EECS 591
DISTRIBUTED SYSTEMS

Manos Kapritsos
Fall 2020
After executing a request, replica \( k \) replies to the client with \(<\text{REPLY}, v, t, c, k, r>\) \( \sigma_k \).
TO ARMS, REPLICAS!!

• A disgruntled replica mutinies:
  • Stops accepting messages (except for VIEW-CHANGE and NEW-VIEW messages)
  • sends $<\text{VIEW-CHANGE, } v + 1, \mathcal{P}>_{\sigma_k}$
  • $\mathcal{P}$ contains all P-Certificates known to replica $k$

• A replica joins mutiny after seeing $f + 1$ distinct VIEW-CHANGE messages

• Mutiny succeeds if the new primary collects a new-view certificate $\mathcal{V}$, 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 $V$:
  - the highest sequence number $h$ of any message for which $V$ contains a P-Certificate:

```
  P  P  P  P  P  P
  0  1  2  3  4  5
```
  
  $h = 12$

- two sets $\mathcal{O}$ and $\mathcal{N}$:
  - if there is a P-certificate for $n, m$ in $V$, 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 $<\text{NEW-VIEW}, v+1, V, \mathcal{O}, \mathcal{N}>_{\sigma_{p'}}$ 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

Problem set #2
  • We will send you your graded papers by tomorrow

Midterm
  • Wednesday 10/21, 3-5pm, during class
    • You can use any material listed on the course website
    • I will post a sample exam today

Research part
  • The review website will be up soon
    • Look for an email with your credentials
EVE: replicating multithreaded servers

Kapritsos, Wang, Quema, Clement, Alvisi, Dahlin
The Achilles’ heel of replication

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

Answer:
State Machine Replication
STATE MACHINE REPLICAATION

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

Server

Server

Server
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 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

Server → Commit

Server → Commit

Verify

YES

YES

YES

match?
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
Making divergence uncommon

Idea: identify commutative requests

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

Mixer is a hint, not an oracle

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

Merkle tree

Application state
Efficient divergence repair

Need to rollback application states after every divergence

if (converged)
commit
else
repair divergence

Application state

Rollback

Copy-on-write
if (converged)
commit
else
repair divergence

1. Make divergence uncommon
   Mixer

2. Detect divergence efficiently
   Merkle tree

3. Repair divergence efficiently
   Copy-on-Write
Masking concurrency bugs

Verify

Server

Server

Server
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)