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

Write\((x,d)\)

- Ask servers in some Q for ts
- Set ts\(_C\) > max\(\{ts\} \cup \text{any previous ts}_C\) 
- Update some Q’ with \((d,ts\_C)\)
How quorum systems work: A read/write shared register

Write(x,d)
- Ask servers in some Q for ts
- Set tsC > max({ts}∪ any previous tsC)
- Update some Q’ with (d,tsC)

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

store at each server a (v,ts) pair
How quorum systems work:
A read/write shared register

Write($x,d$)
- Ask servers in some $Q$ for $ts$
- Set $ts_C > \max\{{ts}\cup\text{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
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 $\mathcal{B}$ if the following properties hold:

**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**

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

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

D-Consistency
\[ \forall Q_1, Q_2 \in Q \ \forall B \in B : (Q_1 \cap Q_2) \not\subseteq B \]

D-Availability
\[ \forall B \in B \ \exists Q \in Q : B \cap Q = \emptyset \]
**f-threshold Masking Quorum Systems**

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

- **M-Consistency**
  \[ \forall Q_1, Q_2 \in Q : |Q_1 \cap Q_2| \geq 2f + 1 \]

- **D-Consistency**
  \[ \forall Q_1, Q_2 \in Q : |Q_1 \cap Q_2| \geq f + 1 \]

- **M-Availability**
  \[ |Q| \leq n - f \]

- **D-Availability**
  \[ |Q| \leq n - f \]

- \[ Q = \left\{ Q \subseteq U : |Q| = \left\lceil \frac{n + 2f + 1}{2} \right\rceil \right\} \]

  \[ n \geq 4f + 1 \]

- \[ Q = \left\{ Q \subseteq U : |Q| = \left\lceil \frac{n + f + 1}{2} \right\rceil \right\} \]

  \[ n \geq 3f + 1 \]
A safe read/write protocol

Client $c$ executes:

Write($d$)
$\rightarrow$ Ask all servers for their current timestamp $t$
$\leftarrow$ Wait for answer from $|Q|$ different servers
    Set $ts_C > \max(\{t\} \cup \text{any previous } ts_C)$
$\rightarrow$ Send $(d,ts_C)$ to all servers
$\leftarrow$ Wait for $|Q|$ acknowledgments

Read()
$\rightarrow$ Ask all servers for latest value/timestamp pair
$\leftarrow$ Wait for answer from $|Q|$ different servers
    Select most recent $(v,ts)$ for which at least $f+1$ answers agree (if any)
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 \([f_{\text{min}} \ldots f_{\text{max}}]\) 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:
    - a) value of $T$ before $o$
    - b) 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 $[f_{\min}...f_{\max}]$) contains sufficiently many correct servers to allow client to determine $T$’s value unambiguously.
The announce set

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

$$n \geq 6f_{max} - 2f_{min} + 1$$
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 ts_c > max({t} ∪ any previous ts_c)
→ Send (d,ts_c) to all servers
← Wait for |Q| acknowledgements

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)
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 an announce set
Set $ts_C > \max\{\{t\} \cup \text{any previous } ts_C\}$
→ Send ($d, ts_C$) to all servers
← Wait for $f+1$ 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)
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 an announce set
  Set tsc > max({t} ∪ any previous tsc)
→ Send (d,tsc) to all servers
← Wait for acknowledgments from an announce set

Read()
→ Ask all servers for latest value/timestamp pair
← Wait for answer from |Qmin| different servers
  Select most recent (v,ts) for which at least f_{max} + 1 answers agree (if any)
A problem

Initially, $T = 1$

\[
\begin{align*}
  f_{\text{min}} &= 1 & f_{\text{max}} &= 3 \\
  n &= 17 & Q_{\text{min}} &= 10 \\
  \text{announce set} &= 14
\end{align*}
\]
A problem

Threshold write: $T = 2$

\[ f_{\text{min}} = 1 \quad f_{\text{max}} = 3 \]

\[ n = 17 \quad Q_{\text{min}} = 10 \]

announce set = 14
A problem

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

\[
\begin{align*}
    f_{\text{min}} &= 1 & f_{\text{max}} &= 3 \\
    n &= 17 & Q_{\text{min}} &= 10 \\
    \text{announce set} &= 14
\end{align*}
\]
A problem

...another client tries to read T

\[ f_{\text{min}} = 1 \quad f_{\text{max}} = 3 \]
\[ n = 17 \quad Q_{\text{min}} = 10 \]

announce set = 14
Countermanding

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

**Write(f)**
- Ask all servers for their current timestamp $t$
- Wait for answer from an announce set
  - Set $ts_c > \max\{\{t\} \cup \text{any previous } ts_c\}$
- Send $(d, ts_c)$ to all servers
- Wait for acknowledgements from an announce set

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

*not countermanded*
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
A simple observation

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 $ts_C > \max(\{t\} \cup \text{any previous } ts_C)$
- Send $(d, ts_C)$ 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 $(v, ts)$ for which at least $f + 1$ answers agree (if any)

(Asynchronous)
Authenticated
Reliable channels

A correct process $q$ receives a message from another correct process $p$ if and only if $p$ sent it
A quorum system $Q$ is an a-masking quorum system for a fail-prone system $B$ if the following properties hold for $Q_r$ and $Q_w$:

**AM-Consistency**

$$\forall Q_r \in Q_r \ \forall Q_w \in Q_w \ \forall B_1, B_2 \in B$$

$$\left( Q_r \cap Q_w \right) \setminus B_1 \not\subseteq B_2$$

**AM-Availability**

$$\forall B \in B \ \exists Q_r \in Q_r : B \cap Q_r = \emptyset$$
# 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>
# Tradeoffs

<table>
<thead>
<tr>
<th>best known n</th>
<th>confirmable</th>
<th>non-confirmable</th>
</tr>
</thead>
<tbody>
<tr>
<td>self-verifying and generic</td>
<td>3f+1</td>
<td>2f+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
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($d$)**

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

$\leftarrow$ Wait for answer from $|Q_w|$ different servers

Set $ts_c > \max(\{t\} \cup \text{any previous } ts_c)$

$\rightarrow$ Send $(d, ts)$ to all servers

$\leftarrow$ Wait for $|Q_w|$ acknowledgments

**Read()**

$\rightarrow$ send READ-START to server set $Q_r$

repeat

$\leftarrow$ receive a reply $(D, ts)$ from $s$ in $Q_r$

set $\text{answer}[s, ts] := D$

until some $A$ in $\text{answer}[ ]$ is vouched for by $|Q_w|$ servers

$\rightarrow$ send READ-STOP to $Q_r$

return $A$
The Slim-Fast version

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

2. \((D,ts)\) from \text{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 $t_{s1}$, no read returns earlier $ts$
   - Suppose write for $t_{s1}$ has completed
   - $\left\lceil \frac{n+f+1}{2} \right\rceil$ servers acked the write
   - At least $\left\lceil \frac{n-f+1}{2} \right\rceil$ are correct
   - Remaining $\left\lceil \frac{n+f-1}{2} \right\rceil$ servers $< |Q_w|$

b) After $c$ reads $t_{s1}$, no later read returns earlier $ts$
   - $c$ reads $t_{s1} \rightarrow \left\lceil \frac{n+f+1}{2} \right\rceil$ servers say $t_{s1}$
   - At least $\left\lceil \frac{n-f+1}{2} \right\rceil$ are correct
   - Remaining $\left\lceil \frac{n+f-1}{2} \right\rceil$ servers $< |Q_w|$.
   - Any read that starts after $t_{s1}$ returns $ts \geq t_{s1}$
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 $t_{\text{max}}$ 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 $|Q_r| = \left\lceil \frac{n+3f+1}{2} \right\rceil$, $|Q_w| \leq |Q_r| - f$ and the read terminates