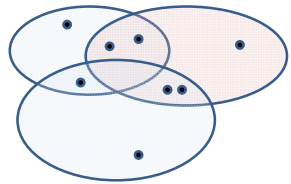


Refined Quorum Systems



Rachid Guerraoui
Marko Vukolić

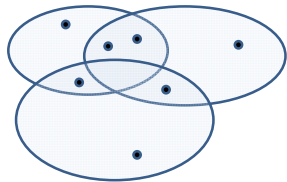
Outline



- Motivation – QS, RQS.
- RQS - Definitions, Properties, B_k and B_1 .
- Definitions and assumptions.
- Consensus - Problem, Algorithm.
- Conclusions.

QS - Quorum systems

- QS = set of subsets that intersect.



- Crash-resilient asynchronous Algorithms.
- Implementations that tolerate asynchrony and optimally resilient to process failures.

Motivation



- Good distributed computing practice: object implementations that tolerate:
 - Contention
 - Periods of asynchrony
 - Large number of failures
- But perform fast otherwise (best-case).

Motivation



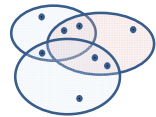
- Byzantine (malicious) failures –
 - Forms of QS: require larger intersections.
 - Useful to reason resilience dimension.
- Complexity – (Simple or Byzantine QS)
 - Not adequate to capture the complexity dimension, specifically the best-case one.

Motivation



	QS $QC1 = QC2 = \emptyset$	RQS $QC1 = \emptyset$	RQS $QC1 \neq \emptyset$
optimal resilience	V	V	V
optimal best-case complexity	X	X	V

RQS - Definitions



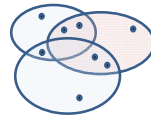
- RQS of some set of elements S is a set of 3 classes of subsets (quorums) of S:
 - QC1 - large intersections with QC2, QC3.
 - QC2 - smaller intersections with QC3.
 - QC3 - correspond to traditional quorums.
- QC1 - are also QC2, which are also QC3.

RQS - Definitions



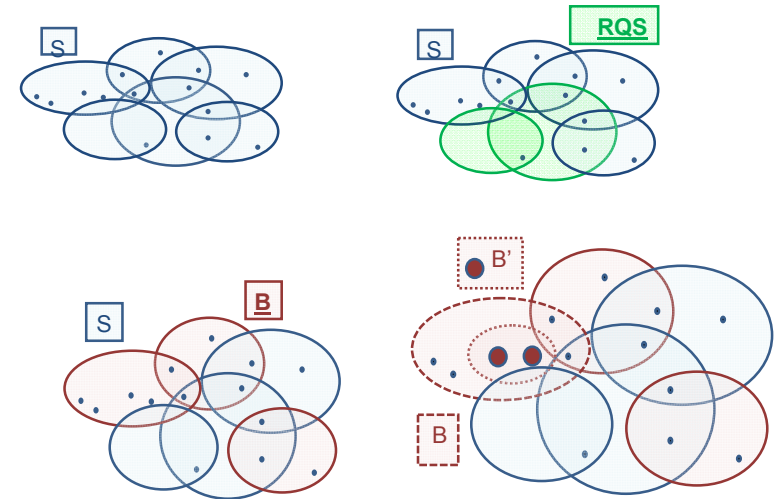
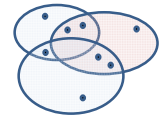
- distributed object implementation under uncontended, synchronous conditions -
 - Expedite operation if a QC1 is available
 - Then degrade gracefully depending on whether a QC2 or QC3 is available.

RQS - Definitions



- S - non-empty set of elements.
 \mathbf{B}, \mathbf{RQS} - any set of subsets of S
- \mathbf{B} - is an adversary structure for S if -
 - $B \cap B' \in \mathbf{B}$
 - $B \cap B' \notin \mathbf{B}$

RQS - Definitions



RQS - Definitions



- \mathbf{RQS} is a refined quorum system for a set S and adversary \mathbf{B} if:

\mathbf{RQS} has two subsets $QC1 \cap QC2 \in \mathbf{RQS}$ such that the following 3 properties hold.

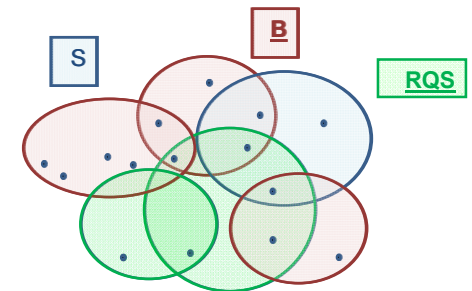
RQS - Property 1



- The intersection of any 2 RQS elements: does not belong to \mathbf{B} .

□ $Q, Q' \in \mathbf{RQS}$:

$Q \cap Q' \notin \mathbf{B}$.



RQS - Property 2



- The intersection of any C1 elements and any RQS element is not a subset of the union of any 2 **B** elements.

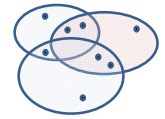
$$\square Q_1, Q'_1 \square QC1,$$

$$\square Q \square \mathbf{RQS},$$

$$\square B_1, B_2 \square \mathbf{B} :$$

$$Q_1 \cap Q'_1 \cap Q \not\subseteq B_1 \cup B_2.$$

RQS - Property 3



- The intersection of any C2 element Q2 and any **RQS** element Q is:

P3a(Q2,Q) –
not a subset of
the union of
any
2 **B** elements

Or

P3b(Q2, Q) – its
intersection with
every C1 element Is
not an element of B

RQS - Property 3a



- The intersection of any C2 element Q2 and any **RQS** element Q is:
not a subset of the union of any 2 **B** elements

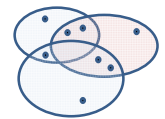
$$\square Q2 \square QC2,$$

$$\square Q \square \mathbf{RQS},$$

$$\square B1, B2 \square \mathbf{B} :$$

$$Q2 \cap Q \not\subseteq B1 \cup B2$$

RQS - Property 3b



- The intersection of C2 element Q2 and any **RQS** element Q with every C1 element Is not an element of B

$$\square Q2 \square QC2,$$

$$\square Q \square \mathbf{RQS},$$

$$\square B1, B2 \square \mathbf{B} :$$

$$(QC1 \neq \emptyset)$$

□

$$(\square Q1 \square QC1 :$$

$$—Q1 \cap Q2 \cap Q—$$

! □ **B**

RQS - Property 3



The intersection of any C2 element Q2 and any RQS element Q is:
 P3a(Q2,Q) - or -
 P3b(Q2, Q)

$$Q_2 \cap Q \subseteq B_1 \cup B_2$$

$$(QC_1 \neq \emptyset)$$

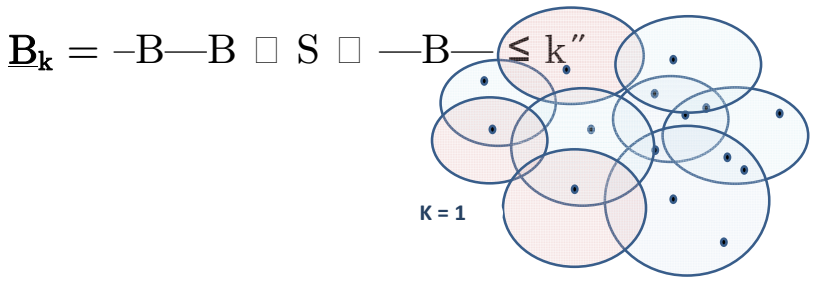
$$(Q_1 \cap QC_1 : Q_1 \cap Q_2 \cap Q \subseteq B)$$

- Q2 □
- QC2,
- Q □ RQS,
- B1, B2 □ B

RQS - B_k



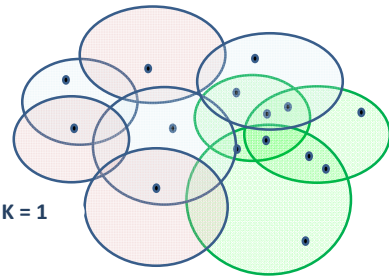
B_k - k-bounded threshold adversary, contains all subsets of S with cardinality at most k.



RQS - B_k



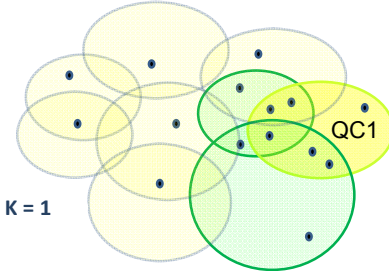
• P1.
 $Q \cap Q' \supseteq B$
 Any 2 quorums intersect in at least k + 1.



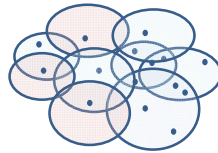
RQS - B_k



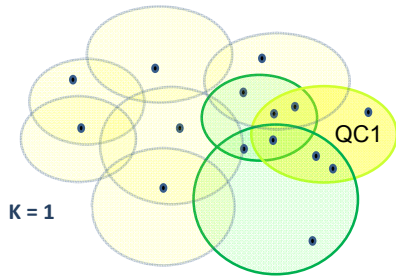
• P2.
 $Q_1 \cap Q'_1 \cap Q \supseteq B_1 \cup B_2$
 The intersection of any two QC1 intersects with any quorum in at least 2k + 1.



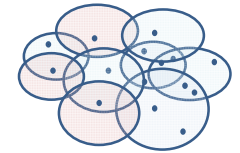
RQS - B_k



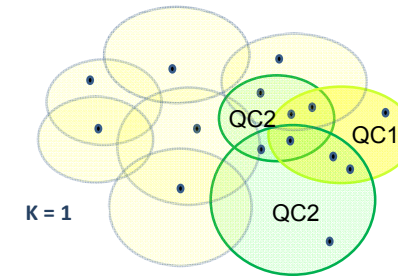
- P3a. $-(Q_2 \cap Q \neq \emptyset \wedge B_1 \sqcap B_2)$
 QC_2, Q intersection at least $2k+1$.



RQS - B_k

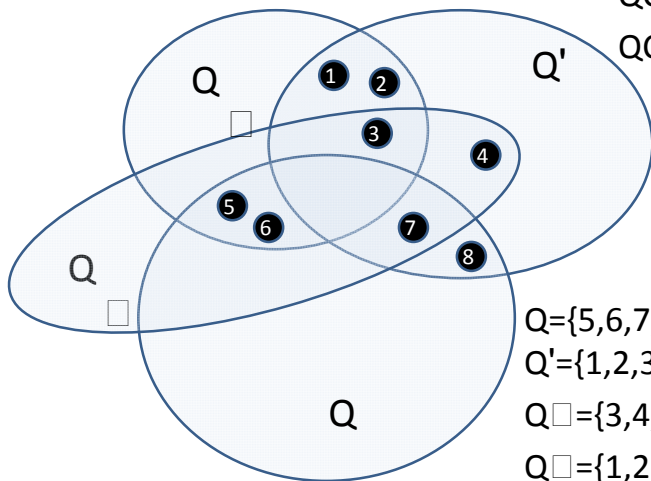


- P3b. $-(QC_1 \neq \emptyset \wedge (\sqcap Q_1 \sqcap QC_1: -Q_1 \cap Q_2 \cap Q - \neq \emptyset \wedge B))$
 this intersection with QC_1 in at least $k + 1$.



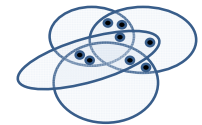
Example 1- B₁

K=1
 RQS={Q, Q', Q_□, Q_□}
 QC_□={Q_□, Q_□}
 QC_□={Q_□}



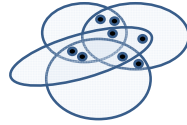
$Q = \{5,6,7,8\} \rightarrow (2,3, 2)$
 $Q' = \{1,2,3,4,7,8\} \rightarrow (3,3,2)$
 $Q_{\square} = \{3,4,5,6,7\} \rightarrow (3,3,3)$
 $Q_{\square} = \{1,2,3,5,6\} \rightarrow (2,3,3)$

Example 1- B₁



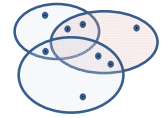
- P1 - Every pair intersects in at least $k + 1$.
- P2 - Q_1 intersects every other in at least $2k + 1$.
- P3 - a. $-Q_2 \cap Q_1 - = 2k + 1$, $-Q_2 \cap Q' - = 2k + 1$
 b. $-Q_2 \cap Q \cap Q_1 - = k + 1$
- RQS={Q, Q', Q_□, Q_□}
- QC_□={Q_□, Q_□}
- QC_□={Q_□}

Example 1- B₁



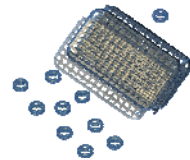
- Cardinality not always indicates class, it is the intersection with others that matters:
 - Q1 contains 5 elements and is a QC1
 - Q contains 6 elements yet is only a QC3.

This paper



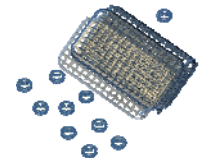
- General adversary structure.
- Subsets of processes can collude.
- Relax the assumption of independent and identically distributed failures.
- Introducing two new optimal Byzantine-resilient atomic object implementations:
 - Atomic storage.
 - Consensus.

Problem I – read/write storage



- challenge – ensure low reads and writes latency in most frequent situations while
 - a) Tolerating –
asynchrony.
failures of many base servers.
wait-freedom: any # of clients accessing.AND
 - b) ensuring strong consistency (ideally atom).

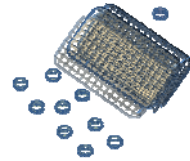
Problem I – read/write storage



- storage algorithm using a RQS:
 - (SWMR) atomic .
 - wait-free.
 - optimal resilience.

- lowest possible R/W latency in best-case
- Alternative (classic centralized storage systems)

Problem I – read/write storage



- Under best-case conditions, the algorithm expedites storage operations (R&W) in:
 - 1 round if a QC1 is accessed,
 - 2 rounds if a QC2 is accessed,
 - 3 rounds otherwise.

Definitions and assumptions

- Correct process p (server/ client):
 - Local step iii Communication step
 - Executes ∞ steps.
- Benign process: Correct or crashes.
- Byzantine process: not benign.

Definitions and assumptions

- Asynchronous - no bound on delays.
- reliable client-server channels.
- No server-server communication.
- global clock- no access to processes.

Definitions and assumptions

- System is synchronous during $[t, t']$ –

if for every 2 correct processes p_1 and p_2 , every message sent by p_1 at time $t_1 \in [t, t']$ is received by a process p_2 at $t_1 + \Delta \in [t, t']$

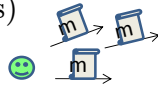
constant $\Delta (> 0)$ known to correct procs.

Definitions and assumptions

- The algorithm is round-based, round of op:

(c- client, s- server, ss-servers)

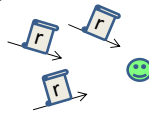
(a) c sends msg to ss



(b) s receives and responds
(not waiting for other msgs)



(c) c receives responses

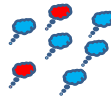


- servers can send only response messages to clients.

Definitions and assumptions

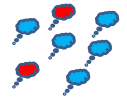
- Latency (complexity)
of an operation op =
number of rounds between
invocation and completion of op.

Consensus - Problem



- Consensus – in distributed computing -
 - Task - group agreement in the presence of faults (any process may crash at any time).
- Consensus abstraction – unlike storage –
 - Byzantine resilient
 - Server-server communication

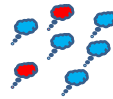
Consensus - Problem



- Consensus abstraction –
- General framework, 3 processes sets:
 - Proposers - propose values. (P)
 - Learners - learn proposed value. (L)
 - Acceptors – (distinct) help learners decide.

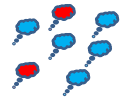


Consensus - Problem



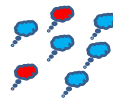
- Consensus abstraction algorithm–
- Tolerate -
 - (1) any number of Byzantine failures of $(L, P, 1)$
 - (2) largest possible number of Acceptor failures
 - (3) arbitrarily long periods of asynchrony

Consensus - Problem



- The algorithm –
 - matches resilience and complexity LB
 - New bound on consensus algorithms that degrade gracefully in best-case executions.
 - minimal conditions for optimally resilient and best-case efficient.

Consensus - Problem



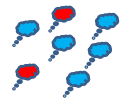
- An algorithm solves consensus if it satisfies the following properties:

(a) Validity.

(b) Agreement.

(c) Termination.

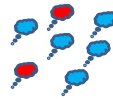
Consensus - Problem



(a) Validity:

If a benign learner learns a value v and all proposers are benign, then some proposer proposes v ;

Consensus - Problem

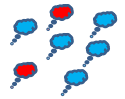


(b) Agreement:

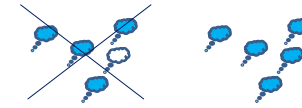


all benign learners learn same value;

Consensus - Problem

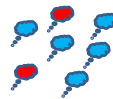


(c) Termination:



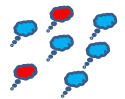
If a correct proposer proposes a value, then eventually, every correct learner learns a value;

Consensus - Algorithm



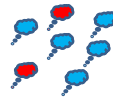
- Consensus safety (Validity and Agreement):
as long as the set of Byzantine acceptors in any execution belongs to B .
- Consensus liveness (Termination) is ensured:
if there is a correct quorum of acceptors $Q_c \cap RQS$
and if the system is eventually synchronous.

Consensus - Algorithm



- We say that an execution ex is best-case if in ex :
 - (1) there is no contention:
 - (a) all proposers are benign and
 - (b) exactly one p (correct) proposes v at time t .
 - (2) and, the system is synchronous during $[t, t+4\Delta]$.

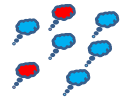
Consensus - Algorithm



- Let \mathbf{P} be any set of subsets of Acceptors.
- Definition - A consensus algorithm is (m, \mathbf{P}) -fast if in every best-case \mathbf{ex} with some $\mathbf{POCA} \subseteq \mathbf{P}$:

All correct Learners learn v in $m + 1$ msg delays.

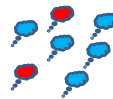
Consensus - Algorithm



- Acceptors = RQS.
- Consider \mathbf{QC}_m for $m \in \{1, 2, 3\}$
- The Consensus algorithm is (m, \mathbf{QC}_m) -fast

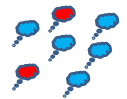
in every best-case \mathbf{ex} with some $\mathbf{QC}_m^{\mathbf{OCA}} \subseteq \mathbf{QC}_m$:
All correct Learners learn v in $m + 1$ msg delays.

Consensus - Algorithm



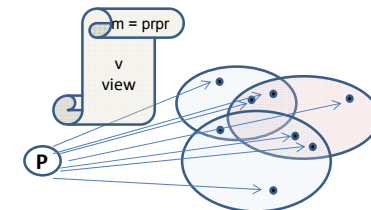
- Under best-case conditions
Algorithm allows a value to be learned in
 - 2 msg-delays, if a \mathbf{QC}_1 is accessed
 - 3 msg-delays, if a \mathbf{QC}_2 is accessed
 - 4 msg-delays, if a \mathbf{QC}_3 is accessed
- learning in 1 message delay is impossible with multiple or potentially Byzantine proposers,
- availability of a \mathbf{QC}_3 is necessary for resilience

Consensus - Algorithm

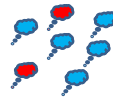


Update phase : (best-case \mathbf{ex})

- (1) the proposer p sends a message $m = \text{prepare}$ containing its proposal value v and view number $\text{view} = \text{initView}$ to all acceptors •

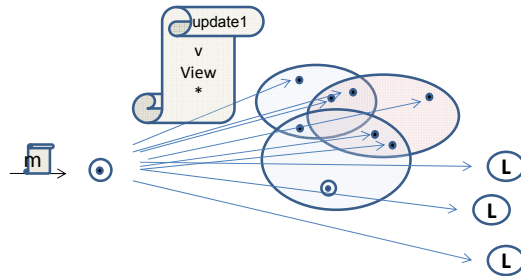


Consensus - Algorithm

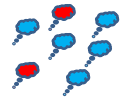


Update phase : (best-case **ex**)

(2) correct acceptor a_j , upon receiving m , performs some local computations, and then echoes $update1$ msg with $v, view$ to all $A_{s, L}$ s.

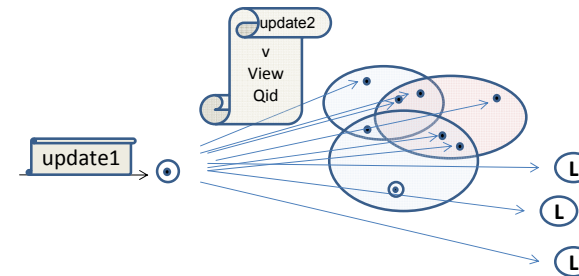


Consensus - Algorithm

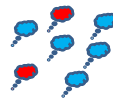


Update phase : (best-case **ex**)

(3) (CA) a_j , upon receiving $update1$ from quorum Q , performs again some local computations and send $update2$ msg with id of Q to all $A_{s, L}$ s.

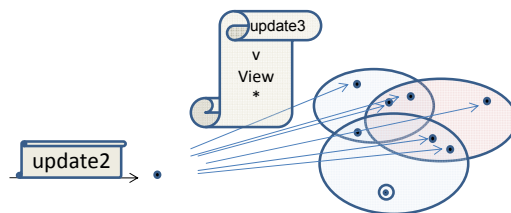


Consensus - Algorithm

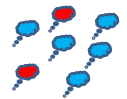


Update phase : (best-case **ex**)

(4) acceptors, similarly to step (3), upon receiving $update2$ from some quorum, 2-update the value v in view and send $update3$ to all acceptors.



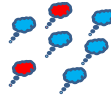
Consensus - Algorithm



Update phase : (best-case **ex**)

(4) This is done only once!
not per every quorum of received update msgs,
as is with 1-updating the value.

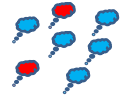
Consensus - Algorithm



Update phase : (best-case **ex**)

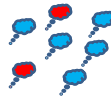
- (1) from (P)_p to all A_s : [prepare(v,view)].
- (2) (CA)_{aj} gets [prepare] :
from(CA)_{aj} to all A_s and L_s : [update1(v,view,*)].
- (3) (CA)_{aj} gets [update1] from quorum Q:
from(CA)_{aj} to all A_s and L_s : [update2(v,view,Q)].
- (4) (A)_a gets [update2] from quorum Q:
from(A)_a to all A_s : [update3(v,view,*)].

Consensus - Algorithm



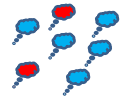
- at every acceptor and learner:
- Upon –
 - get same update1<v, view,*> from Q1 \square QC1
 - get same update2<v, view,Q2> from Q2 \square QC2
 - get same update3<v, view,*> from Q \square RQS
- if x has not yet decided then decide(v)

Consensus - Algorithm



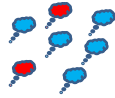
- If some class 1 (resp., class 2, class 3) quorum contains only correct acceptors, then all correct learners and acceptors decide v in initView after two (resp., three, four) message delays.
- Since learners learn a value upon deciding it, we ensure algorithm is (m, QC_m)–fast for all m \square {1, 2, 3}.

Consensus - Algorithm



- The algorithm consists of two modules:
 - (1) Locking module, ensures safety.
 - (2) Election module for liveness.
- In a best-case **ex**, the Election module, responsible for view changes, does not change the view before all correct learners learn v.

Consensus - Algorithm



- However, if the proposer p is Byzantine, or the system is asynchronous, then the Election module might designate a different proposer p_w to be the leader for the new view w .

Conclusions

- Introduces the notion of RQS
- useful to reason optimally resilient and efficient distributed object implementations assuming general adversary structures.
- RQS are necessary and sufficient (minimal) for implementing atomic storage and consensus.
- Best possible latency of an object implementation