Replication for Fault Tolerance - Byzantine Quorums

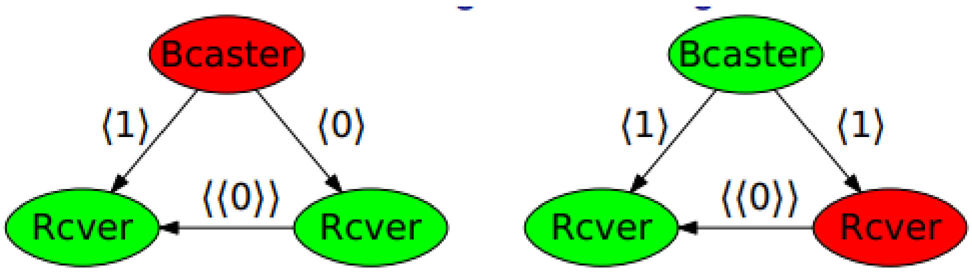
Consensus with Byzantine Failures

- **Byzantine Generals Problem (BGP)** this is reliable broadcast
    - **Agreement**: all non-faulty processes deliver the same message
    - **Validity**: if the broadcaster is non-faulty processes deliver the message sent by the broadcaster

There is no solution to the BGP in a system with 3 processes, if one of them may be Byzantine, if messages are not signed

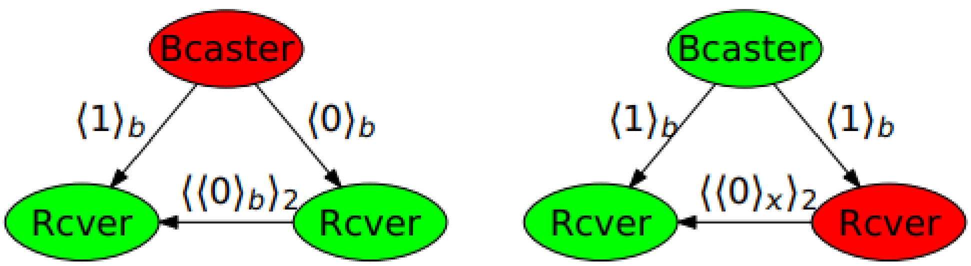
Byzantine Failures and Cryptographic Signatures

Communication without signed messages



- The LHS receiver gets the same messages in both scenarios
    - It should deliver the same message in both scenarios
    - But no message satisfies the desired properties in both cases

Communicatio with signed messages



- Byzantine receiver cannot properly sign a modified message

Quorum Consensus with Byzantine Replicas

Quorum Consensus

- Each (replicated) operation (e.g. read/write) requires a quorum
- Fundamental Property:
  - If the result of one operation depends on the result of another, then their quorums must overlap, i.e. have common replicas
- consider all replicas as peers.

Byzantine Quorums: Model

Processes: Some servers in U may exhibit byzantine failures
Network: any two processes (clients or servers) can communicate over an authenticated and reliable point-to-point channel

Problem Definition

- Clients perform read and write operations on a variable x replicated at all servers (in U)
- Each server stores a replica (copy) of x and a timestamp t
- Timestamps are assigned by clients when the client writes the replica

Safety: any read that is not concurrent with writes returns the last value written in some serialization of the preceding writes

- Operations are assumed to have **begin and end** events that determine a partial order
- Essentially, this means **linearizability** of read/write operations

Access Protocol

Write of value v by a client c

1. c queries all servers of some quorum Q to obtain a set of timestamps $A = \{t_u : u \in Q\}$
2. c chooses a timestamp $t \in T_c$ greater than
   - the highest timestamp value in A
   - any timestamp it has chosen previously
3. c sends the update $\langle v, t \rangle$ to servers until it has received an acknowledgement from every server in some quorum Q'

   A server updates its local copy and timestamp to the received values $\langle v, t \rangle$, only if t is greater than the timestamp of its current copy
   - In both cases, it returns an acknowledgement to the client

Read of variable x by client c

1. c queries all servers of some quorum Q to obtain a set of value/timestamp pairs $A = \{(v_u, t_u) : u \in Q\}$
2. c applies a **deterministic function** $Result()$ to A to obtain the result $Result(A)$ of the read operation

Gifford

Read

1. Collect a read quorum, to find out the current version
2. 2. Read the object state from an up-to-date replica.

Write

1. Collect a read quorum, to find out the current version
2. 2. Write the new value with the new version to a write quorum

Asynchrony

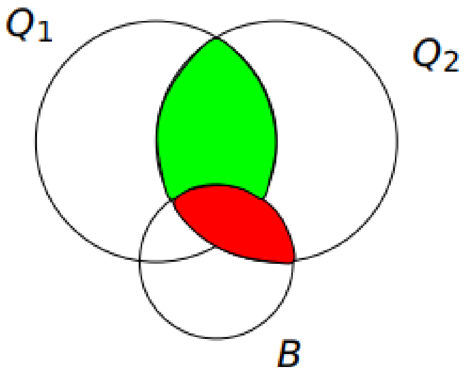
- Both in the read and write operations, a client has to get replies from all servers in a quorum

Beginning and End Events

- **write** of value v with timestamp t
  - **begin** when the client initiates the operation
  - **end** when all correct servers in some quorum have processed the update (v, t)
- **read**
  - **begin** when the client initiates the operation
  - **end** when the Result() function returns, thus determining the read result

- **op1 precedes op2** if op1 ends before op2
- **op1 and op2 are concurrent** if neither op1 precedes op2, nor op2 precedes op1

Byzantine Masking Quorums



Q_1 latest write quorum

B set of byzantine nodes

Q_2 read quorum

$(Q_1 \cap Q_2) \setminus B$ servers with up-to-date values

$Q_2 \setminus (Q_1 \cup B)$ servers with stale values

$Q_2 \cap B$ arbitrary values

Masking Quorum System, $\mathcal{Q}$ for a fail-prone system $\mathcal{B}$ if:

M-consistency

$\forall Q_1, Q_2 \in \mathcal{Q}, \forall B_1, B_2 \in \mathcal{B} : (Q_1 \cap Q_2) \setminus B_1 \not\subseteq B_2$

M-availability

$\forall B \in \mathcal{B} : \exists Q \in \mathcal{Q} : B \cap Q = \emptyset$

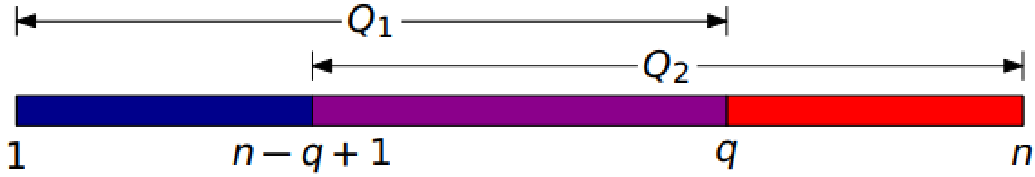
Size-based Byzantine Masking Quorums

M-consistency

- client always obtains an up-to-date-value

Assuming all servers are peers

- Let f be the bound on faulty servers
- Then we need at least f + 1 up-to-date non-faulty servers
- Thus every pair of quorums must intersect in at least 2f + 1 servers



n: number of servers
q: quorum size
f: upper bound on byzantine servers

$$q-(n-q+1)+1 \geq 2f+1$$
$$2q-n \geq 2f+1$$

M-availability

$\forall B \in B : \exists Q \in Q : B \cap Q = \varnothing$

Assuming all servers are peers

- Let f be the upper-bound on byzantine servers
- Let q be the size of a quorum
- Let n be the number of servers
- Then $n - f \geq q$

Non-byzantine Read-Write Quorums Based on Size

$w \geq f + 1$ (1) this ensures that writes survive failures
$w + r > n$ (2) this ensures that reads see most recent write
$n - f \geq r$ (3) this ensures read availability
$n - f \geq w$ (4) this ensures write availability

- increasing n only worsens performance (as it requires larger quorums)
    - by increasing n we can increase fault tolerance, as f can increase

Byzantine Quorums Based on Size: Read operation

Read (generic) of variable x by client c

1. c queries all servers of some quorum Q to obtain a set of value/timestamp pairs $A = \{(v_u, t_u) : u \in Q\}$
2. c applies a **deterministic function** $Result()$ to A to obtain the result $Result(A)$ of the read operation

Read for size-based quorums of variable x by client c

1. c queries servers until it gets a reply from a set Q of $3f + 1$ **different servers**. Let $A = \{(v_u, t_u) : u \in Q\}$ be the **set** of value/timestamp pairs received from at least $f + 1$ servers
2. the result of the read operation is the value returned by $Result(A)$
   Where $Result(X) = \{v : (v, t) \in X \wedge t \geq t', \forall (v', t') \in X\}$
   I.e. $Result(X)$ returns the value of the value/timestamp pair with the highest timestamp in the set X or $\perp$, if no such value
   - It is possible a read to fail because of concurrent writes

Safety

Any read that is not concurrent with writes returns the last value written in some serialization of the preceding writes

Safety - No concurrent writes

Scenario the begin event of a write operation occurs always after the end event of the write operation that precedes it

Consider the latest write, (v, t) i.e. the one completed before the read

All correct servers of some quorum have updated their local state with (v, t) by definition of the end event of a write operation and by the writing protocol

The read client gets a reply from at least $2f + 1$ servers of that quorum by the read protocol the client gets a reply from a server quorum and any two quorums intersect in at least $2f + 1$ servers

At least $f + 1$ of these servers are correct since f is the maximum number of faulty servers

$(v, t) \in A$ because the read client receives the (v, t) of the last write from at least $f + 1$ servers

$Result(A)$ returns the value v written in the last write because the timestamp of (v, t) is higher than that of the previous writes, by the writing protocol

Safety - With concurrent writes

Scenario the begin of a write operation **may** occur always before the end event of a write operation whose begin event happened earlier

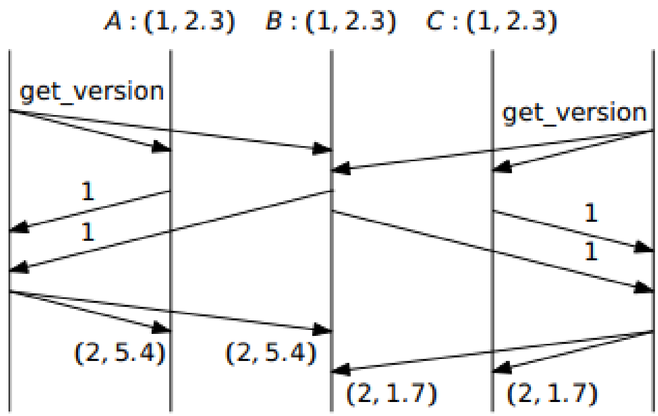
But the read operation **is not concurrent** with any write

Thus, it suffices to argue that the concurrent execution of the write protocol ensures that once all write operations end, the correct servers of a quorum have applied the update (v, t_{max}) with the highest timestamp to their local copy last

- Timestamps are totally ordered and every write operation has a different timestamp, by the rule used to choose timestamps in the writing protocol
- (v, t_{max}) is received by a quorum of servers, by the write protocol
- (v, t_{max}) is always applied by the correct servers in this quorum because t_{max} is the highest timestamp of the concurrent writes and by the writing protocol
- Once (v, t_{max}) is applied, no other concurrent update is applied by the same reasons as above.

Naïve Implementation with Concurrent Writes

- $N = 3, N_R = 2, N_W = 2$
- Two clients attempt to write the replicas at more or less the same time
- The two write quorums are not equal, even though they overlap
- Again, replicas end up in an inconsistent state.



Server Failures and Liveness

- By M−Availability, there is always a quorum, even if all f byzantine server fail
- By the write protocol:
  - a client sends the update (v, t) to servers until it has received an acknowledgement from every server in some quorum Q'
  - **A server** only updates its local copy and timestamp to the received values (v, t), only if t is greater than the timestamp of its current copy

Dissemination Quorum Systems

Application repositories of self-verifying information

- I.e. apps. in which clients can verify the validity of the information, i.e. where clients can detect the tampering of servers. E.g.

  Repositories of public keys they are signed by CAs
  Blockchains we will cover them in coming lectures

Dissemination Quorum System, $\mathcal{Q}$ for a fail-prone system $\mathcal{B}$ if:

D-consistency
$\forall Q_1, Q_2 \in \mathcal{Q}, \forall B \in \mathcal{B} : (Q_1 \cap Q_2) \not\subseteq B$

D-availability
$\forall B \in \mathcal{B} : \exists Q \in \mathcal{Q} : B \cap Q = \emptyset$

Safety

1. any read that is not concurrent with writes returns the last value written in some serialization of the preceding writes
2. **a read that is concurrent with one or more writes returns**
   - either the value written by the last preceding write
   - or any of the values being written in a concurrent write

Read Operation

Read operation of variable x by client c

1. c queries servers until it gets a reply from some quorum Q. Let $A = \{(v_u, t_u) : u \in Q\}$ the set of value/timestamp pairs received
2. The client then:
   - 2.1 discards those pairs that fail the verification algorithm
   - 2.2 chooses the pair (v, t) with the highest timestamp of the remaining pairs
   - 2.3 the value v is the result of the read operation.

IMP. timestamps must be part of the verified information

- Otherwise, byzantine servers may mess with these and break the protocol

Note: although an attempt to tamper with a value is detected, a byzantine server can send a stale value/timestamp pair.

Size-based Byzantine Dissemination Quorums

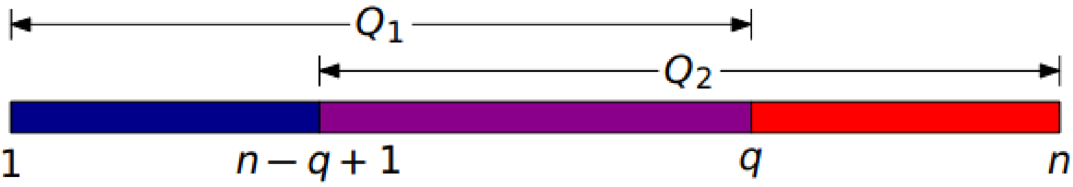
D-consistency

$\forall Q_1, Q_2 \in \mathcal{Q}, \forall B \in \mathcal{B} : (Q_1 \cap Q_2) \not\subseteq B$

- This ensures that there is at least a non-byzantine server in the intersection of every two quorums

Assuming all servers are peers

- Let f be the bound on faulty servers
- Thus every pair of quorums must intersect in at least $f+1$ servers



n: number of servers
q: quorum size
f: upper bound on byzantine servers

$q-(n-q+1)+1 \geq f+1$
$2q-n \geq f+1$

D-availability

$\forall B \in \mathcal{B} : \exists Q \in \mathcal{Q} : B \cap Q = \emptyset$

- This is required for liveness

Assuming all servers are peers

- Let f be the upper-bound on byzantine servers
- Let q be the size of a quorum
- Let n be the number of servers
- Then $n - f \geq q$

Combining with the inequality derived from D-consistency:

$2q - n \geq f + 1$
We get:
$2(n-f) - n \geq f + 1$
Thus:
$n \geq 3f + 1$
Let $n = 3f + 1$
Then:
$q = 2f + 1$

Critical Evaluation

- Compared with state machine replication, Byzantine Quorums appear to require fewer messages, by a large margin
- But the protocols we have seen assume that:

    - Either clients can be trusted
    - Or the information stored is self-verifiable
- To handle other cases, in Phalanx (the SRDS 1998 paper) Malkhi and Reiter use **consensus-objects**, which appear to require about the same number of messages as Byzantine SMR
- Furthermore, these protocols support only read/write operations.