HQ Replication

Hybrid Quorum Protocol for Byzantine Fault Tolerance

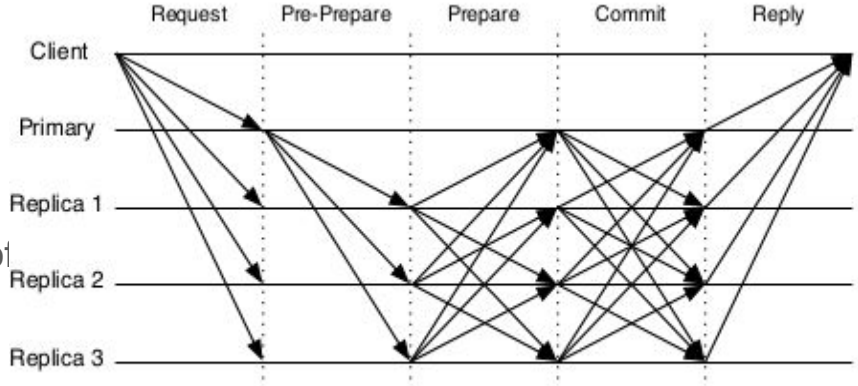
presentation based on HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance, Nov 2006

Byzantine-fault-tolerant state machine replication

- Q/U
  - Query/Update, quorum based approach
- BFT
  - communication between replicas
- **HQ**
  - Combines Q/U with BFT
  - Default: Q/U approach, BFT in case of contention

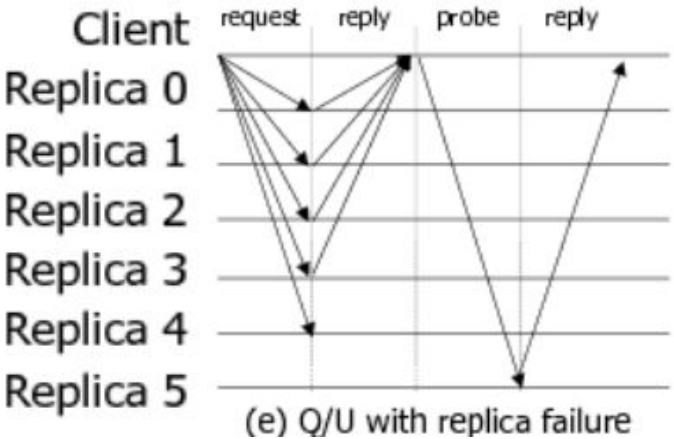
BFT , Q/U

- BFT
  - All-to-all communication to agree on the order of operations
  - 3*f*+1 to tolarate *f* failures



(b) Agreement-based: BFT

- Q/U
  - Requires 5*f* +1 replicas to tolerate *f* failures (m than thoretical minimum of 3*f*+1)
  - Q/U performs poorly when there is contention (exponential back-off to resolve contention with write operations)



(e) Q/U with replica failure

Hybrid Quorm (HQ)

- $3f + 1$ replicas to survive, $2f + 1$ quorum
- Quorum and agreement-based state machine replication
- Scalability

No contention vs contention

- Switch to BFT when there is contention

Model

Byzantine failures

Asynchronous

Assume:

- Unforgeable digital signatures
- Trusted key distribution
- Collision-resistant hash func
- Non repeating nonces chosen by clients

HQ Replication, *normal case*

- Reads do not modify the state
  - One phase
- Writes do
  - Two phase write when **no contention**
  - quorum of msg - certificate

phase-1

- client issues RPC call to obtain certificate, (operation# at timestamp *t*)

phase-2

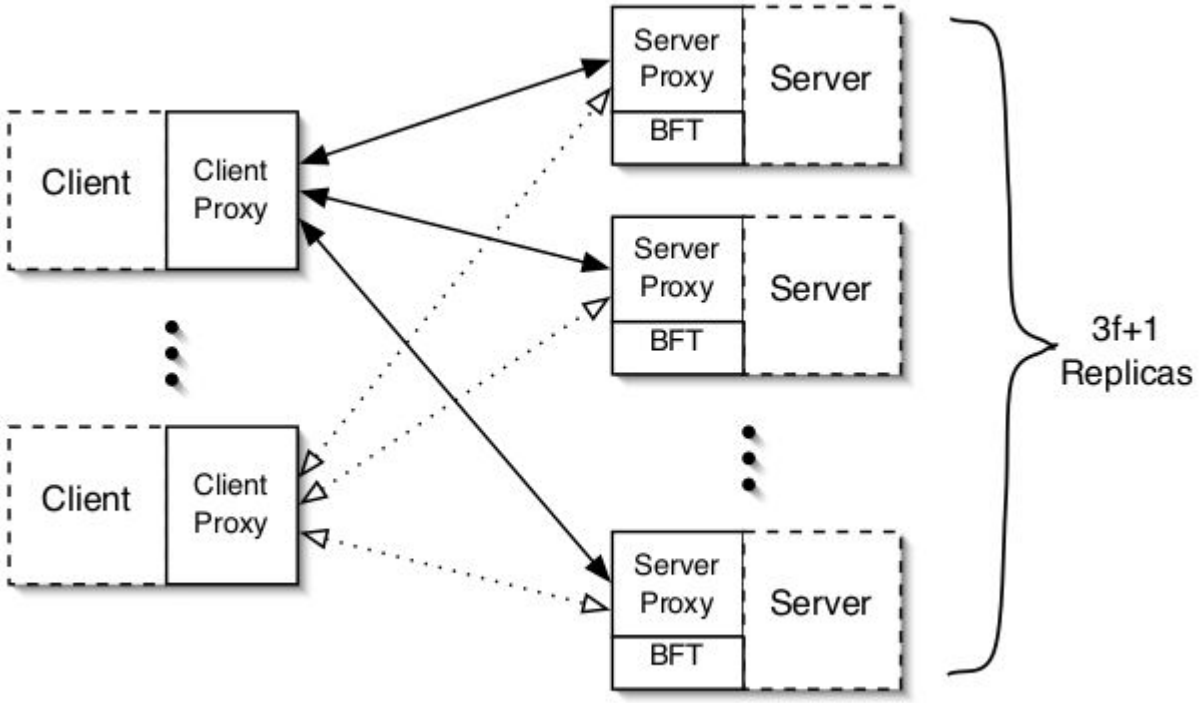
- using obtained certificate to make replicas execute operation at given timestamp,
- writeBackWrite, writeBackRead

HQ Replication, *contention*

- with contention client can't obtain usable certificate
- ask system to resolve contention for the timestamp
- HQ contention resolution uses BFT

  - *1.* if phase-2 completed for operation ***o*** at ***t*** , ***o*** will continue to be assignet to ***t***
  - *2.* if client has obtained certificate to run ***o*** at ***t*** but ***o*** has not completed yet, ***o*** will run at other timestamp **>= *t***
  - replicas maintain single backup state in case *2.* to undo the last write

system architecture



protocol details, normal case

- system supports multiple objects
- for simplicity (example) each operation concerns single object
- multiple operations allowed in parallel but on **distinct** objects

write certificate contains quorum of *grants*, each of form:

$$\langle cid,\ oid,\ op\#,\ h_{op},\ vs,\ ts,\ rid \rangle_{\sigma_r}$$

grant is signed by replica *r,* with *rid. r* grants client *cid* to run operation numbered *op#* with hash *h_op,* an object *oid,* at timestamp *ts*

write-1-phase, Client

$\langle \text{WRITE-1}, cid, oid, op\#, op \rangle_{\sigma_c}$

$\langle \text{WRITE-1-OK}, grantTS, currentC \rangle$

$\langle \text{WRITE-1-REFUSED}, grantTS, cid, oid, op\#, currentC \rangle_{\mu_{cr}}$

$\langle \text{WRITE-2-ANS}, result, currentC, rid \rangle_{\mu_{cr}}$

write-2-phase, Client

send $\langle \text{WRITE-2}, writeC \rangle$

wait for $\langle \text{WRITE-2-ANS}, result, currentC, rid \rangle_{\mu_{cr}}$

it will receive quorum of <Write-2-Ans … unless there is contention

read protocol

send $\langle \text{READ}, cid, oid, op, nonce \rangle_{\mu_{cr}}$

receive $\langle \text{READ-ANS}, result, nonce, currentC, rid \rangle_{\mu_{cr}}$

waits for quorum of ***read-ans*** and then outputs to calling app

processing at the replicas

•currentC, the certificate for the current state of the object.

• grantTS, a grant for the next timestamp, if one exists

• ops, a list of WRITE -1 requests that are currently under consideration (the request that was granted the next timestamp, and also requests that have been refused), plus the request executed most recently.

• oldOps, a table containing, for each client authorized to write, the op# of the most recently completed write request for that client together with the result and certificate sent in the WRITE -2- ANS reply to that request.

• vs, the current viewstamp (which advances each time the system does contention resolution).

write-1 request

request: $\langle \text{WRITE-1}, cid, oid, op\#, op \rangle_{\sigma_c}$

if $op\# < oldOps[cid].op\#$ **->** request is old and gets discarded

if $op\# = oldOps[cid].op\#$ **->** replica returns <Write-2-Ans response containing result and certificate stored in oldOps[cid]

write-2 request

request to node: $\langle \text{WRITE-2}, writeC \rangle$

- any node is permitted to run the request. the meaning of request depends on the certificate, not the id of sender
- certificate has client ***c*** that run phase-1 write, ***oid*** and ***op#***
- replica looks up oldOps to identify duplicate requests, discards them and returns responses stored in oldOps[***c***]
- otherwise replca executes operation ***if***
- $vs = writeC.vs$ and $currentC.ts = writeC.ts - 1.$
- if this information isn't available replica contacts others and executes earlier operations.

Contention resolution

Contention occurs when several clients are competing to write at the same timestamp.

Clients detect contention when they receive conflicting grants from replicas with same timestamps.

- (but can also be faulty client sending different requests to different replicas)

resolve by sending: $\langle \text{RESOLVE}, \mathit{conflictC}, \mathit{w1} \rangle$

w1 ir <Write-1 request that led to conflict being discovered

Contention resolution ...

BFT state machine protocol is used to resolve contention

- one ***primary*** replica
- BFT only for reaching agreement on a deterministic ordering of the conflicting updates
- additional states: ***conflictC, backupC, prev***
- replica thats resolving <Resolve> is *frozen* by seting varialble *conflictC* to non nill (delays write and resolve client requests)
-

Contention resolution ...

$$\langle \text{RESOLVE}, \ conflictC, \ w1 \rangle$$

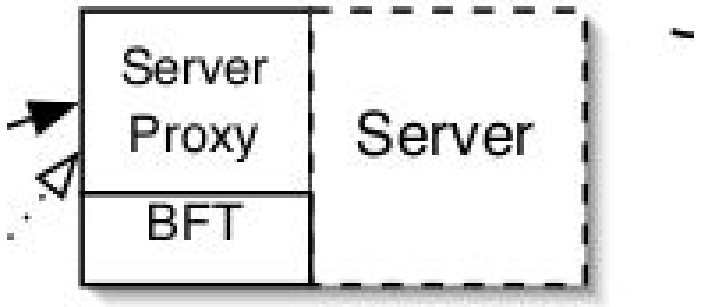
If **currentC** is later than **clientConflictC,** or if they mach but its already executed according to **oldOps** the conflict is already **resolved.**

Otherwise **conflictC** is set to **clientConflictC** and replica adds *w1* to *ops* if it's not there, then it sends $\langle \text{START}, conflictC, ops, currentC, grantTS \rangle_{\sigma_r}$

to the server proxy running at BFT :

when quorum of <Start> is received

replica creates BFT resolve operation



Contention resolution ...

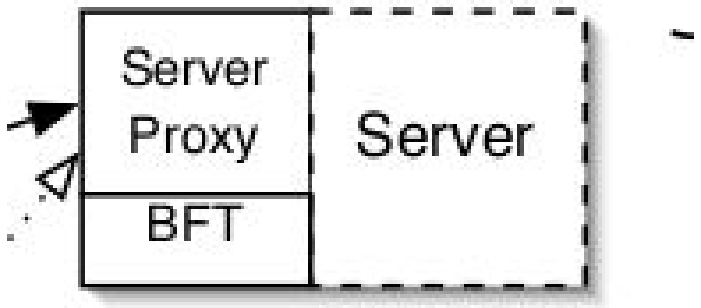
$$\langle \text{RESOLVE}, \ conflictC, \ w1 \rangle$$

quorum of <Start> is *startQ*

replica executes operation on BFT with funciton argument *startQ*

the server proxy is “client” of BFT

BFT replica set is the same set as **HQ** set



resolution ...

$\langle \text{RESOLVE}, \ conflictC, \ w1 \rangle$

check if *grant* is valid - forms a certificate (quorum)

check if it has to undo previous operation -> cas wen *currentC* is later than *startQ.currentC*

next: replica executes operation from *startQ.currentC* and is now up to date. *oldOps* is now identical at all ***honest replicas***.

next: operations from startQ.ops are executed in order, but before it has to gather certificates for every operation, so it sends grant for every operation in ops to other replicas.

after quorum of grants is received, it executes and updates ops, oldOps, currentC

the conflict is resolved, conflictC is cleared (replica is unfrozen). processing is like with WRITE-1 request

Optimizations

Early Grants

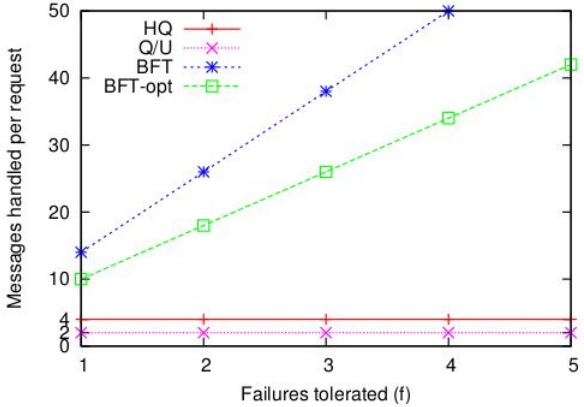
MACs instead of signing

Preferred quorums

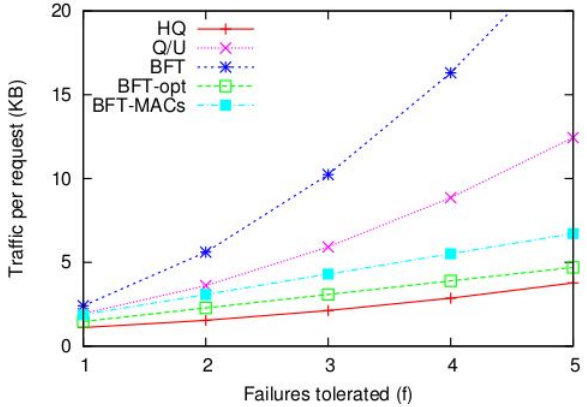
BFT improvement TCP instead of UDP…

Analysis

server:

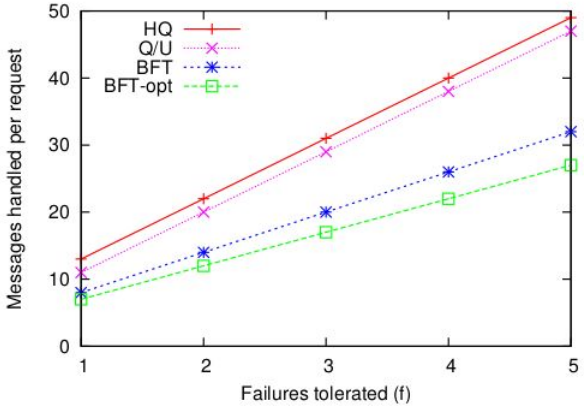


(a) Server

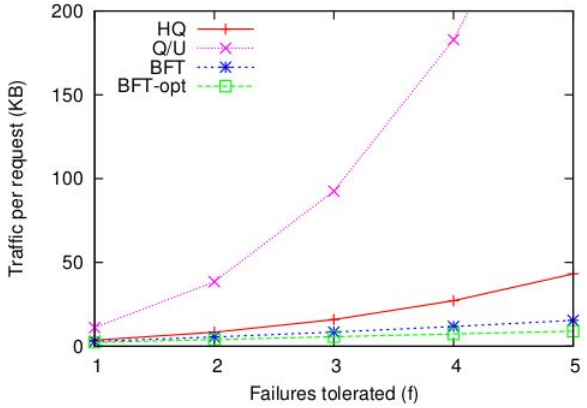


(a) Server

client:



(b) Client



(b) Client

Gorums