How quorum systems work:
A read/write shared register
store at each server
a (v,ts) pair
X
Write(x,d)
• Ask servers in some Q for ts
• Set tsc > max({ts} \cup any previous tsc)
• Update some Q' with (d,tsc)

Quorum Systems
Given a set U of servers, |U| = n:
A quorum system is a set \( Q \subseteq 2^U \)
such that
\[ \forall Q_1, Q_2 \in Q : Q_1 \cap Q_2 \neq \emptyset \]
Each \( Q \) in \( Q \) is a quorum

Do we have a quorum?
How quorum systems work: A read/write shared register

Write(x,d)
- Ask servers in some Q for (v,ts)
- Set ts_c > max((v,ts) ∪ any previous ts_c)
- Update some Q' with (d,ts_c)

Read(x)
- Ask servers in some Q for (v,ts)
- Select most recent (v,ts)

store at each server a (v,ts) pair

What semantics?

Safe:
A read not concurrent with any write returns the most recently written value

Regular:
Safe + a read that overlaps with a write obtains either the old or the new value

Atomic:
Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping

What semantics?

Safe:
A read not concurrent with any write returns the most recently written value

Regular:
Safe + a read that overlaps with a write obtains either the old or the new value

Atomic:
Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping
What semantics?

**Safe:**
A read not concurrent with any write returns the most recently written value

**Regular:**
Safe + a read that overlaps with a write obtains either the old or the new value

**Atomic:**
Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping

---

What semantics?

**Safe:**
A read not concurrent with any write returns the most recently written value

**Regular:**
Safe + a read that overlaps with a write obtains either the old or the new value

**Atomic:**
Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping

---

System Model

- **Universe U** of servers, |U| = n
- **Byzantine faulty servers**
  - Modeled as a non-empty fail-prone system $B \subseteq 2^U$
  - No $B \in B$ is contained in another
  - Some $B \in B$ contains all faulty servers
- **Clients are correct** (can be weakened)
- **Point-to-point authenticated and reliable channels**

A correct process $q$ receives a message from another correct process $p$ if and only if $p$ sent it
**Masking Quorum System**

[Malkhi and Reiter, 1998]

A quorum system \( \mathcal{Q} \) is a *masking quorum system* for a fail-prone system \( B \) if the following properties hold:

- **M-Consistency**
  \[ \forall Q_1, Q_2 \in \mathcal{Q} \forall B_1, B_2 \in B : (Q_1 \cap Q_2) \setminus B_1 \subseteq B_2 \]

- **M-Availability**
  \[ \forall B \in B \exists Q \in \mathcal{Q} : B \cap Q = \emptyset \]

**Dissemination Quorum System**

A masking quorum system for *self-verifying data*—client can detect modification by faulty server

- **D-Consistency**
  \[ \forall Q_1, Q_2 \in \mathcal{Q} \forall B \in B : (Q_1 \cap Q_2) \subseteq B \]

- **D-Availability**
  \[ \forall B \in B \exists Q \in \mathcal{Q} : B \cap Q = \emptyset \]

---

**f-threshold Masking Quorum Systems**

A quorum system is a masking quorum system if the following properties hold:

- **M-Consistency**
  \[ \forall Q_1, Q_2 \in \mathcal{Q} \forall B_1, B_2 \in B : (Q_1 \cap Q_2) \setminus B_1 \subseteq B_2 \]

- **M-Availability**
  \[ \forall B \in B \exists Q \in \mathcal{Q} : B \cap Q = \emptyset \]

**A safe read/write protocol**

Client \( c \) executes:

- **Write\( d \)**
  - Ask all servers for their current timestamp \( t \)
  - Wait for answer from |\( Q \)| different servers
  - Set \( t_{2c} > \text{max} \{ t \} \cup \) any previous \( t_{2c} \)
  - Send \( (d, t_{2c}) \) to all servers
  - Wait for \( |Q| \) acknowledgments

- **Read\( \)**
  - Ask all servers for latest value/timestamp pair
  - Wait for answer from |\( Q \)| different servers
  - Select most recent \( (v, t_s) \)
  - Wait for most recent answers to agree (if any)

}\[ B = \{ B \subseteq U : |B| = f \} \]
Reconfigurable quorums

Design a Byzantine data service that
- monitors environment
  - uses statistical techniques to estimate number of faulty servers
- adjusts its tolerance capabilities accordingly:
  - fault-tolerance threshold changes within \([f_{\text{min}} - f_{\text{max}}]\) range
    - very efficient when no or few failures
    - can cope with new faults as they occur
- does not require read/write operations to block
- provides strong semantics guarantees

Managing the threshold

- Keep threshold value in a variable \(T\)
- Refine assumption on failures:
  - For any operation \(o\), number of failures never exceeds \(f\), the minimum of:
    - value of \(T\) before \(o\)
    - any value written to \(T\) concurrently with \(o\).
- Which threshold value should we use to read \(T\)?
  - Update \(T\) by writing to an announce set
  - A set of servers whose intersection with every quorum (as defined by \(f\) in \([f_{\text{min}} - f_{\text{max}}]\)) contains sufficiently many correct servers to allow client to determine \(T\)'s value unambiguously.

The announce set

- Intersects all quorums in at least \(2f_{\text{max}} + 1\) servers
- Conservative announce set size: \(n - f_{\text{max}}\)
- Hence: \(\frac{n + 2f_{\text{min}} + 1}{2} + (n - f_{\text{max}}) - n \geq 2f_{\text{max}} + 1\)
  - \(n \geq 6f_{\text{max}} - 2f_{\text{min}} + 1\)

Updating \(T\)

- Client \(c\) (with current threshold \(f\)) executes:
  - Write(\(d\))
    - Ask all servers for their current timestamp \(t\)
    - Wait for answer from \(|Q|\) different servers
      - Set \(t_{\text{SC}} > \max\{t_i\} \cup \text{any previous} t_{\text{SC}}\)
    - Send \((d,t_{\text{SC}})\) to all servers
    - Wait for \(|Q|\) acknowledgements
  - Read(\(l\))
    - Ask all servers for latest value/timestamp pair
    - Wait for answer from \(|Q|\) different servers
    - Select most recent \((v,ts)\) for which at least \(f + 1\) answers agree (if any)
Updating T

Client c (with current threshold f) executes:

Write(d)
→ Ask all servers for their current timestamp t
← Wait for answer from |Q| different servers
Set ts_c > max({t} ∪ any previous ts_c)
→ Send (d,ts_c) to all servers
← Wait for n acknowledgments from an announce set

Read()
→ Ask all servers for latest value/timestamp pair
← Wait for answer from |Q| different servers
Select most recent (v,ts) for which at least f + 1 answers agree (if any)

A problem

Initially, T = 1

Threshold write: T = 2
Countermanding

(v, ts) is countermanded if at least $f_{\text{max}}+1$ servers return a timestamp greater than ts

Write\( (v, ts) \):

$\rightarrow$ Ask all servers for their current timestamp \( t \)

$\leftarrow$ Wait for answer from an announce set

Set \( t_{\text{ts}} \geq \max \{ (v, t) \} \cup \text{any previous } t_{\text{ts}} \)

$\rightarrow$ Send \((d, t_{\text{ts}})\) to all servers

$\leftarrow$ Wait for acknowledgements from an announce set

Read\( (v, ts) \):

$\rightarrow$ Ask all servers for latest value/timestamp pair

$\leftarrow$ Wait for answer from \( 1Q_{\text{min}} \) different servers

Select most recent \((v, ts)\) for which at least \( f_{\text{max}}+1 \) answers agree (if any)

\((\text{not countermanded})\)

A problem

While a client is performing a threshold write to set \( T = 3 \)...

A problem

...another client tries to read \( T \)

Minimizing quorum size

Who cares? Machines are cheap...

But achieving independent failures is expensive!

- Independently failing hardware
- Independently failing software

- Independent implementations of server
- Independent implementation of underlying OS
- Independent versions to maintain
### Tradeoffs

<table>
<thead>
<tr>
<th>best known n</th>
<th>confirmable</th>
<th>non-confirmable</th>
</tr>
</thead>
<tbody>
<tr>
<td>self-verifying</td>
<td>3f+1</td>
<td>2f+1</td>
</tr>
<tr>
<td>generic</td>
<td>4f+1</td>
<td>3f+1</td>
</tr>
</tbody>
</table>

Lower bound: never two rows again!

### The intuition

Trade replication in space for replication in time

Traditional: 4f+1 servers

Now: 3f+1 servers
The intuition

Trade replication in space for replication in time

Traditional: 4f+1 servers

Now: 3f+1 servers

Both cases: wait until 4th server receives write

The protocol

Client c executes:

Write(d)

→ Ask all servers for their current timestamp t
← Wait for answer from |Q_w| different servers
Set ts_C = max{t} ∪ any previous ts_C
→ Send (d, ts) to all servers
← Wait for |Q_w| acknowledgments

Read()

→ send read-start to server set Q_r
   repeat
      ← receive a reply (D, ts) from s in Q_r
      set answer[s, ts] := D
      until some A in answer[|Q_r|] is vouched for by |Q_w| servers
→ send read-stop to Q_r
   return A

The Slim-Fast version

1. When c gets first message from a server, it builds
   \( T = \{ \text{largest } f+1 \text{ timestamps from distinct servers} \} \)

2. \((D, ts)\) from answer[s][] is discarded unless either
   a) \( ts \in T \) or
   b) \( ts \) is the latest timestamp received from s

The Goodies

Theorem

The protocol guarantees atomic semantics
Proof: Safety

Lemma 1: If it is live, it is atomic

a) After write of ts1, no read returns earlier ts
   • Suppose write for ts1 has completed
   • \( \lceil \frac{n+f+1}{2} \rceil \) servers acked the write
   • At least \( \lceil \frac{n-f+1}{2} \rceil \) are correct
   • Remaining \( \lceil \frac{n-f-1}{2} \rceil \) servers < |Qw|
   • c reads ts1 \( \rightarrow \lceil \frac{n+f+1}{2} \rceil \) servers say ts1
   • At least \( \lceil \frac{n-f+1}{2} \rceil \) are correct
   • Remaining \( \lceil \frac{n+f-1}{2} \rceil \) servers < |Qw|
   • Any read that starts after ts1 returns ts ≥ ts1

b) After c reads ts1, no later read returns earlier ts

Proof: Liveness

Lemma 2: Every operation eventually terminates

WRITE: trivial, because only waits for |Qw| < n − f
READ:
Consider T after c gets first message from last server.
Let \( t_{\text{max}} \) be the largest timestamp from a correct server in T.
A client never removes \( t_{\text{max}} \) from its answers[s](), for a correct s
Eventually, all correct servers see a write with ts = \( t_{\text{max}} \) and echo client
Since |Qw| = \( \lceil \frac{n+3f+1}{2} \rceil \), |Qw| < |Qr| − f and the read terminates