EECS 591
Distributed Systems

Manos Kapritsos
Fall 2021
After collecting a P-Certificate, replica $k$ sends $\langle\text{COMMIT}, v, n, d, k\rangle_{\sigma_k}$ to all replicas.
Commit Certificate

• C-Certificates ensure consistent order of requests across views
  • Cannot miss a P-Certificate during view change

• A replica has a C-Certificate \((m,v,n)\) iff:
  • it had a P-Certificate \((m,v,n)\)
  • its log contains \(2f + 1\) matching COMMIT messages from distinct replicas (including itself)

• A replica executes a request when:
  • it gets a C-Certificate for it
  • it has executed all requests with smaller sequence numbers
After executing a request, replica $k$ replies to the client with $\langle \text{REPLY}, v, t, c, k, r \rangle_{\sigma_k}$.

- **Primary**
- **Replica 1**
- **Replica 2**
- **Replica 3**

**Phase Breakdown**:
- Pre-prepare phase
- Prepare phase
- Commit phase
- Reply phase

How many matching requests must the client wait for?
To arms, replicas!!

- A disgruntled replica mutinies:
  - Stops accepting messages (except for \(\text{VIEW-CHANGE}\) and \(\text{NEW-VIEW}\) messages)
  - Sends \(\langle \text{VIEW-CHANGE}, v+1, P_{\sigma_k} \rangle\)
  - \(P\) contains all P-Certificates known to replica \(k\)

- A replica joins mutiny after seeing \(f + 1\) distinct \(\text{VIEW-CHANGE}\) messages

- Mutiny succeeds if the new primary collects a new-view certificate \(V\), indicating support from \(2f + 1\) distinct replicas (including itself)
ON TO VIEW v+1: THE NEW PRIMARY

- The “primary-elect” $p'$ (replica $v+1 \mod N$) extracts from the new-view certificate $\mathcal{V}$:
  - the highest sequence number $h$ of any message for which $\mathcal{V}$ contains a P-Certificate $\mathcal{P}$

$$\begin{array}{ccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & \ldots \\
\mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \mathcal{P} & \ldots \\
\end{array}$$

- two sets $\mathcal{O}$ and $\mathcal{N}$:
  - if there is a P-certificate for $n, m$ in $\mathcal{V}$, where $n \leq h$
    - add $<\text{PRE-PREPARE}, v+1, n, m>_{\sigma_p}$ to $\mathcal{O}$
  - otherwise, if $n \leq h$ but there is no P-Certificate
    - add $<\text{PRE-PREPARE}, v+1, n, \text{null}>_{\sigma_p}$ to $\mathcal{N}$

- $p'$ sends $<\text{NEW-VIEW}, v+1, \mathcal{V}, \mathcal{O}, \mathcal{N}>_{\sigma_p}$ to all replicas
ON TO VIEW v+1: THE REPLICA

- A replica accepts a NEW-VIEW message for v+1 if
  - it is signed properly
  - it contains in V valid VIEW-CHANGE messages for v+1
  - it can verify locally that O is correct (repeating the primary’s computation)

- Adds all entries in O to its log (as did p’)

- Sends a PREPARE to all replicas for each message in O

- Adds all PREPARE messages to its log and enters new view
BFT: A PERSPECTIVE

On the other hand:
Google has used BFT in its datacenters and so do many blockchain approaches
ADMINISTRIVIA

Midterm
- Next Wednesday 10/27, 12-1:20pm, during class
  - You can use any material listed on the course website

No class on Monday 10/25
- Conflict with SOSP

Research part
- Starts on Monday 11/1
  - You should read both papers and you can review either one
Eve: replicating multithreaded servers

Kapritsos, Wang, Quema, Clement, Alvisi, Dahlin
THE ACHILLES’ HEEL OF REPLICATION

birth of most dependability techniques


Challenge: scale to multithreaded execution
How do we build dependable multithreaded services?

Answer: State Machine Replication
**State Machine Replication**

**Ingredients:** a server

1. Make server deterministic (state machine)
2. Replicate server
3. Provide all replicas with the same input

**Guarantee:** correct replicas will produce the same output
SMR IMPLEMENTATION

Agree
How do we build dependable **multithreaded** services?
How do we build dependable multithreaded services?
Eve (OSDI ’12)

Scaling replication to multithreaded execution
SMR requires replica convergence

Agree

Execute

Agree-Execute enforces sequential execution
**EXECUTE-VERIFY**

First execute...
(multithreaded and without agreeing on the order)

...then verify
(that replicas agree on the outcome)
ON CONVERGENCE

Server
Commit

Verify

match?

Server
Commit
YES

Server
Commit
YES

Server
Commit
YES
ON DIVERGENCE

Repair: rollback and re-execute sequentially
Eve’s logic at a glance

```
if (converged)
  commit
else
  repair divergence
```

1. Make divergence uncommon
2. Detect divergence efficiently
3. Repair divergence efficiently
if (converged)
commit
else
repair divergence

Idea: identify commutative requests

**Mixer:** group together commutative requests
- Execute requests within a group in parallel

Mixer is a hint, not an oracle
## Example: TPC-W Mixer

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Read tables</th>
<th>Write tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBestSellers</td>
<td>item, author, order_line</td>
<td></td>
</tr>
<tr>
<td>doCart</td>
<td>item</td>
<td>shopping_cart_line, shopping_cart</td>
</tr>
<tr>
<td>doBuyConfirm</td>
<td>customer, address</td>
<td>order_line, item, cc_xacts, shopping_cart_line</td>
</tr>
</tbody>
</table>

3 frequent transactions of the TPC-W browsing workload
Efficient divergence detection

Need to compare application states & responses frequently

if (converged)
  commit
else
  repair divergence

Application state

Merkle tree
Efficient divergence repair

Need to rollback application states after every divergence

```
if (converged)
  commit
else
  repair divergence
```
if (converged)
    commit
else
    repair divergence

1. Make divergence uncommon
2. Detect divergence efficiently
3. Repair divergence efficiently
Masking concurrency bugs

Server

Verify

token

token

token
Execute-Verify: An architectural change

- Synchronous
- Asynchronous

Arbitrary failures

Crash failures
**Configurations**

**Asynchronous BFT**
- Execution
- Verification

Tolerates 1 arbitrary fault

**Synchronous primary-backup**
- Primary
- Backup

Tolerates 1 omission fault
EVALUATION

What is the performance benefit of Eve compared to traditional SMR systems?
Application: H2 Database Engine
Workload: TPC-W (browsing)