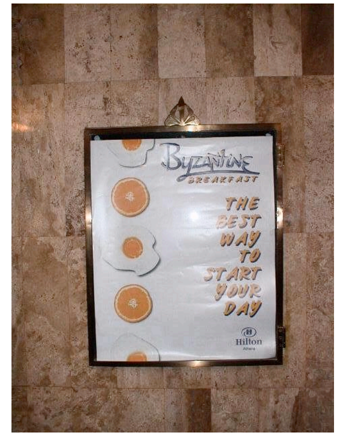
Byzantine Quorum Systems

by

Anurag Agarwal



Courtesy : *From Byzantine Agreement to Practical survivability*

D. Malkhi

System Model

[MR98]

- Universe U of servers |U| = *n*
- Quorum system $\mathcal{Q} \subseteq 2^U$
- Byzantine faulty servers
  - Modeled as *fail-prone system* $\mathcal{B} \subseteq 2^U$
  - $\forall B_1, B_2 \in \mathcal{B} : B_1 \neq B_2 \Rightarrow B_1 \not\subseteq B_2$
  - Some $B \in \mathcal{B}$ contains all the faulty servers
- Initially clients assumed to be correct
- Point to point authenticated and reliable asynchronous channel

Access Protocol

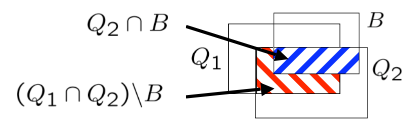
- Each server stores value *v* and timestamp T
- Client c chooses timestamps from set T_c
- For clients c and c', T_c and $T_{c'}$ don't intersect
- **Write (v)**
  - Query servers to get a set A of timestamps for some quorum Q
  - Choose a timestamp t in T_c greater than highest in A and greater than any of its previous timestamps
  - Send update <v,t> to servers until acknowledgement from some quorum Q' is received

Access Protocol

- **Read()**
  - Query servers to obtain value/timestamp pairs A for some quorum Q
  - Applies a deterministic function “Result(A)” to obtain the result of read operation
- “Result” function
  - Depends on the type of quorum systems
  - Chosen to guarantee safe semantics

Masking Quorum Systems

- A quorum system $\mathcal{Q}$ is a masking quorum system for a fail prone system $\mathcal{B}$ if the following properties are satisfied :
  - M-consistency
    $\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B_1, B_2 \in \mathcal{B} : (Q_1 \cap Q_2) \backslash B_1 \not\subseteq B_2$
  - M-availability
    $\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q} : B \cap Q = \phi$



Masking Quorum Systems

- Read : “Result()” function
  - Client receives the responses $A = \{< v_u, t_u >\}_{u \in Q}$
  - Computes the set

$$A' = \{< v, t >: \exists B^+ \subseteq Q[\forall B \in \mathcal{B}[B^+ \not\subseteq B] \wedge \forall u \in B^+[v_u = v \wedge t_u = t]]\}$$

  - Chooses the pair $< v, t >$ in A' with highest timestamp

f – masking Quorum System

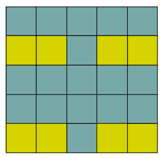
- $\mathcal{B} = \{B \subseteq U : |B| = f\}$
- M-consistency
  $\forall Q_1, Q_2 \in \mathcal{Q} : |(Q_1 \cap Q_2)| \geq 2f + 1$
- M-availability
  $\forall Q \in \mathcal{Q} : |Q| \leq n - f$
- Quorum system exists *iff* $n \geq 4f + 1$

$$\mathcal{Q} = \{Q \subseteq U : |Q| = \lceil \frac{n+2f+1}{2} \rceil\}$$

Grid Quorum

- $\mathcal{B} = \{B \subseteq U : |B| = f\}$, $n = k^2, 3f + 1 \leq k$
- Arrange the universe in a k x k grid
- A masking quorum system (C_j – columns, R_i – rows)

$$\mathcal{Q} = \{C_j \cup \bigcup_{i \in I} R_i : I, \{j\} \subseteq \{1 \ldots k\}, |I| = 2f + 1\}$$



n = 5 x 5, f =1

Dissemination Quorum Systems

- Quorums for self verifying data
  - Only clients can create the data
  - Clients can detect attempted changes by a faulty server
- 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

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

Dissemination Quorum Systems

- Read : "Result()" function
  - Client receives the responses $A = \{< v_u, t_u >\}_{u \in Q}$
  - Computes the set A' of pairs which are verifiable
  - Chooses the pair $< v, t >$ in A' with highest timestamp

f – dissemination Quorum System

- $\mathcal{B} = \{B \subseteq U : |B| = f\}$
- D-consistency

  $\forall Q_1, Q_2 \in \mathcal{Q} : |(Q_1 \cap Q_2)| \geq f + 1$
- D-availability

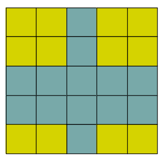
  $\forall Q \in \mathcal{Q} : |Q| \leq n - f$
- Quorum system exists *iff* $n \geq 3f + 1$

$$\mathcal{Q} = \{Q \subseteq U : |Q| = \lceil \frac{n+f+1}{2} \rceil\}$$

Grid Quorum

- $\mathcal{B} = \{B \subseteq U : |B| = f\}$, $n = k^2, 2f + 1 \leq k$
- Arrange the universe in a k x k grid
- A masking quorum system (C_j – columns, R_i – rows)

$$\mathcal{Q} = \{C_j \cup \bigcup_{i \in I} R_i : I, \{j\} \subseteq \{1 \dots k\}, |I| = f + 1\}$$



n = 5 x 5, f =1

Opaque Masking Quorum Systems

- Masking quorums require the client to know the fail prone system $\mathcal{B}$
- Problems
  - Read protocol becomes complicated
  - Revealing possible failure scenarios for which the system is designed
- Design quorums such that clients don't need to know $\mathcal{B}$

Opaque Masking Quorum Systems - Properties

- O-Consistency1:

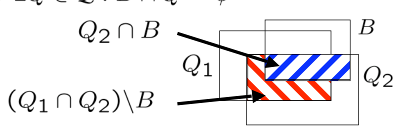
$\forall Q_1, Q_2 \in \mathcal{Q} \forall B \in \mathcal{B} : |(Q_1 \cap Q_2) \backslash B| \geq |(Q_2 \cap B) \cup (Q_2 \backslash Q_1)|$

- O-Consistency2:

$\forall Q_1, Q_2 \in \mathcal{Q} \forall B \in \mathcal{B} : |(Q_1 \cap Q_2) \backslash B| > |(Q_2 \cap B)|$

- O-availability

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



Opaque Masking Quorum Systems

- Read : "Result()" function
  - Client receives the responses $A = \{< v_u, t_u >\}_{u \in Q}$
  - Computes the set A' of pairs which appear most often
  - Chooses the pair $< v, t >$ in A' with highest timestamp
- f-opaque quorum system
  - Exists iff $n \geq 5f$
  - $\mathcal{Q} = \{Q \subseteq U : |Q| = \lceil \frac{2n+2f}{3} \rceil\}$

Byzantine Clients

- Problems
  - Send different updates to different servers
    - Leaves system inconsistent
  - May write data that corrupts the state
    - Impossible for servers to avoid
- Protocol ensures that clients don't leave system in an inconsistent state
- Works for single writer, multiple reader
- Masking Quorum Systems

Client Write protocol

1) Choose a timestamp t in T_c greater than any value it has chosen before
2) Choose a quorum Q and send an update message <update,Q,v,t> to each server in Q
3) If it does not receive ack from all the servers in Q within some time, repeat the steps 2 and 3

Server Write Protocol

- Two sets to remember
  - $B^+ \subseteq Q$ such that $\forall B \in \mathcal{B}, B^+ \not\subseteq B$
  - $Q^- = Q \backslash B$
- Protocol
  - If a server receives a *fresh* <update,Q,v,t>, then send <echo,Q,v,t> to each member of Q
  - If a server receives identical echo messages <echo,Q,v,t> from every server in Q, then it sends a <ready,Q,v,t> to each member in Q

Server Write Protocol

- If a server receives identical ready messages <ready,Q,v,t> from a set B^+, then it sends a <ready,Q,v,t> to each member in Q if it has not done so already.
- If a server receives identical ready messages <ready,Q,v,t> from a set Q^- of servers, then

    (i) if t is greater than its current timestamp, update v and t

    (ii) send an ack to c even if the value was not updated

At this point the server is said to have *delivered* <v,t>

Proof of Correctness

- Duration of write operation
  - Starts when first correct server receives **update** message
  - Ends when all correct servers in a quorum have delivered the update
- Need to show
  - Safe semantics
- Also prove
  - Liveness
  - Completeness

Proof of Correctness

Lemma 1 : A correct server delivers <v,t> only if some correct server previously received <update,Q,v,t>

Proof :

(1) To deliver <v,t>, a correct server must have received **ready** message from a correct server

(2) First **ready** message from a correct server when it has received echo from all the servers in Q

(3) Correct member sends <echo,Q,v,t> only if it receives <update,Q,v,t>

Lemma 2 (Agreement): If a correct server deilvers <v,t> and a correct server delivers <v',t>, then v = v'

Proof : (1) One quorum Q_1 must have sent <echo,Q_1,v,t>

(2) Another quorum Q_2 must have sent <echo,Q_2,v',t>

(3) Q_1 and Q_2 have at least one correct server in common, so v must be identical to v'

Proof of Correctness

Lemma 3(Safe Semantics) : A correct process's read operation that is concurrent with no write operations returns the value written by the last preceding write in some serialization of all preceding write operations.

Proof : Let W be the set of write operations preceding read.

- By Lemma 2, any value/timestamp pair is well defined
- By definition, every write in W was delivered to a full quorum
- By Lemma 1, no correct server has delivered any write outside W

So the read operation will return the value written by the write operation in W with the highest timestamp.

- No write operation in W follows the write operation with highest timestamp because there is a single writer and servers echo request only if its timestamp is higher than the one they have in store

Proof of Correctness

Lemma 4 : $\forall Q \in \mathcal{Q} \forall B_1, B_2, B_3 \in B, Q \not\subseteq B_1 \cup B_2 \cup B_3$

Proof : Assume that there is a $Q \in \mathcal{Q}$ and $B_1, B_2, B_3 \in \mathcal{B}$ such that $Q \subseteq B_1 \cup B_2 \cup B_3$. By M-Availability, $\exists Q' \in \mathcal{Q}, Q' \cap B_1 = \phi$. Then, $Q \cap Q' \subseteq B_2 \cup B_3$, violating M-consistency.

Lemma 5 (Propagation) : If a correct server delivers <v,t>, then eventually there exists a quorum Q such that every server in Q delivers <v,t>

Proof: The correct server that delivered <v,t> received a message <ready,Q,v,t> from each server in $Q^- = Q \backslash B$. Since for some $B' \in \mathcal{B}$, all the members in $Q^- \backslash B'$ are correct, every correct member of Q receives <ready,Q,v,t> from each of the members $B^+ = Q^- \backslash B'$. The messages from B^+ cause each correct member of Q to send a ready message. Hence, <v,t> would be delivered by all correct servers in Q

Proof of Correctness

Lemma 6(Validity) : If a correct client c sends <update,Q,v,t> to every server in Q and all servers in Q are correct, then eventually a correct server delivers <v,t>

Proof : Follows from algorithm

Minimal Quorum Systems

[MAD02-1]

- Reducing quorum size

Best known n	Confirmable	Non-confirmable
Self-verifying	3f + 1	2f + 1
Generic	4f + 1	3f + 1

Minimal Quorum Systems

[MAD02-1]

- Reducing quorum size

Best known n	Confirmable	Non-confirmable
Self-verifying and generic	3f + 1	2f + 1

- Lower bounds independent of self-verifying or generic data !
- Guarantees atomic semantics !!

Semantics

- Consistency semantics defined in terms of conditions when read and write *complete*
  - But when do we say that a write has completed ?
- Confirmable
  - Write completes at the instant when the writer completes its protocol
- Non-confirmable
  - Write completion cannot be determined locally by the writer, but writes are still guaranteed to complete

Key Ideas

- Use different quorum sizes for read and write [MAD02-2]
- AM-Consistency : $\forall Q_r \in \mathcal{Q}_r \forall Q_w \in \mathcal{Q}_w \forall B_1, B_2 \in \mathcal{B} : Q_r \cap Q_w \backslash B_1 \not\subseteq B_2$
- AM-Availability : $\forall B \in \mathcal{B}\ \exists Q_r \in \mathcal{Q}_r : B \cap Q_r = \phi$
- f-threshold case : $|Q_r| = \lceil \frac{n+f+1}{2} \rceil$, $|Q_w| = |Q_r| + f$
- Non-confirmable write : Doesn't wait for acks from servers when writing

Key Ideas

- Use replication in time instead of replication in space
- Previously, 4f +1 needed as some correct servers may not be updated.
- Solution : Wait for those servers to be updated !!

SBQ-L Protocol

- f-threshold
- Confirmable
- Authenticated, reliable, asynchronous, point-to-point channels
- Clients correct
- $Q_w = \lceil \frac{n+f+1}{2} \rceil, Q_r = \lceil \frac{n+3f+1}{2} \rceil$

Algorithm (Client)

Write(v)

- **Ask all servers for their current timestamp t**
- Wait for answer from $|Q_w|$ different servers
- Set ts_c > max({t} U any previous ts_c)
- **Send STORE(v,ts_c) to all servers**
- Wait for acks from $|Q_w|$ different servers

Read()

- send READ to Q_r servers
- loop
  - receive (ANSWER,v,ts) from s in Q_r
  - set answer[s,ts]:= (v,ts)
- until some (v,ts) in answer[ ][ ] is vouched for by $|Q_w|$ servers
- send READ_COMPLETE to Q_r
- return (v,ts)

Modification

- Bound the size of answer[][] array
  - Upon receiving first msg from server s, update T = { f+1 largest timestamps sent by servers}
  - On receiving a (v,ts) from a server s, store if
    - ts is in T
    - ts is the latest timestamp from server s

Proof : Atomicity

Lemma 1: If the protocol is live, it is atomic

a) After write of ts_1, no read returns earlier ts

- Suppose write for ts_1 has completed
- $\lceil\frac{n+f+1}{2}\rceil$ servers acked the write
- At least are $\lceil\frac{n-f+1}{2}\rceil$ correct
- Remaining $\lceil\frac{n+f-1}{2}\rceil$ servers < $|Q_w|$

b) After c reads ts_1, no later read returns earlier ts

- c reads ts_1
  $\Rightarrow \lceil\frac{n+f+1}{2}\rceil$ servers say ts_1
- At least $\lceil\frac{n-f+1}{2}\rceil$ are correct
- Remaining $\lceil\frac{n+f-1}{2}\rceil$ servers < $|Q_w|$
- Any read that starts after ts_1 returns $ts \geq ts_1$

Proof : Liveness

Lemma 2: Every operation eventually terminates

Write: Trivial, because only waits for $|Q_w| < n - f$

Read:
- Consider T after c gets first message from last server.
- Let t_{max} be the largest timestamp from a correct server in T.
- A client never removes t_{max} from its answers[s][], for a correct s
- Eventually, all correct servers see a write with ts = t_{max} and echo client
- Since $Q_r = \lceil\frac{n+3f+1}{2}\rceil, |Q_w| \leq |Q_r| - f$ and the read terminates

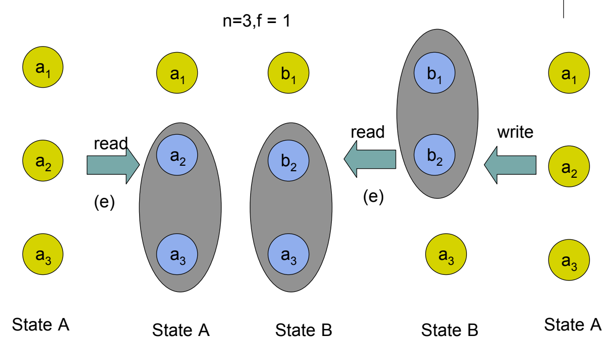
Bounds

Theorem : *In the authenticated asynchronous model with byzantine failures and reliable channels, no live confirmable protocol can satisfy the safe semantics for distributed shared memory using 3f servers*

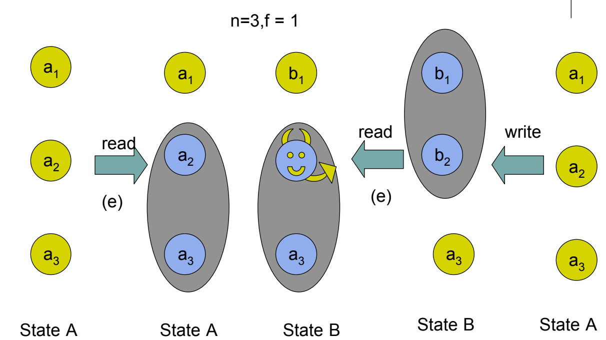
Proof (Sketch) : Such a protocol must violate safety or liveness.

- There must exist an execution in which is an operation influenced by a subset of 2f or fewer servers.

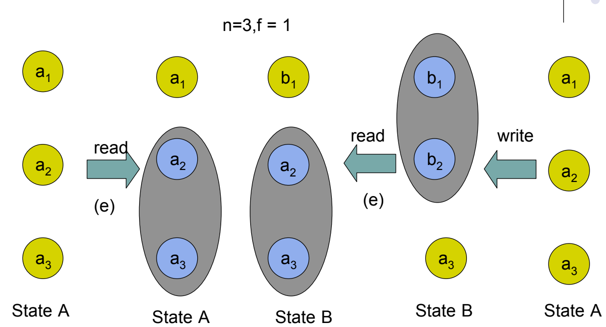
Safety Violation



Safety Violation



Safety Violation



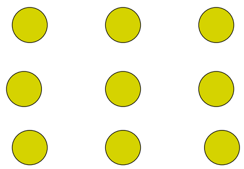
States indistinguishable for e !!

Dynamic Byzantine Quorums

[AMP00]

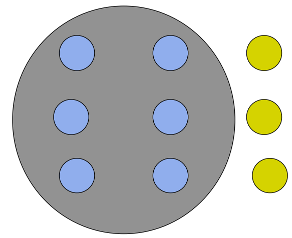
- Designing quorums requires estimating number of faulty servers present at a time
  - Optimistic : Can violate safety
  - Pessimistic : Wastes resources
- Solution
  - Monitor environment to estimate f
  - Adjust resilience threshold dynamically
- Advantages
  - Efficient for small number of failures
  - Read/Write does not block on changing f

Problem to be tackled…



Masking quorum system
n = 9 , f = 1
$\mathcal{Q} = \{Q \subseteq U : |Q| = 6\}$

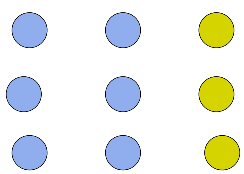
Problem to be tackled…



Masking quorum system
n = 9 , f = 1
$\mathcal{Q} = \{Q \subseteq U : |Q| = 6\}$

Write

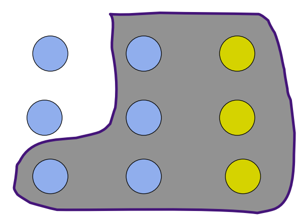
Problem to be tackled…



Reconfiguration ….

Masking quorum system
n = 9 , ~~f = 1~~ f = 2
$\mathcal{Q} = \{Q \subseteq U : |Q| = 7\}$

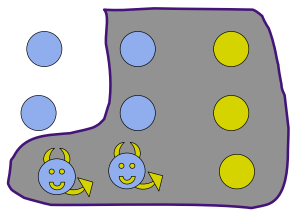
Problem to be tackled…



Masking quorum system
n = 9 , ~~f = 1~~ f = 2
$\mathcal{Q} = \{Q \subseteq U : |Q| = 7\}$

Read

Problem to be tackled…



Masking quorum system
n = 9 , ~~f = 1~~ f = 2
$\mathcal{Q} = \{Q \subseteq U : |Q| = 7\}$

Read

No majority or even a wrong answer!!!

Approach

- Every server stores a threshold variable T giving the present value of f
- Assumption
  - For any operation o, number of failures never exceeds the minimum of :
    - The value written in last write to T
    - Values written to T in writes concurrent with o
  - f lies between f_{min} and f_{max}

New Problem

- What value of threshold to use to read T ?
- Define “announce set”
  - A set of servers whose intersection with all possible quorums is large enough to allow unambiguous determination of T
  - Intersection > $2f_{max}$ ensures this
  - Taking announce set to be n-f_{max} leads to :
    $n \geq 6f_{max} - 2f_{min} + 1$

Operations on T

Client c with current threshold = f

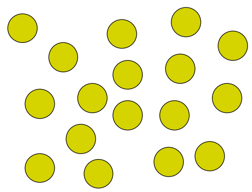
Write(d)

- Ask all servers for their current timestamp t
- Wait for answer from an **announce set**
- Set ts_c > max({t} U any previous ts_c)
- Send (d,ts_c) to all servers
- Wait for acks from an **announce set**

Read()

- Ask all servers for latest value/timestamp pair
- Wait for answer from $|Q_{min}|$ different servers
- Select most recent (v,ts) for which at least $f_{max} + 1$ answers agree (if any)

More problems…..



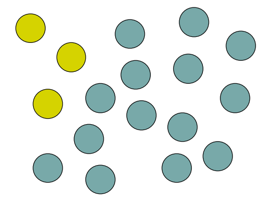
f_{min} = 1 , f_{max} = 3

n = 17, Q_{min} = 10

announce set = 14

Initially T = 1

More problems…..



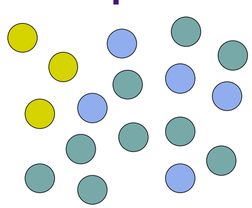
f_{min} = 1 , f_{max} = 3

n = 17, Q_{min} = 10

announce set = 14

Threshold Write : T = 2

More problems…..



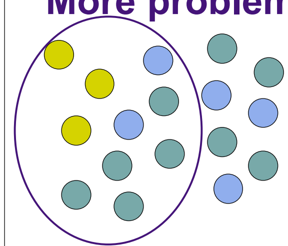
f_{min} = 1 , f_{max} = 3

n = 17, Q_{min} = 10

announce set = 14

While a client performing threshold write to set T = 3 …..

More problems…..



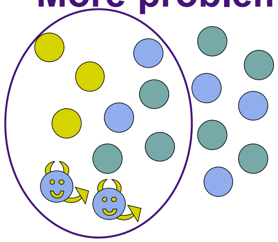
f_{min} = 1 , f_{max} = 3

n = 17, Q_{min} = 10

announce set = 14

another client tries to read T ….

More problems…..



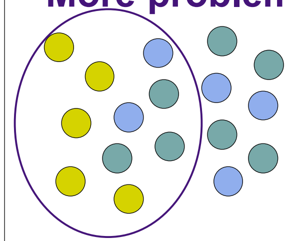
f_{min} = 1 , f_{max} = 3

n = 17, Q_{min} = 10

announce set = 14

another client tries to read T ….

More problems…..



f_{min} = 1 , f_{max} = 3

n = 17, Q_{min} = 10

announce set = 14

The read would return T=1 which is incorrect !!

Solution

- Note that there are still f_{max}+ 1 servers which have a later timestamp
- (v,ts) is countermanded if at least f_{max}+1 servers return a timestamp greater than ts

Read()

- Ask all servers for latest value/timestamp pair
- Wait for answer from $|Q_{min}|$ different servers
- Select most recent (v,ts) for which at least f_{max} + 1 answers agree (if any), **if it is not countermanded**

New Framework for Dynamic Byzantine Storage

[MA04]

- Can adapt to both failure threshold and server count
- Provides confirmable wait-free atomic semantics
- No bounds on number of failures that can be tolerated
- Optimal and fast

The Methodology

- Existing protocols based on *Q-RPC* primitive
- For dynamic quorums, simply replace *Q-RPC* calls by *DQ-RPC*
- Proving correctness requires defining new properties independent of the quorum intersection
- Focus on properties of the data that is retrieved by quorum operations

Transquorum properties

- Timeliness : Any read value must be as recent as the last written value
- Soundness : Any read value must have been written before
- Three sets of Q-RPC-like quorum operations
  - The set of write operations $\mathcal{W}$
  - The set of timely operations $\mathcal{T}$
  - The set of timely and sound operations $\mathcal{R}$

U-Dissemination Protocol

READ

1. Q := Q-RPC("READ")
   //Q is a set of <ts,writer,data>$_{writer}$
2. reply r := phi(Q)
   // returns the largest valid value
3. Q:= Q-RPC("WRITE",r)
4. return r.data

WRITE

1. Q := Q-RPC("GET_TS")
2. ts := max{Q.ts} + 1
3. m := <ts,writer_id,D>$_{writer}$
4. Q := Q-RPC("WRITE",m)

New U-Dissemination Protocol

READ

1. Q := TRANS-Q$_{\mathcal{R}}$("READ")
   //Q is a set of <ts,writer,data>$_{writer}$
2. reply r := phi(Q)
   // returns the largest valid value
3. Q:= TRANS-Q$_{\mathcal{W}}$("WRITE",r)
4. return r.data

WRITE

1. Q := TRANS-Q$_{\mathcal{T}}$("GET_TS")
2. ts := max{Q.ts} + 1
3. m := <ts,writer_id,D>$_{writer}$
4. Q := TRANS-Q$_{\mathcal{W}}$("WRITE",m)

Byzantine Tolerant Erasure-coded Storage

[WGG04]

- If a group of servers coming together to get an answer, then can store parts of information at servers
- Use m-of-n erasure codes
- Requires less bandwidth and storage space than full replication

References

[LAM77] L. Lamport. On Interprocess Communication--Part I: Basic Formalism, Part II: Algorithms *Distributed Computing 1*, 2 (1986), 77-101.

[MR98] D. Malkhi, M. Reiter. Byzantine quorum systems, *Distributed Computing* 11 (4) (1998) 203-213

[MAD02-1] J-P. Martin, L. Alvisi, M. Dahlin. **Minimal Byzantine Storage** *Proceedings of the 16th International Symposium on Distributed Computing (DISC 2002).* Toulouse, France.

[MAD02-2] J.-P. Martin, L. Alvisi, M. Dahlin. Small Byzantine Quorum Systems. *Proceedings of the 2002 International Conference on Dependable Systems & Networks (DSN 2002)*

[AMP00] L. Alvisi, D. Malkhi, E. Pierce, M. Reiter, R. Wright. Dynamic Byzantine Quorum Systems *Proceedings of the 2000 International Conference on Dependable Systems and Networks (IEEE Computer Society)*

[MA04] J.P. Martin and L. Alvisi. Dynamic Byzantine Storage. *Proceedings of the 2004 International Conference on Dependable Systems & Networks (DSN 2004)*

[WGG04] J.J.Wylie, G.R. Goodson, G.R. Ganger, M. Reiter. Efficient Byzantine-tolerant erasure-coded storage. *Under Review*