CRaft: Using Accurate Clocks to Build a Multi-Leader Version of Raft

Feiran Wang*, Balaji Prabhakar*, Mendel Rosenblum*, Gene Zhang†
*Stanford University, †eBay Inc.
Overview

- Limitation with leader-based consensus protocols
- CRaft: a multi-leader extension to Raft enabled by accurate clocks
  - 2-2.5x throughput improvement
Replicated State Machines

- State machines execute same commands in same order
- Consensus: ensures all servers agree on the same log
- Tolerate f failures with 2f + 1 replicas
- Failure model: fail-stop

The Raft Consensus Protocol

- A widely used consensus protocol
- **Leader-based**: one server selected as leader
  - Leader accepts commands from clients and replicate to other servers
- Benefits: simple and efficient
- **Limitation**: leader is the bottleneck for throughput and scalability

Limitations with Single Leader

- Single leader limits throughput and scalability
- Measurements taken from HashiCorp Raft in CloudLab, using a key-value store

![Graphs showing latency vs throughput and throughput vs cluster size. The graphs illustrate system saturation and decreasing throughput with larger cluster sizes.](image-url)
Challenge in a Multi-Leader Protocol

• Challenge: how to coordinate leaders?
• Solution: ordering with accurate clocks
Our Approach: CRaft

<table>
<thead>
<tr>
<th></th>
<th>Raft</th>
<th>CRaft (Clocks + Raft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>![Single server and nodes]</td>
<td>![Multiple servers and nodes]</td>
</tr>
<tr>
<td>Safety &amp; Consistency</td>
<td>✓ ✓</td>
<td>Same guarantee as Raft</td>
</tr>
<tr>
<td>Practicability</td>
<td>✓ ✓</td>
<td>A simple add-on to Raft; easy to implement</td>
</tr>
</tbody>
</table>
CRaft Overview

- Run multiple concurrent Raft groups
- Merge logs entries across groups using timestamps from synchronized clocks
Clock Synchronization
Clock Synchronization

- $c_i(t)$ – system clock of machine $i$ at real time $t$
- Clock synchronization:
  \[ \forall t, \forall i, j: |c_i(t) - c_j(t)| \leq \epsilon \]
- $\epsilon$: precision

Huygens Clock Synchronization

• **Huygens**: a software clock synchronization system
• Measurements taken from CloudLab, 20 machines in a data center (with software timestamping)

### Distribution of clock offsets between servers

<table>
<thead>
<tr>
<th>Percentile</th>
<th>90th</th>
<th>99th</th>
<th>99.9th</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock offset</td>
<td>7us</td>
<td>11us</td>
<td>15us</td>
<td>26us</td>
</tr>
</tbody>
</table>

Network Time Protocol (NTP) vs Huygens

- **NTP precision:** ~20ms
- **Huygens precision:** ~20us
The CRaft Consensus Protocol
CRaft Overview

- Run multiple concurrent Raft groups
- Merge logs entries across groups using timestamps from synchronized clocks
Life of a Request

- Replicated on a majority of servers
- Safe and durable
Entry Timestamps

- Each entry has a timestamp assigned by its leader
- CRaft guarantees **monotonically increasing timestamps in each log**
Safe Times

How up-to-date is this log?

Now

Log
Safe time = 20

1 4 6 17 18 23 25 ...

Current entries:
timestamps <=
safe time

No entries come in
with a timestamp
smaller than safe time

• Safe to process log entries up to its safe time
• Safe times are updated when replicating entries
Merging

- Log entries are merged on each server, in increasing timestamp order
- Can merge up to the smallest safe time
- CRaft ensures merged log in monotonically increasing timestamp order
Maintaining Safe Times

• Goals: keeping safe times correct and up-to-date

Incorrect safe times cause inconsistencies

Log 1
1 4 6 17 18 ts = 18

Log 2
2 5 12 13 ts = 17

Small safe times block merging

Log 1
1 4 6 17 18 ts = 18

Log 2
2 5 ts = 7
Maintaining Safe Times

- Exchange safe time information at the same time of replicating log entries
- Must hear back from at least a majority and group leaders
Optimization: Fast Path

• For a write operation, confirmation is sufficient, as long as later reads can see the write result
Leader Election Policy

• Each group performs leader election independently
• Elect leaders according to priorities – distribute leaders to different servers
• Leaders can be auto re-balanced
Protocol Guarantees

• **State machine safety:** all servers will apply the same set of log entries to their state machines, in the same order.

• **Linearizability:** if operation A is responded before B starts, A will be executed before B
  - Example: a read following a write will return the newly written value
Evaluation
Experiment Setup

• Implementation
  • Based on HashiCorp Raft – a popular and well-optimized implementation

• Environment
  • CloudLab, single data center

• Workload
  • In-memory key-value store
  • Multiple clients send get or set requests concurrently
Throughput vs Cluster Size

- Up to ~2x read and ~2.5x write throughput compared to Raft
Latency vs Throughput

- CRaft improves throughput and latency under high load.
Timing Requirement

• Timing requirement for high performance:
  \[ \epsilon < \text{one-way message delay} \]

• Intuition from a server’s perspective: to execute a command, the other servers’ times need to be larger than mine
Performance vs Number of Clients

Throughput

- Raft
- CRAft - Huygens
- CRAft - NTP

Average Latency

Latency is bounded by clock difference

• NTP precision: ~20ms, Huygens: ~20us
## Conclusion: CRaft

<table>
<thead>
<tr>
<th></th>
<th>Raft</th>
<th>CRaft (Clocks + Raft)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td><img src="image1.png" alt="Illustration" /></td>
<td><img src="image2.png" alt="Illustration" /></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>A replicate log</td>
<td>A replicated log</td>
</tr>
<tr>
<td><strong>Safety &amp; Consistency</strong></td>
<td>✓</td>
<td>✓ Same guarantee as Raft</td>
</tr>
<tr>
<td><strong>Practicability</strong></td>
<td>✓</td>
<td>✓ A simple add-on to Raft; easy to implement</td>
</tr>
</tbody>
</table>