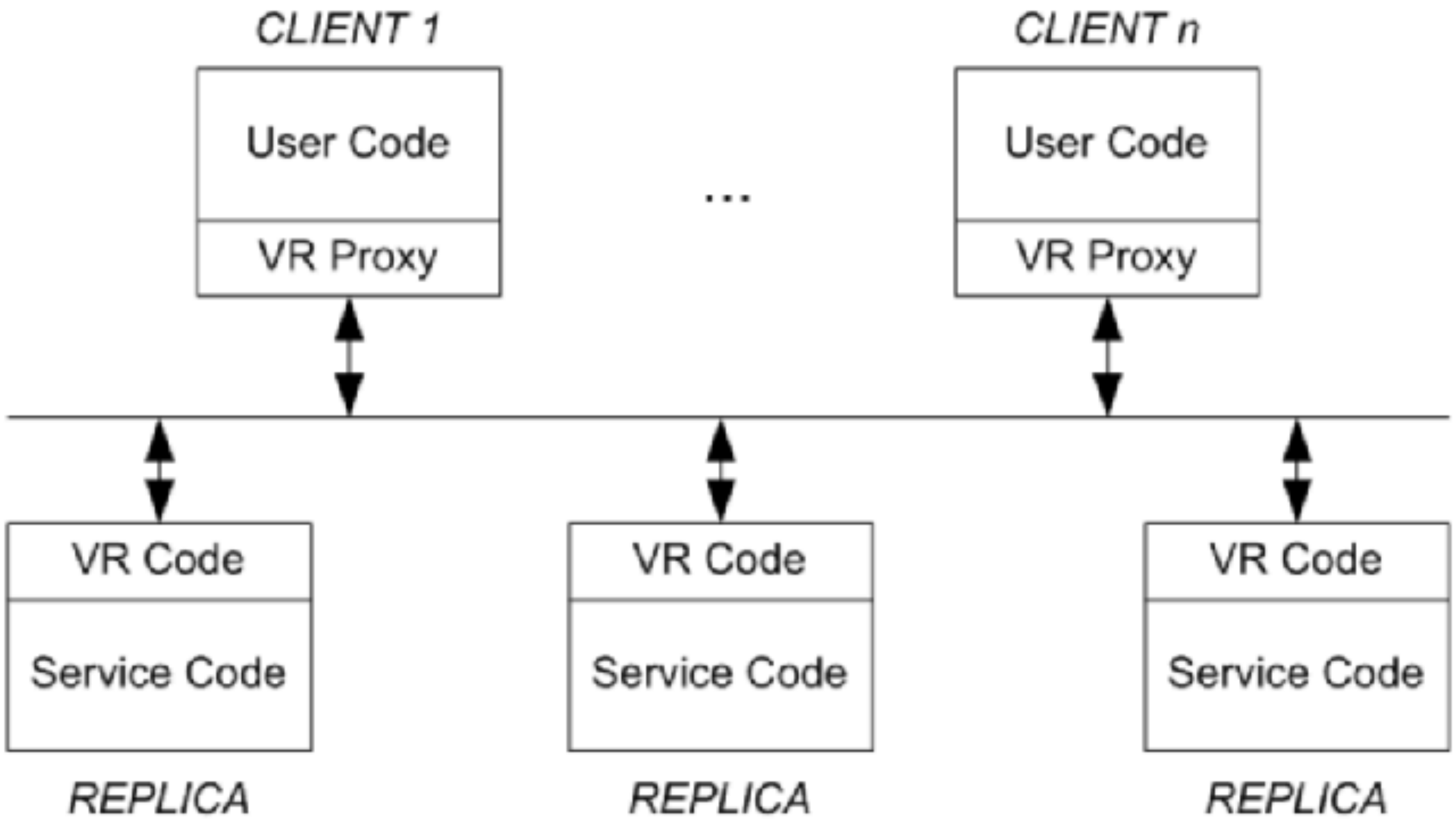


Figure 1: VR Architecture; the figure shows the configuration when $f = 1$.

Performing Updates

- Updates are serialized and performed by a primary replica
- The order determined by the primary is used by the backup replicas
- The system moves through “views”: each view has a primary replica
- If the primary fails, a view change protocol selects a new primary

View Changes

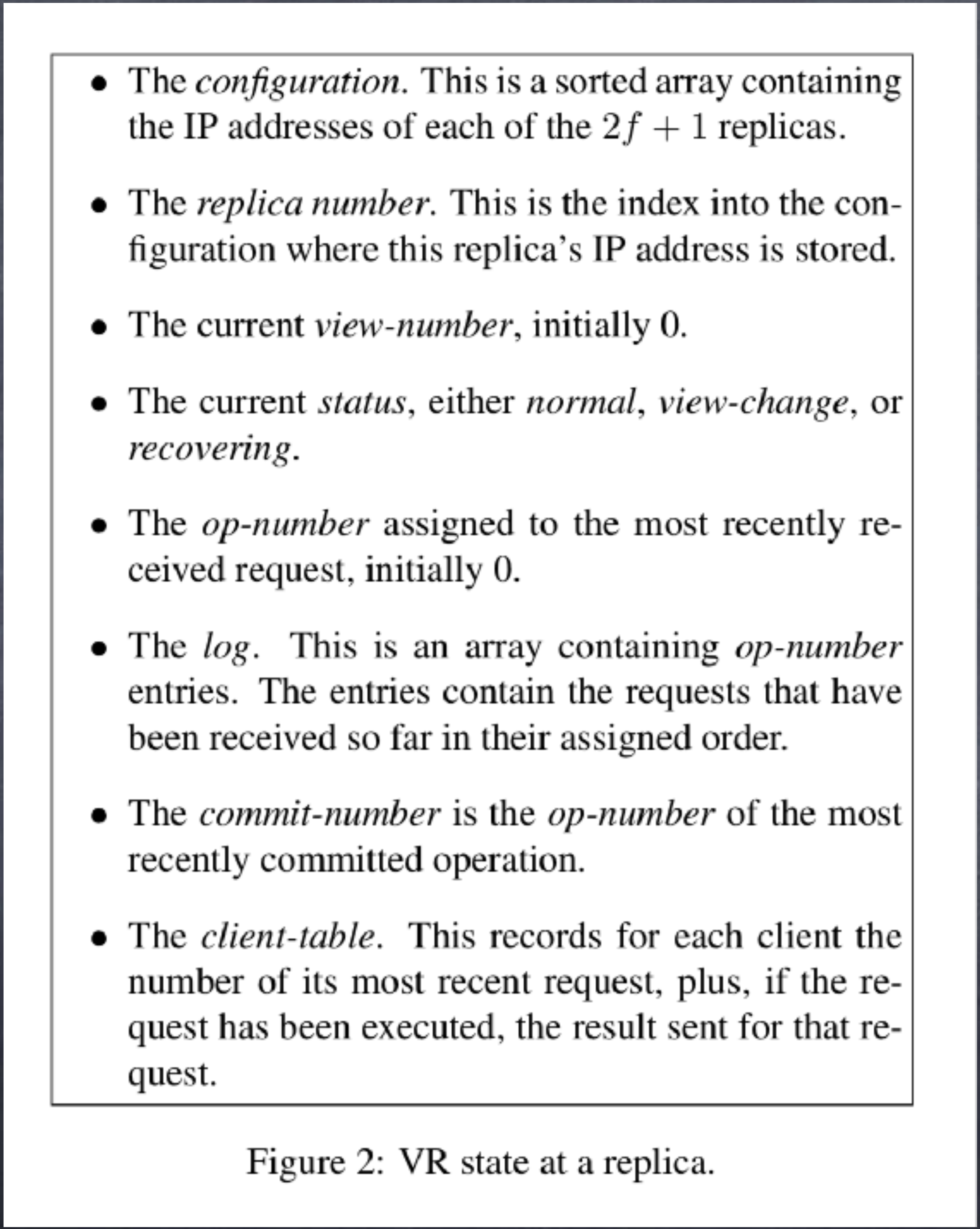
- A view change should not result in old updates being lost
- A view change is processed by a quorum
- Each request is also handled by a quorum
- Hence there will be at least one node that remembers the update before the view change

Recovering from failure

- A replica that recovers from a transient failure can rejoin the cluster
- Can only rejoin when its state is equivalent to its state at the failure point
  - VR does not keep disks, which would trivially satisfy this requirement

Selecting the primary

- Primary selected in round robin fashion
- Replica i is the primary in view (i mod n)
- Given view number and number of replicas, primary can be computed



- The *configuration*. This is a sorted array containing the IP addresses of each of the $2f + 1$ replicas.
- The *replica number*. This is the index into the configuration where this replica's IP address is stored.
- The current *view-number*, initially 0.
- The current *status*, either *normal*, *view-change*, or *recovering*.
- The *op-number* assigned to the most recently received request, initially 0.
- The *log*. This is an array containing *op-number* entries. The entries contain the requests that have been received so far in their assigned order.
- The *commit-number* is the *op-number* of the most recently committed operation.
- The *client-table*. This records for each client the number of its most recent request, plus, if the request has been executed, the result sent for that request.

Figure 2: VR state at a replica.

Normal Operation

- Condition: view number of node has to match view number of request
- Client -> Primary REQUEST
- Primary -> Replicas PREPARE
- Replicas -> Primary PREPARE OK
- Primary -> Replicas COMMIT

View Changes

- Replica -> Others STARTVIEWCHANGE (view++)
- Replicas -> New Primary DOVIEWCHANGE with log
- New Primary uses longest log in new view
- New Primary -> Replicas STARTVIEW

Recovery

- Recovering node (RN) -> Replicas RECOVERY
- Replicas -> RN RECOVERYRESPONSE, with logs
- RN updates its log based on response from Primary

Optimizations

- Fast reads at the primary
- Witnesses
- Batching

Reconfiguration

- PREPARE with new config
- Increment epoch number
- COMMIT to old replicas that are still in config
- STARTEPOCH to new replicas that are being added
- Nothing to old replicas not part of new config
- Important: while transitioning, no new requests are accepted

Reconfig: members of new group

- STARTEPOCH or COMMIT message
- Records old and new configs
- New epoch-number, view 0
- status: transitioning
- catches up state is required
- status: normal
- EPOCHSTARTED to replicas that are being replaced

Reconfig: members being replaced

- COMMIT message
- Records old and new configs
- New epoch-number, view 0
- status: transitioning
- waits for f+1 EPOCHSTARTED
- shuts down