CORFU
A Shared Log Design for Flash Clusters

Michał Czerski

SR.12/13

November 6, 2012

Table of Contents

1. Introduction
2. Motivation
3. Design and implementation
4. CORFU applications examples
5. Evaluation
6. Conclusion

Definition

Definition

CORFU stands for *Clusters of Raw Flash Units*, and also for an island near Paxos in Greece.

CORFU organizes a cluster of flash devices as a single, shared log that can be accessed concurrently by multiple clients over the network. CORFU is designed to work directly over network-attached flash devices, slashing cost, power consumption and latency by eliminating storage servers.

CORFU properties

CORFU provides

- strong consistency
- high throughput
- low latency
- distributed wear-leveling
- fault tolerance
- incremental scalability
- (network locality)
- (geo-distribution)

Possible applications

CORFU can be used to build

- databases
- transactional key-value stores
- replicated state machines
- metadata services

Flash properties

Flash storage is an ideal, but inherently flawed, medium for shared log designs

- fast, contention-free random reads
- fast sequential writes
- flash is read and written in increment of pages (typically of 4KB size)
- before page can be overwritten, it must be erased
- erasures can only occur at the granularity of multi-page blocks (of size 256KB)
- flash wears out and ages

This is why it is always best to write sequentially to flash. Almost all filesystems or databases designed for flash storage implement a log-structured design.

Why shared log?

Shared logs can be used

- as a building block for distributed applications that require strong consistency
- for failure atomicity and node recovery
- for recovery from multicast packet loss
- for consistent remote mirroring
- to build databases that speculatively execute transactions and then decide commit/abort status using log order
- as a consensus engine, providing functionality to consensus protocols such as Paxos

Why distributed log?

Some problems exist

- with multiple, independent logs, a total order no longer exists on all updates
- in partitioned system strongly consistent operations are limited in size and scope to a single partition
- skewed workloads can age drives at different rates

A distributed log solves these problems. Partitioning is ultimately necessary for achieving scale, but being able to do this on the level of a cluster is better than having to treat a single drive as single partition.

Setting

The setting for CORFU is a data center with a large number of application servers (which we call *clients*) and a cluster of *flash units*. The goal is to provide applications running on the clients with a shared log abstraction implemented over the flash cluster.

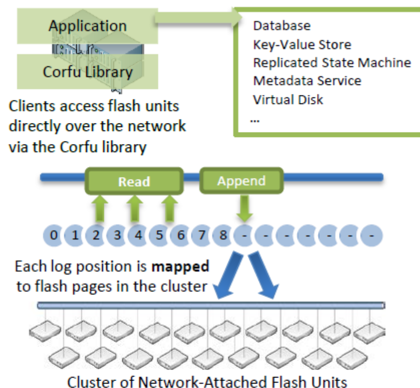

Figure 1: CORFU presents applications running on clients with the abstraction of a shared log, implemented over a cluster of flash units by a client-side library.

Guiding principles

- keep flash units as simple, inexpensive and power-efficient as possible
- place all CORFU functionality at the clients
- treat flash units as passive storage devices

We require some specific functionality from flash units, which we will discuss shortly.

CORFU library API

- *append(b)* - Append an entry *b* and return the log position *l* it occupies
- *read(l)* - Return entry at log position *l*
- *trim(l)* - Indicate that no valid data exists at log position *l*
- *fill(l)* - Fill log position *l* with junk

CORFU building blocks

To implement shared log abstraction three functions are needed

- A *mapping function* from logical positions in the log to flash pages on the cluster of flash units.
- A *tail-finding mechanism* for finding the next available logical position on the log for new data.
- A *replication protocol* to write a log entry consistently on multiple flash pages.

Flash unit requirements

- supports reads and writes on an address space of fixed-size pages
- reads on pages that have not yet been written should return an *error_unwritten* error code
- writes on pages that have already been written should return *error_overwritten* error code
- exposes a trim command
- exposes a seal command
- exposes an infinite address space (for efficient garbage collection)

Flash unit implementation

Flash unit maintains

- an epoch number
- a hash-map from 64-bit virtual addresses to the physical address space of flash
- a watermark before which no unwritten addresses exist
- a special address for marking positions as junk

Two flash unit types has been built to date

- Server + SSD
- FPGA + SSD

Mapping in CORFU

Definition

A *projection* is a local, read-only replica of data structure that splits the log into disjoint ranges. Each such range is mapped to a list of *extents* within the address spaces of individual flash units. Within each range in the log, positions are mapped to flash pages in the corresponding list of extents via a simple, deterministic function.

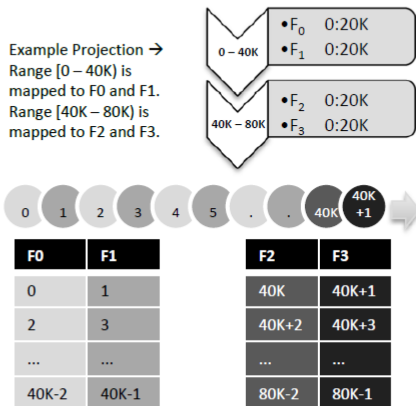

Figure 3: Example projection that maps different ranges of the shared log onto flash unit extents.

Changing the mapping

All operations on flash units are issued by clients within the context of a single projection. When some event occurs that necessitates a change in the mapping a new projection has to be installed consistently on all clients in the system. To achieve this CORFU uses a simple auxiliary-driven reconfiguration protocol which has two steps

1. seal the current projection
2. write the new projection at the auxiliary

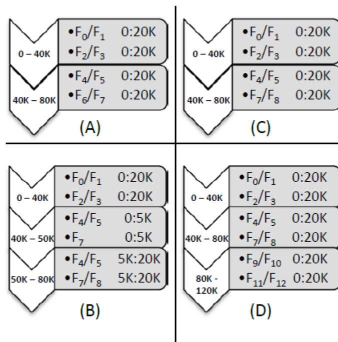

Figure 4: Sequence of projections: When F_6 fails in (A) with the log tail at 50K, clients move to (B) in order to replace F_6 with F_8 for new appends. Once old data on F_6 is rebuilt, F_8 is used in (C) for reads on old data as well. When the log tail goes past $80K$, clients add capacity by moving to (D).

Finding the tail in CORFU

- CORFU uses a dedicated sequencer that assigns clients *tokens*, corresponding to empty log positions.
- the sequencer does **not** represent a single point of failure
- the sequencer introduces a new way for possible failures - *holes* in log
- the sequencer does **not** remove the contention for log positions entirely

Replication in CORFU

Replication algorithm has to provide two properties

- safety-under-contention
- durability (has to tolerate f failures with just $f + 1$ replicas)

CORFU's solution: simple variant of chain replication

This approach allows us to fill holes left in the log by other clients that crashed midway through an append.

Reconfiguration and replication

The replica chain for a log position in a projection must satisfy some conditions

- if previous chain had a prefix of non-zero length storing a value, the new chain must have one as well
- if previous chain had zero-length suffix, the new chain must have one too
- at least one flash unit in the old replica set must be sealed in old projection's epoch

Garbage collection

- applications are required to use *trim* command on log positions that are no longer in use.
- CORFU has to deal with infinite address space of the log that can become increasingly sparse
- an adversarial workload can result in bloated projections
- some proactive data movement can keep even such projections under reasonable size

CORFU applications

Two applications already implemented

- CORFU-Store
- CORFU-SMR

Several other are currently in development, most notably Hyder, a recently proposed high performance database.

Setting

- a cluster of 32 Intel X25V drives
- 2 racks with 8 servers (2 drives each)
- 11 clients per rack
- each machine has 1 Gigabit link, 10 Gbps switch between racks
- appends are mirrored on drives in other rack
- reads go from the client to the replica in the local rack
- the total possible read throughput is 2 GB / sec
- append throughput is half that

Latency

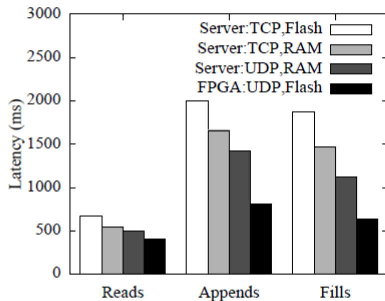

Figure 5: Latency for CORFU operations on different flash unit configurations.

Latency

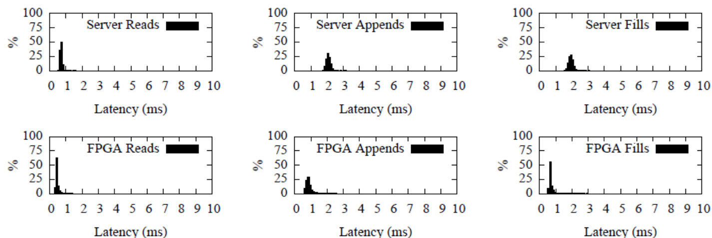

Figure 6: Latency distributions for CORFU operations on 4KB entries.

Throughput

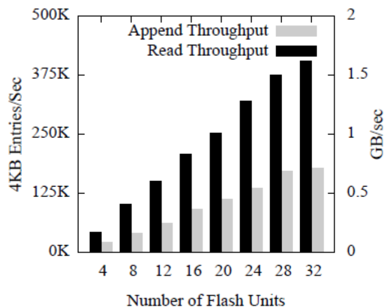

Figure 7: Throughput for random reads and appends.

Reconfiguration

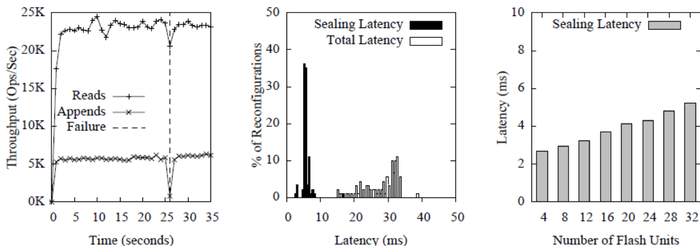

Figure 8: Reconfiguration performance on 32-drive cluster. Left: Appending client waits on failed drive for 1 second before reconfiguring, while reading client continues to read from alive replica. Middle: Distribution of sealing and total reconfiguration latency for 32 drives. Right: Scalability of sealing with number of drives.

Conclusion

- new storage designs are required to unlock the full potential of flash storage
- CORFU tries to fill this hole by presenting system, which organizes a cluster of flash drives as a single, shared log
- CORFU's client-centric design eliminates the need for storage servers in favor of simple, efficient and inexpensive flash chips

End

Thank you for your attention.