Replication for Fault Tolerance - Quorums-Consensus Replicated ADT

Initial and Final Quorums

Quorum-Consensus and Replicated Abstract Data T.

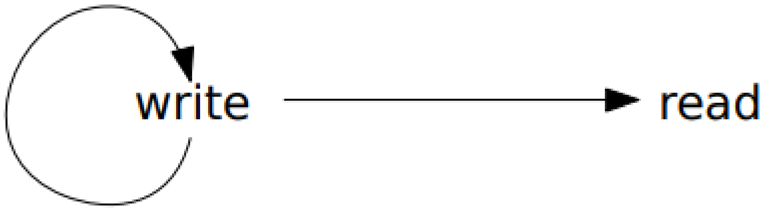
Quorum: any set of replicas whose cooperation is sufficient to execute the operation

- When executing an operation, a client:
    - reads from an **initial quorum**
    - writes to a **final quorum**
- A quorum for an operation is any set of replicas that includes both an initial and a final quorum.
- (m, n), where m = size of its initial quorum and n = size of its final quorum

Quorum intersection constraints are defined between the final quorum of one operation and the final quorum of another

Gifford's Read/Write Quorums

- Object (e.g. file) read/write operations are subject to two constraints:
  1. Each final quorum for write must intersect each initial quorum for read
  2. Each final quorum for write must intersect each initial quorum for write

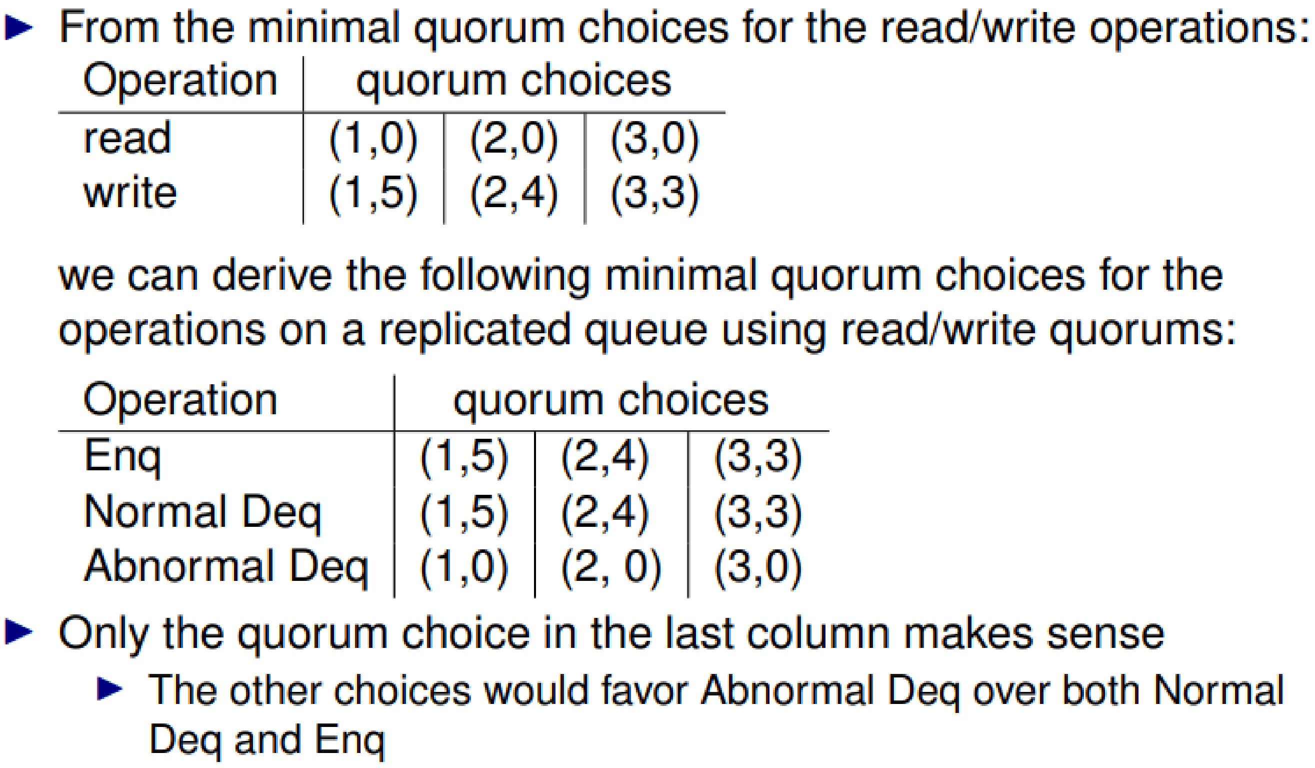


- Gifford's read/write quorums can be used to implement arbitrary data types

A queue has two basic operations:

- **Enq**: adds an item to the queue
- **Deq**: removes least recent item from the queue, raising an exception if the queue is empty
    1. Read an initial **read quorum** to determine the current version of the queue
    2. Read the state from an updated replica
    3. If the queue is not empty, **normal deq**
        - Remove the item at the head of the queue
        - Write the new queue state to a final write quorum

    - Return the item removed
4. If the queue is empty, **abnormal deq**, raise an exception



- From the minimal quorum choices for the read/write operations:

Operation	quorum choices		
read	(1,0)	(2,0)	(3,0)
write	(1,5)	(2,4)	(3,3)

we can derive the following minimal quorum choices for the operations on a replicated queue using read/write quorums:

Operation	quorum choices		
Enq	(1,5)	(2,4)	(3,3)
Normal Deq	(1,5)	(2,4)	(3,3)
Abnormal Deq	(1,0)	(2, 0)	(3,0)

- Only the quorum choice in the last column makes sense
    - The other choices would favor Abnormal Deq over both Normal Deq and Enq

(Herlihy's) Replicated ADTs

- **Timestamps** instead of version numbers
- **Logs** instead of (state) versions
- Clients are able to generate timestamps that can be totally ordered

Replicated Read/Write Objects with Timestamps

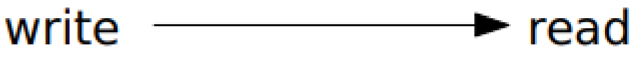
Read

Similar to the version-based, except that a client uses the timestamp instead of the version to identify a replica that is up-to-date

Write

There is no need to read the versions from an initial quorum:

Quorum Intersection Graph



Replicated Event Logs vs Replicate State

Event: State change, represented as a pair of

- **Operation** with respective arguments (Read() or Write(x))
- **Outcome** a termination condition and returned results (Ok(x) or Ok())

E.g. [Read(), Ok(x)] and [Write(x), Ok()]

Event log: a sequence of log entries

- **Log entry**: timestamped event (t0: [Enq(x); Ok()])

Idea: rather than replicate state, replicate event logs
- an event log subsumes the state

Replicated Queue: an example of replicated ADT

Herlihy's Replicated Queue

Deq implementation - Client:

1. reads the logs from an inital Deq quorum and creates a **view**
    - log obtained by:
        - merging in timestamp order the entries of a set of logs
        - discarding duplicates
2. reconstructs the queue state from the view and find the item to return
3. if the queue is not empty:
    - records the Deq event, by appending a new entry to the view
    - sending the modified view to a final Deq quorum of replicas
4. returns the response to Deq's caller

Constraints

1. Every initial Deq quorum must intersect every final Enq quorum
2. Every initial Deq quorum must intersect every final Deq quorum

Optimizations

Problem: logs and messages grom indefinitely
Solutions:

- Garbage collect logs
- Cache logts ae clients

Deq Implementation

1. Clients reads from an initial Deq Quorum
2. The client:
    1. creates a view as before
    2. discards all entries arlier than the latest observed horizon time

3. The oldest remaining Enq entry indicates
    1. the item to dequeue
    2. the new horizon time
4. The client writes the new horizon time to a final Deq quorum

Critical Evaluation

Issues with Replicated ADTs

- Timestamps generated by clients and consistent with linearizability
- Logs must be garbage collected to bound the size of messages

(Herlihy’s) Replicated ADTs vs CvRDTs

- They both support replicated data types
- Herlihy’s logs appear to be a monotonic semi-lattice object.
- They both require design ingenuity

Differences

- CRDTs do not ensure strong consistency, but strong eventual consistency
- However, CRDTs will likely be more available, an operation can be executed as long as there is one accessible replica
    - Replicated ADTs require a quorum to perform one operation

Summary

- Quorum-based systems are usually restricted to "simple" data storage systems
- SMR with Paxos, uses majority voting – which is a special kind of quorum
- Quorums need not be restricted to assigning/counting votes
- Quorums may be dynamic, e.g. by changing the replicas and/or the vote assignments