CORFU: A Shared Log Design for Flash Clusters

Authors: Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobber, Michael Wei, John D. Davis

EECS 591
11/7/18
Presented by Evan Agattas and Fanzhong Kong
Agenda

• Motivation
• Design
• Implementation
• Evaluation
• Applications
Past Structure

- Large scale storage systems
- Performance vs Safety
- Impractical on Disk or Ram
Flash Advantages

• Hundreds of concurrent reads
• Fast writes in pages
• Power efficient
• Becoming cheaper and cheaper
Motivation

• Shared log - Strong consistency
• Distributed log - Wear-leveling
• Consensus Engine, Key-Value Store
• Avoid Flash unit limitations
Distributed Log

• Overcoming disadvantage of Flash Memory
• Balance out writes
• Less failures over time
Motivation

- Shared log - Strong consistency
- Distributed log - Wear-leveling
- Consensus Engine, Key-Value Store
- Avoid Flash unit limitations
CORFU Design Overview

- Single Shared Log
- Client-centric Model
- Cluster of Flash Units
- Mapping log positions to memory
Design Overview
CORFU API

• Append(data)
  ○ Returns log position

• Read(log position)
  ○ Returns data

• Trim(log position)

• Fill(log position)
Flash Requirements

- Write-once Semantics
- Exposing Trim command
- Epoch numbers
- Reject lower epoch numbers
- Bonus: Infinite Address space
Main Functionality

1. Mapping
2. Finding the Tail
3. Replication protocol

- Handle Failures: Reconfiguration
Function 1: Mapping

- Projection Data Structure
  - Log positions -> Sets of Flash
- Client’s Projection copy
- Active Range
### Example Projection

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀: 20K, F₁: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₂: 20K, F₃: 20K</td>
</tr>
</tbody>
</table>

#### Log Positions

- F₀
- F₁
- F₂
- F₃
**Example Projection**

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
</table>
| 0 -- 40K-1   | \( F_0: 20K \)  
|              | \( F_1: 20K \)  |
| 40K – 80K    | \( F_2: 20K \)  
|              | \( F_3: 20K \)  |

**Log Positions**

```
F_0: 0
F_1: 0
F_2: 0
F_3: 0
```
Example Projection

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀: 20K</td>
</tr>
<tr>
<td></td>
<td>F₁: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₂: 20K</td>
</tr>
<tr>
<td></td>
<td>F₃: 20K</td>
</tr>
</tbody>
</table>

Log Positions

<table>
<thead>
<tr>
<th>F₀</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Example Projection

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀: 20K</td>
</tr>
<tr>
<td></td>
<td>F₁: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₂: 20K</td>
</tr>
<tr>
<td></td>
<td>F₃: 20K</td>
</tr>
</tbody>
</table>

## Log Positions

- F₀
- F₁
- F₂
- F₃
# Example Projection

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>(F_0: 20K) (F_1: 20K)</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>(F_2: 20K) (F_3: 20K)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Log Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

- **F_0**
  - 0
  - 2
  - ...
  - 40K - 2
- **F_1**
  - 1
  - 3
  - ...
  - 40K - 1
- **F_2**
- **F_3**
Example Projection

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀: 20K, F₁: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₂: 20K, F₃: 20K</td>
</tr>
</tbody>
</table>

Log Positions:

<table>
<thead>
<tr>
<th>F₀</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>40K</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40K-2</td>
<td>...</td>
<td>40K</td>
<td>1</td>
</tr>
</tbody>
</table>
Example Projection

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀: 20K, F₁: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₂: 20K, F₃: 20K</td>
</tr>
</tbody>
</table>

Log Positions

0 1 2 3 ...

40K 40K +1 40K +2 40K +3

F₀
0 2 ...
40K - 2

F₁
1 3 ...
40K - 1

F₂
40K 40K +2 ...
80K - 2

F₃
40K + 1 40K + 2 ...
80K - 1
Mapping - Reconfiguration

- Sealing command
- Auxiliary for storing projections

Steps
1. Sealing current projection
2. Writing new position to auxiliary
Reconfiguration Example

- 80K Positions allocated
- Replications: 2
- Current position: 50K

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀/F₁: 20K</td>
</tr>
<tr>
<td></td>
<td>F₂/F₃: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₄/F₅: 20K</td>
</tr>
<tr>
<td></td>
<td>F₆/F₇: 20K</td>
</tr>
</tbody>
</table>
Reconfiguration Example

- F6 Fails
- F7 Handles Reads
- New Unit for Writes
Reconfiguration Example

- Copy everything from F7 to F8

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀/F₁: 20K F₂/F₃: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₄/F₅: 20K F₇/F₈: 20K</td>
</tr>
</tbody>
</table>
Reconfiguration Example

- Need more log positions

<table>
<thead>
<tr>
<th>Log Position</th>
<th>Flash Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 40K-1</td>
<td>F₀/F₁: 20K</td>
</tr>
<tr>
<td></td>
<td>F₂/F₃: 20K</td>
</tr>
<tr>
<td>40K – 80K</td>
<td>F₄/F₅: 20K</td>
</tr>
<tr>
<td></td>
<td>F₇/F₈: 20K</td>
</tr>
<tr>
<td>80K – 120K</td>
<td>F₉/F₁₀: 20K</td>
</tr>
<tr>
<td></td>
<td>F₁₁/F₁₂: 20K</td>
</tr>
</tbody>
</table>
Function 2: Finding the Tail

• All clients contend for positions
• Safety-under-contention
  ○ Only one client can eventually “win:”
Naïve way – Finding the Tail
Naïve way – Finding the Tail
Naïve way – Finding the Tail
Naïve way – Finding the Tail
Naïve way – Finding the Tail
In the end
Cons for naïve way

• Con: Very bad performance
• Consider hundreds of concurrent requests
Alternative way – Finding the Tail

• Con: Very \textbf{bad} performance

• Consider hundreds of concurrent requests

• Alternative: Use \textit{Sequencer} (Similar to NoPaxos)
Sequencer – Finding the Tail
Sequencer – Finding the Tail

I want to append to log
Sequencer – Finding the Tail

write to position 1

write to position 2

write to position 3
Sequencer – Finding the Tail

write to position 1

Fails before writing to log
Junk Values

• Log entry assigned but not filled
  ○ Empty hole

• State machine can be blocking
  ○ Just like Multi-Paxos
Junk Values

- Log entry assigned but not filled
  - Empty hole

- State machine can be blocking
  - Just like Multi-Paxos

- Solution: Reserve junk value by other clients
  - “Junk aware” – No need to write it to flash
Contention Still Exists

write to position 1

write to position 2
Contestation Still Exists

Timeout!
Filled with junk value.
Contestation Still Exists

Timeout!
Filled with junk value.
Start to write value.
Function 3: Replication - Write

- Chain replication
- Overwrite error => safety-under-contention
Function 3: Replication - Read

- If we know the append was a success
  - Read any replica
- If we don’t know
  - Always read last replica in chain
Evaluation - Setup

- The hardware
  - 32 Intel X25V Flash Drives
  - 8 servers, 1 Gigabit links
  - 11 clients (44 processes)
Figure 6: Latency distributions for CORFU operations on 4KB entries.
Evaluation – Scalability

- Read: Linear to # of flash units
- Append: No increase after 28 flash units
- Reason: Sequencer was bottleneck
Evaluation - Performance

- Failure results in severe throughput drop – reconfiguration
- Sealing latency grows linear with # of flash units
CORFU Applications

- **CORFU-Store**
  - Put: append multiple keys and append a commit log.
  - Get: get log position and perform CORFU read.

- **CORFU-SMR**
  - Propose: append new command to log.
  - Execute: execute the command on the log.
Conclusion

• Single Shared Log
• Client-centric Model
• Fault Tolerance, almost scalable
• Consistency
• Main functions: Mapping, Finding tail, Replication