uKharon
A Membership Service for Microsecond Applications

Rachid Guerraoui, Antoine Murat, Javier Picorel,
Athanasios Xygkis, Huabing Yan, Pengfei Zuo

EPFL

HUAWEI
The data center is a zoo!

Today more than ever, failures are first class citizens!
How do we usually deal with failures?

Using etcd or ZooKeeper

Membership services:

- Are reliable configuration stores
- Update their configuration sequence
- Invalidate old memberships

The problem is NOT solved at the microsecond scale.
Our Contribution:

- A Microsecond-scale Membership Service
- Detects failures in 15us
- Updates the membership in 10us
- Invalidates old memberships in 25us

uKharon reacts to failures in 50us by leveraging RDMA!
Remote Direct Memory Access (RDMA)

Allows µs-scale communication
Detects failures

Updates the membership

Invalidates old memberships
Microsecond-scale failure detection

- Timeouts do not help avoiding false positives
- Not all failures are equal

<table>
<thead>
<tr>
<th>Process failures:</th>
<th>Help from kernel → No timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGSEGV, Out-of-Memory, ...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel failures:</th>
<th>Help from NIC → No timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oops, core hang, ...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Catastrophic failures:</th>
<th>Catch-all → Timeout-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power failure, NIC crash, ...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byzantine failures:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer overflow, corruption...</td>
<td></td>
</tr>
</tbody>
</table>
Microsecond-scale failure detectors

Process failures

- App
- Kernel
- RDMA NIC
  - Capture process failure
  - CRASH

Kernel failures

- App
- Counter 2
- Kernel
- RDMA NIC
- Remote node

Take network synchrony out of the equation ➔ fast and accurate failure detection
Detects failures

Updates the membership

Invalidates old memberships
Microsecond-scale replication

- Uses Paxos
- But optimizes it for RDMA!
- Paxos, briefly:

```python
def acceptor-rpc(x):
    return transform(state, x)
```

**State**
- min_proposal
- accepted_proposal
- accepted_value

**Proposer**
- `propose`
- `decide`

**Acceptor**
- `acceptor-rpc(x)`
One-sided Paxos

# RDMA-based RPC
def rdma-rpc(x):
    state = READ(remote_state)
    ret = transform(state, x)
    CAS(state, remote_state)

    if CAS failed:
        retry
    else:
        return ret

One-sided RPC allows for blazing fast consensus!
Detects failures

Updates the membership

Invalidates old memberships
Membership invalidation

- What is the active membership?
- Learn via $\text{Active}(\text{Membership}) \rightarrow \text{bool}$
- $\text{Active}(M) == \text{true} \rightarrow M$ was active between invoc. and resp.
- High latency

How to make $\text{Active}$ cheap?
Microsecond-scale leases

- Clients lease Active’s response for ~20µs
- Renew their lease in the background
- NO synchronized clocks required
  - Only bounded clock drift for safety
- Delays view changes by no more than ~20µs

Leases make Active take ~40ns
Detects failures
Updates the membership
Invalidates old memberships

How does it perform?
Evaluation: setup
Evaluation: Replicating a KV-Store

uKharon helps beating the state of the art!
Evaluation: Are leases renewed in time?

Microsecond leases are stable!
Conclusion

- uKharon:
  - A membership service for μs-fast failover (down to 50μs)
  - Easy to integrate
  - Only 40ns latency overhead

Check out our paper for more details!

[GitHub Link: github.com/LPD-EPFL/ukharon]