Day 13: Snapshots & RPCs
Today

1. Snapshots
2. RPCs
Snapshots

Chandy Lamport Snapshots

for every msg in the snapshot, the snapshot includes the send and recv events

Consistent snapshots

for every msg in the snapshot, the snapshot includes the send event and maybe the recv
Snapshot techniques

Synchronized

Consistency is not guaranteed

Periodic

There is at least one guaranteed snapshot (at the beginning)

Chandy Lamport

Every snapshot is consistent
Chandy Lamport Assumption

no reordering of msgs

if violated

then consistency is compromised

if violated

then the protocol never ends

if violated

servers never crash

msgs never get lost

I to ensure that pkts before marker are in snapshot

I to avoid loss of marker
no reordering

reordering

With reordering, the marker is delivered after $x$ & not before $x$ hence a huge problem.

inconsistency
I

\[ [1,0] \quad [2,0] \]

\[ \alpha \quad \beta \]

1

\[ [0,1] \quad [2,2] \]

an event \( x \) comes after \( y \) iff

\[ VC_x > VC_y \]

for \( i \in \mathbb{N} \)

Thus \( VC_x[i] \geq VC_y[i] \)

if we compare \( a \) with \( c \) if we compare \( c \) with \( a \)

\( VC_a \) is not \( > VC_c \)

because \( VC_a[2] \leq VC_c[2] \)

\( VC_c \) is not \( VC_a \)

because \( VC_c[1] \leq VC_a[1] \)

Here \( a \) and \( c \) happen in parallel.
• $\pm a \pm b$:

- $a = [1,0]$ and $b = [2,0]$
- $c = [0,1]$ and $d = [2,2]$

1. Parallel so can be in any order.

2. Both are valid.

\[ \begin{align*}
  c & \equiv d \\
  a & \equiv b
\end{align*} \]
• The snapshot at [1,0] is inconsistent because S2 knows about more events than S1 knows.

Also, if we try to order the events we will discover that we are missing events. This is not a reachable state under normal circumstances.
Remote Procedure Call (RPC)

- Call \( f(c) \)
- Return value of \( f(c) \)
You can't copy pointers because they point to memory and memory in different machines will have different contents.
<table>
<thead>
<tr>
<th></th>
<th>at least once</th>
<th>at most once</th>
<th>exactly once</th>
</tr>
</thead>
<tbody>
<tr>
<td>(or more)</td>
<td>(0 or 1)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**RPC (semantics)**

Impossible because bugs may prevent you from ever running a function
failure

(a)

\[
\begin{align*}
A & \rightarrow B \\
\text{server fails}
\end{align*}
\]

(b)

\[
\begin{align*}
A \otimes B & \\
\text{return value} \\
n/w \text{ fails (drops)} \\
\text{after server process}
\end{align*}
\]

(c)

\[
\begin{align*}
A \oplus B & \\
n/w \text{ drops msg before it gets to the server}
\end{align*}
\]

from the point of view of server A all failures are the same: server A never gets a response So server A retries

if the failure is the network dropping the response packet then a retry will lead to server B re-executing the function. We may need additional logic to prevent duplicate executions
Techniques to deal with failures

Request retry: set a timeout & if a reply isn't received within that time then retry the request.

Suppress duplicates: maintain a "log" of processed requests. Check every request against the log; if you notice a duplicate then drop & ignore this request.

Response replay: maintain a "log" of processed requests & their responses. Check every request against the log; if you notice a duplicate then reply with the cached response.

Duplicate suppression: identify duplicates & ignore them. Both prevent the system from doing duplicate work.
 retry logic goes here

client

A

Make() 

return value

B

Server

Suppression & replay logic goes here
### RPC (Semantics)

<table>
<thead>
<tr>
<th></th>
<th>at-least-once</th>
<th>at-most-once</th>
<th>exactly-once</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1 or more)</td>
<td>(0 or 1)</td>
<td>1</td>
</tr>
<tr>
<td>Retry</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Suppress duplicates</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replay (server side)</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Response</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Impossible because bugs may prevent you from ever running a function*
Today

Snapshots
Ordering VCs
RPCs
Assumptions

① servers never crash (if they do the system stops)

② external entity starts the snapshot process

③ msg are deliver in order (FIFO order)

④ also msg don't get lost

out of order introduces inconsistency

> If marker is lost then protocol never ends