Corfu: A Shared Log Design for Flash Clusters

Presented by: Christopher Boggs, Tiberiu Vilcu
EECS 591 on 21 February 2018

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

1. Background and motivation
2. Design and implementation
3. Testing and applications
4. Takeaways and future works
The status quo

- Designs trade off safety vs performance
- Data centers are still full of disk (hard drives)
- Disk limits practicality of designs
- Common protocols implemented over disk and RAM servers
What they intend to do

Build an efficient shared log system using only commodity flash drives.

Potential roles/uses:

- Consensus engine
- Replication execution engine
- Primary data store
What they intend to do: CORFU

- Can be viewed as one large, distributed SSD
- Has a client-centric design
  - Clients maintain own address maps
  - Clients communicate with a sequencer
- Eliminates storage servers entirely
- Enables strong consistency and high speeds
- Ensures drives themselves are not a bottleneck
Taking advantage of flash

- Allows fast random parallel reads
- Allows fast writes to the “end” of memory
- Is power efficient and easier to scale
- Is becoming increasingly cheap to manufacture/purchase
- Can be written to irregularly to even out wear
Other work with flash

Enterprise hardware companies are also transforming flash use.

- Creating large data arrays for data centers
- Focusing on data compression and de-duplication
- Not working on shared-log design or consensus

Examples: Dell EMC, Pure Storage, Nimble Storage
High-level overview

Supports:

- Contention-free writes/appends to end of log
- Parallel reads from multiple replicas
High-level overview

Consists of:

- Clients
- CORFU client-facing API and map
- Requencer
- (Replicated) flash unit extents
High-level overview

Clients access flash units directly over the network via the Corfu library.

Each log position is mapped to flash pages in the cluster.
High-level overview
Requirements of flash devices

Needs commodity SSD’s that support:

- Fixed-sized flash pages
- Write-once semantics
- Trim (mark page as not in use)
- Seal (mark page with epoch)

2010 Intel drives used in paper; better and cheaper nowadays
CORFU API

- **append**(d)
  Appends d and returns occupied position

- **read**(l)
  Reads log at position l

- **trim**(l)
  Indicate no valid data at position l

- **fill**(l)
  Fill l with junk
Mapping addresses

Clients map log positions (addresses) to extents [sets]

Projection: disjoint ranges → list of extents → flash page on single extent

- Deterministic function used to select from list of extents
- Active range is the last disjoint range in map
Mapping addresses

Example Projection →
Range [0 – 40K] is mapped to F0 and F1.
Range [40K – 80K] is mapped to F2 and F3.

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

<table>
<thead>
<tr>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>40K</td>
<td>40K+1</td>
</tr>
<tr>
<td>40K+2</td>
<td>40K+3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>80K-2</td>
<td>80K-1</td>
</tr>
</tbody>
</table>
The sequencer

Simple networked counter:

- Assigns clients tokens to clients for log positions (200K/sec)
- Can be changed when necessary by reconfiguration (identity stored)
- Is a potential bottleneck in the system
- Is not required for safety or progress

Safety-under-contention is not “solved” by sequencer
Appending to the tail

Utilizes sequencer so not everyone fights for tail at once

1. Contact sequencer for a token
2. Hopefully not crash or die
3. Write to the given location in the log
Making “holes”

If client has a token then dies, spot in the log is a hole

- Slow down clients that read log in order
- Can exist within replica chains
Filling “holes”

Aggressive solution: clients fill discovered holes with junk [metadata]

- Uncompleted replica chains can be filled in by others
- Causes contention issue when original client just slow
Replication

Set of flash drives used to store the same data

- $f+1$ drives in set tolerate $f$ faults
- Safe when all reading clients observe the same value
- Durable when data visible only after reaching all replicas
Replication

Chain Replication:

- Write in deterministic order; *err_overwrite* prevents different values
- Read from last replica if success unsure; any otherwise
Reconfiguration

Necessary when flash drive fails or active range ends

Projections evolve over time:

- Sequence marked with epoch number
- New projection reflects writes made in older ones

Auxiliary defined as sequence of projections: stored elsewhere

All clients install new projection!
Reconfiguration

1. Seal flash units that change log position mapping
2. Write new projection at auxiliary (if first)
3. Read next projection from auxiliary

Any client can propose a projection, all have consensus!
Garbage Collecting

- Trim command for positions no longer using
- Client side
- Infinite space can become very sparse
  - Bloated projections
  - Even little data movement can reduce this drastically
Test setup

The hardware:

- 32 Intel X25V flash drives
- 8 servers, 1 Gigabit links
- 11 clients (44 processes)

2-way replication was used
Test results

Read:
- Scales linearly
- Almost perfectly in parallel

Write:
- Somewhat linear
- Tapers off, problems with hardware
Test results

- Failed drive drops throughput severely during reconfiguration
- Sealing latency grows linearly with flash units, but consistent
Test doesn’t show the big picture

- Test runs only lasted for 60 seconds
  - Live 24/7 in real life scenarios
  - Issues with holes and sequencer failovers could be more severe
- Only 4K writes / reads were done
  - Real usage has varying I/O sizes
  - SMR saving data in the log is much more to append
- Frequency of reconfigurations uncertain
Successes with CORFU

- Presents common solution to: consensus, state machine replication, replica recreation, consistent snapshots, etc.
- Utilizes “cheap” but fast hardware for dramatic results
- Scales very well with sizes of clusters
- Has a single most common point of failure: holes
  - Solved by sub-millisecond filling primitive
CORFU-Store

Key-value store feat. multi-key ops and snapshots

- Local maps has key to log-offset mapping
- PUTs append key/val pairs to log and append commits
- GETs query map for latest key-offset and reads
- Snapshots done by playing log to a certain point
- All linearizable and atomic
CORFU-SMR

Replaces Paxos in State Machine Replication

- Proposing done by appending to log
- Replicas execute in order by playing log forward
- Log entries can include data
- Persistent storage not necessary locally
CORFU DB

- Single system, shared core log
- Under-hood log is over a cluster
- Data structures at runtime
- “Higher performance Zookeeper”
- Utilizes Tango: distributed objects over CORFU
Related and future works

Hyder-Log: older proposal for shared log (inspiration)

- No sequencer
- Appends went through single unit

Yet to be implemented commonly; data centers are inflexible.

Will be interesting to see as flash gets more accessible!
Summary

● Need specialized designs for flash storage

● CORFU
  ○ Cluster of flash drives
  ○ Single, shared log
  ○ Client centric, no storage servers

● Simple, cheap, effective flash chips
References

CORFU: A Shared Log for Flash Clusters -- Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobber, Michael Wei, John D. Davis

CorfuDB VCA Presentation (June 2015) -- Dahlia Malkhi

CORFU Presentation (November 2012) -- Michal Czerski