CORFU: A Shared Log Design for Flash Clusters

Paper by Mahesh Balakrishnan et al.

Presented by Qingyi Chen

Before Entering the topic…

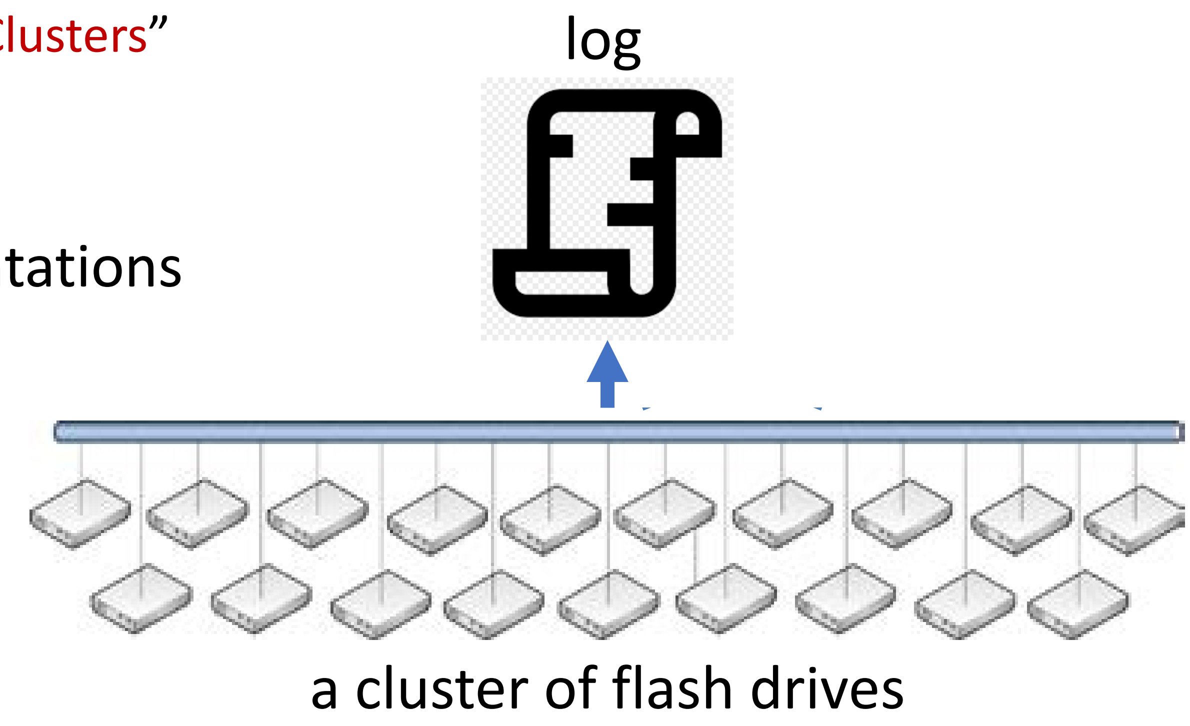
- A paper about "design"
  - unlike previous papers about insights and optimization
  - unlike a specific problem-solving algorithm or protocol
  - a collection of problems met when building a system, and proposed solutions

- A paper about a complex system
  - different problems in the system are not always closely related
  - logic flow is not linear, but like breath-first search
  - think as the designer

- Let's get into the topic

Introduction

- CORFU:
  - "A Shared Log Design for Flash Clusters"
  - Uses a cluster of flash drives
  - Implements a shared log
- Detailed design and Implementations
  - User interface
  - Core functions
  - Flash Unit Specifications
- Applications
- Evaluations



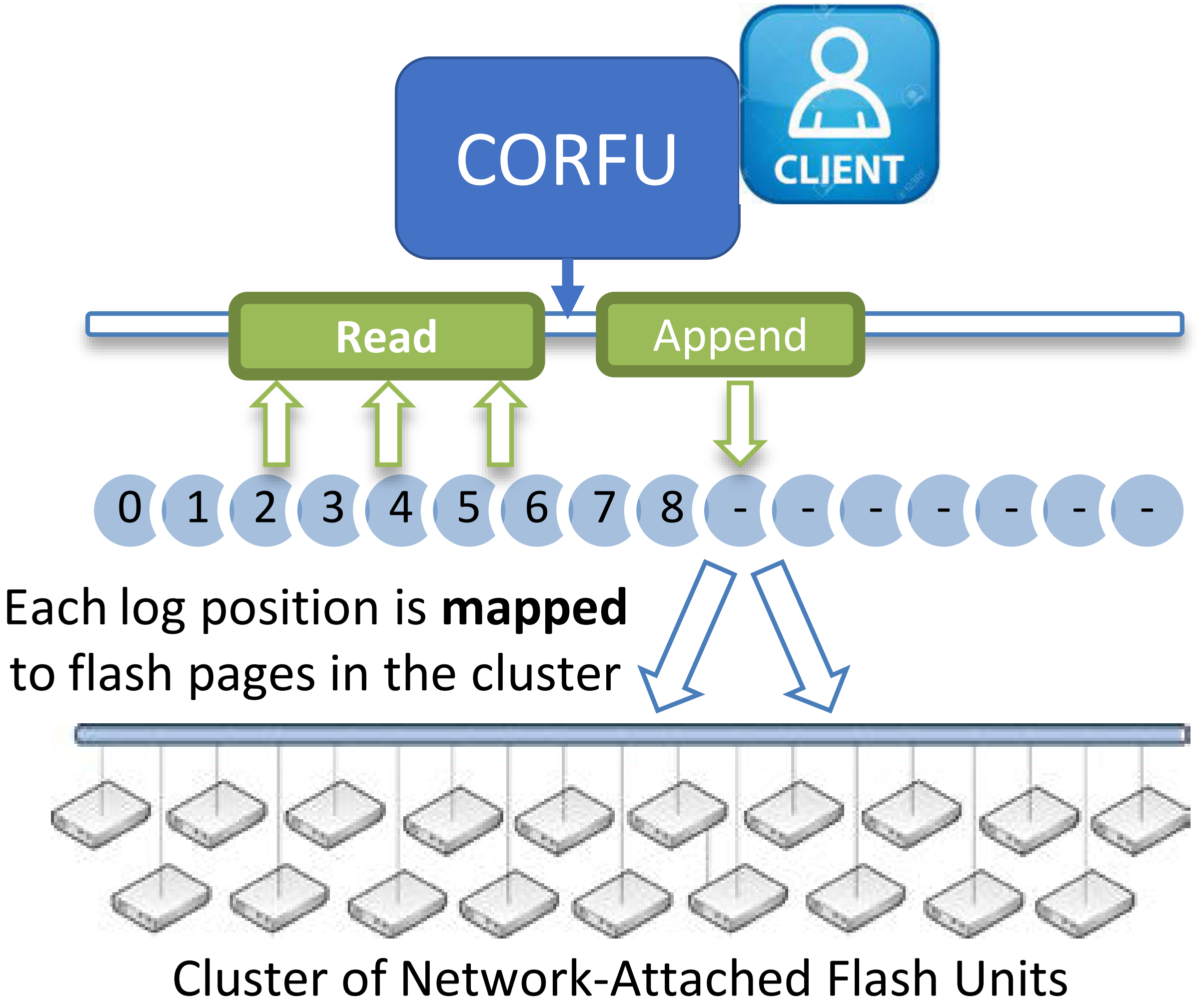
Motivation

- Why shared log?
  - High consistency
  - Making ordering easy
  - Straight-forward applications in distributed systems
    - State Machine Replication

- Flash Drives
  - Persistence, high throughput, low latency
  - Fast random read
  - Fast append

Design - Overview

- Client: interact using CORFU
- CORFU: the abstraction with “API”s
- Flash Units: the “log”



Design - Client Interface

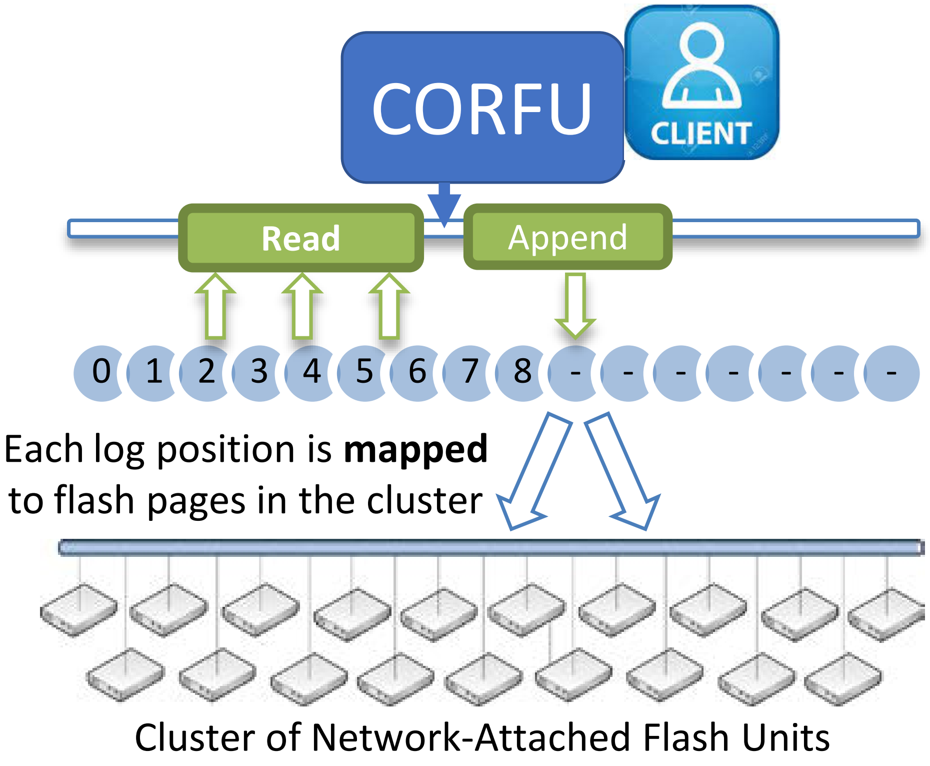
- Client: interact using CORFU
  - `Append(b)`

    // Append an entry b, gets the log position l it occupies
  - `Read(l)`

    // Gets the entry at log position l
  - `Trim(l)`

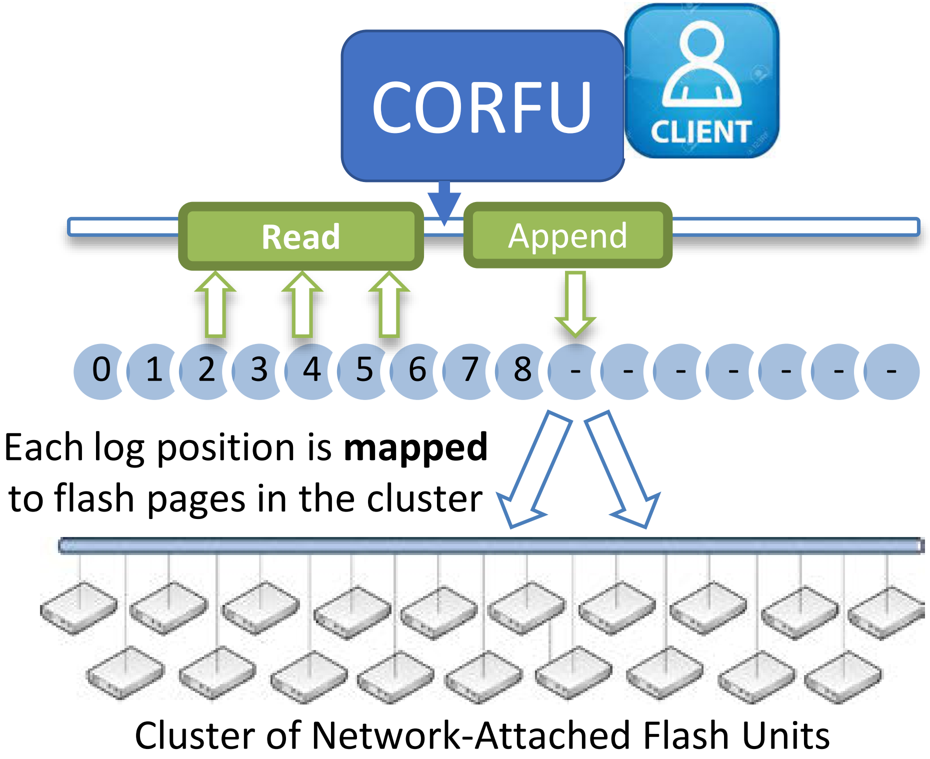
    // Indicates that no valid data exist at log position l
  - `Fill(l)`

    // Fills log position l with junk



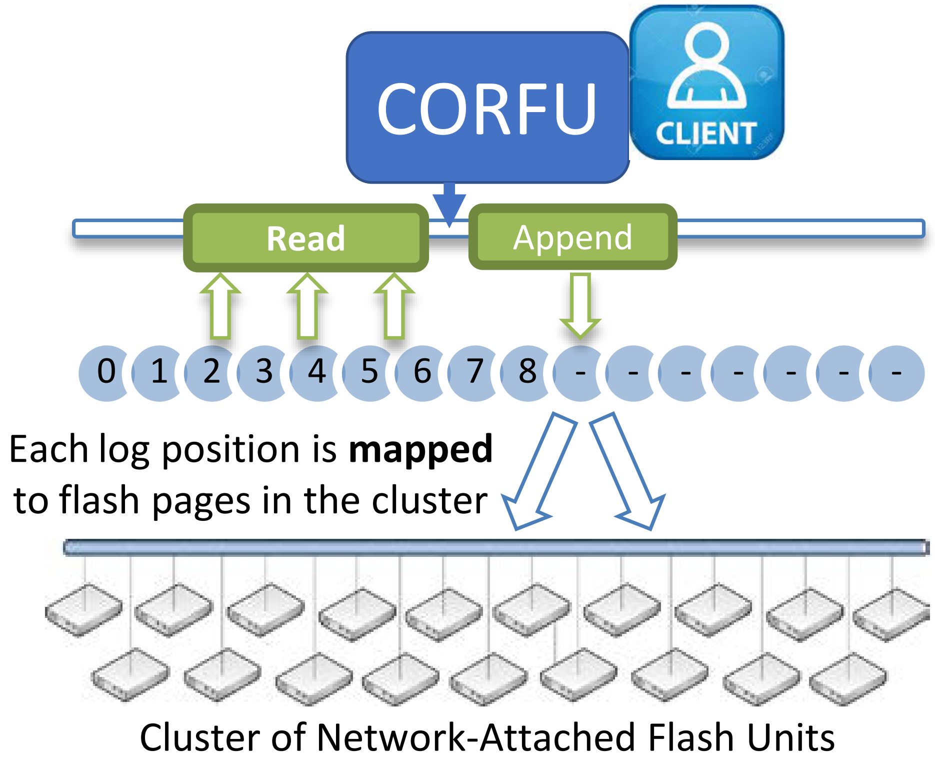
Design - CORFU API

- CORFU: the abstraction with “API”s
  - A mapping function
    Maps logical positions to flash pages
  - A tail-finding mechanism
    Finds the next available logical position on the log
  - A replication protocol
    Writes a log entry consistently on multiple flash pages



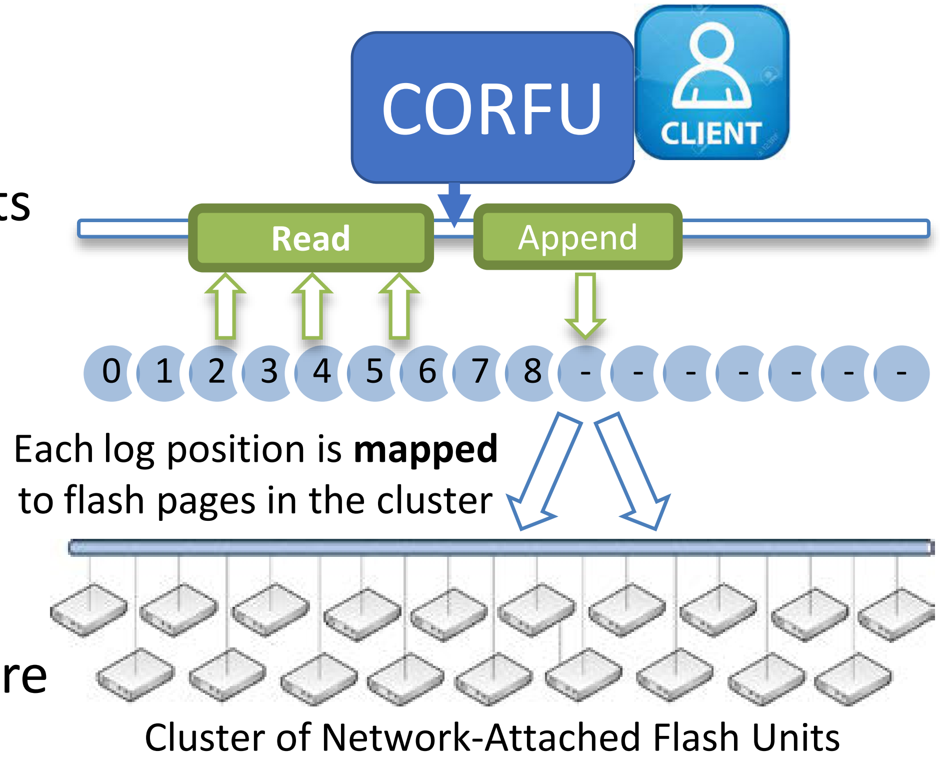
Design - Flash Unit Specifications

- Flash Unit: the "log"
  - Supports read/write in the unit of pages
  - Holds "Write-once" semantics
    - Returns an error if read on unwritten pages
    - Returns an error if written on written pages
  - Supports a "`trim`" command
    - Releases occupied pages
  - Supports a "`seal`" command
    - Every request is tagged with an `epoch` number
    - Rejects subsequent requests with a lower or equal `epoch` number



Design - The Full View

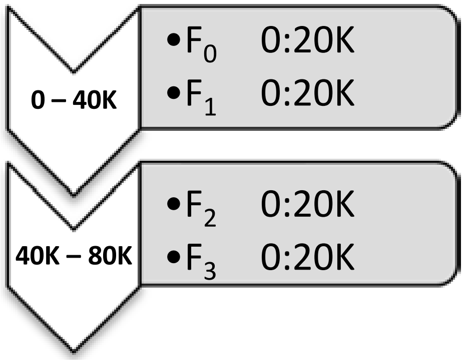
- When a client requests `read(l)`, CORFU
  - consults its mapping function
  - finds the corresponding flash pages in the flash units
  - Issues a read to the hardware
- When a client requests `append(b)`, CORFU
  - finds the tail position of the log
  - maps it to flash pages
  - initiates the replication protocol to write to hardware

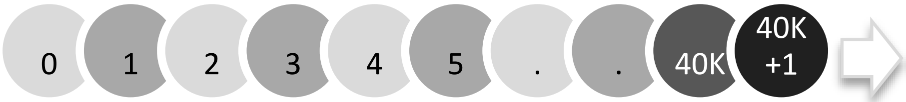


Implementation - Mapping (Overview)

- “**Projection**”: (1) splits log into disjoint ranges (2) maps log position to a list of extents
  - default: round-robin (right figure)
    - e.g., log position 0 -> F0: 0
    - e.g., log position 1 -> F1:0
    - e.g., log position 2 -> F0: 1
    - log position 45k -> ?
    - log position 45k -> F2: 2500
  - Any mapping function works
  - Replication
    - each extent associated with a replica set of units
    - e.g., F0: 0:20K -> F0 / F0’: 0:20K
  - Essentially providing a logical address space

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





F0	F1
0	1
2	3
…	…
40K-2	40K-1

F2	F3
40K	40K+1
40K+2	40K+3
…	…
80K-2	80K-1

Implementation - Mapping (View Change)

- **Problem**: “projection” is like views, and is subject to change
  - e.g., when a flash unit fails
  - therefore, we need `seal`

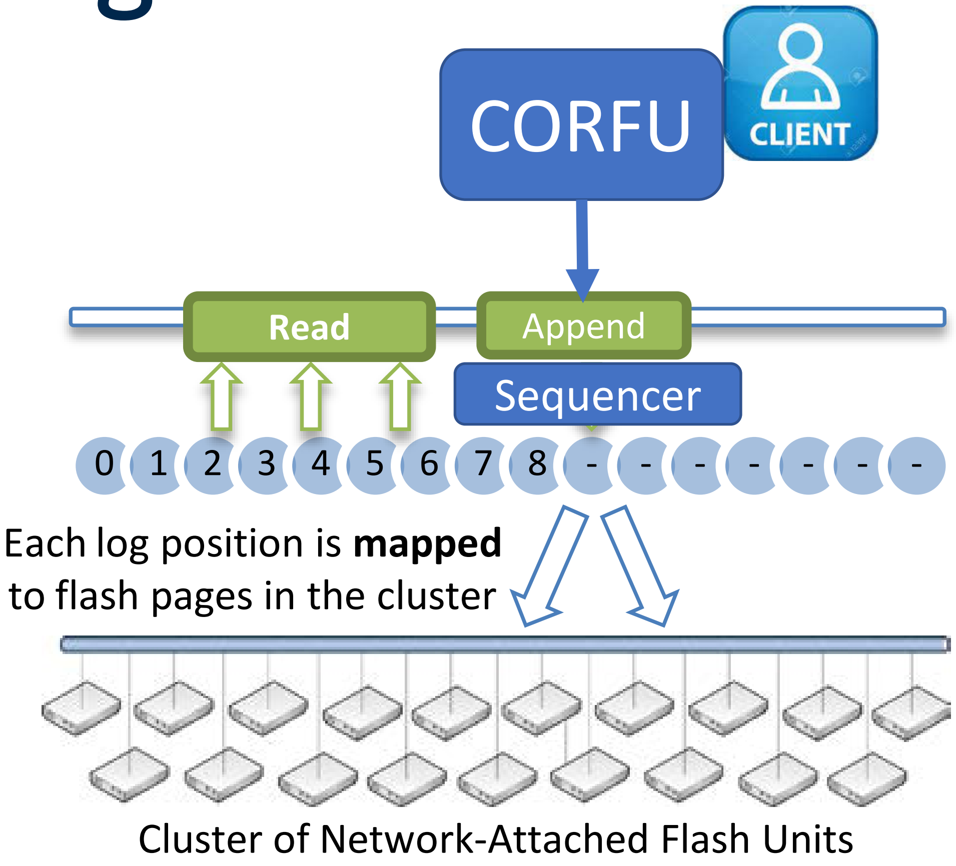
- **Requirement**: during change,
  - completed writes/trims must be kept
  - in-flight activities must be aborted and re-tried

- **Solution**: an auxiliary-driven reconfiguration protocol:
  - stores a sequence of projections called “auxiliary”
  - seals the current projection: in-flight activities rejected
  - writes the new projection at the auxiliary

Auxiliary			
Projection0 epoch:0	Projection1 epoch:1	Projection2 epoch:2	…

Implementation - Tail-Finding

- Naïve Approach:
  - clients contend for positions
- **Sequencer**:
  - “a simple networked counter”
  - client reserves a log position by consulting the sequencer first
- Hole?
  - what if a client reserves a log position, but fails…
  - let other clients fill the holes by marking a position “junk”
  - what if the writing client is just slow?



Implementation - Replication

* A log position is mapped to a replica set of flash pages

- **Requirement**:
    - safety-under-contention: when multiple clients write to the replica set for a log position, reading clients should observe a single value
    - durability: written data should be visible to reads only after it reaches f+1 replicas
- **Problem**:
    - different clients writing in parallel?
- **Solution**: a chaining protocol
    - a client-driven variant of Chain Replication
    - write in a deterministic order
    - read the last unit of the chain when unsure

Implementation - Flash Unit

- **Requirements**:
  - write-once semantics
  - a seal-capability
  - an infinite address space

- **Solutions**:
  - a hash-map from virtual address to physical address
  - an epoch number `cur_sealer_epoch`

Applications - CORFU-SMR

CORFU is ideal for implementing replicated state machine!

Each server

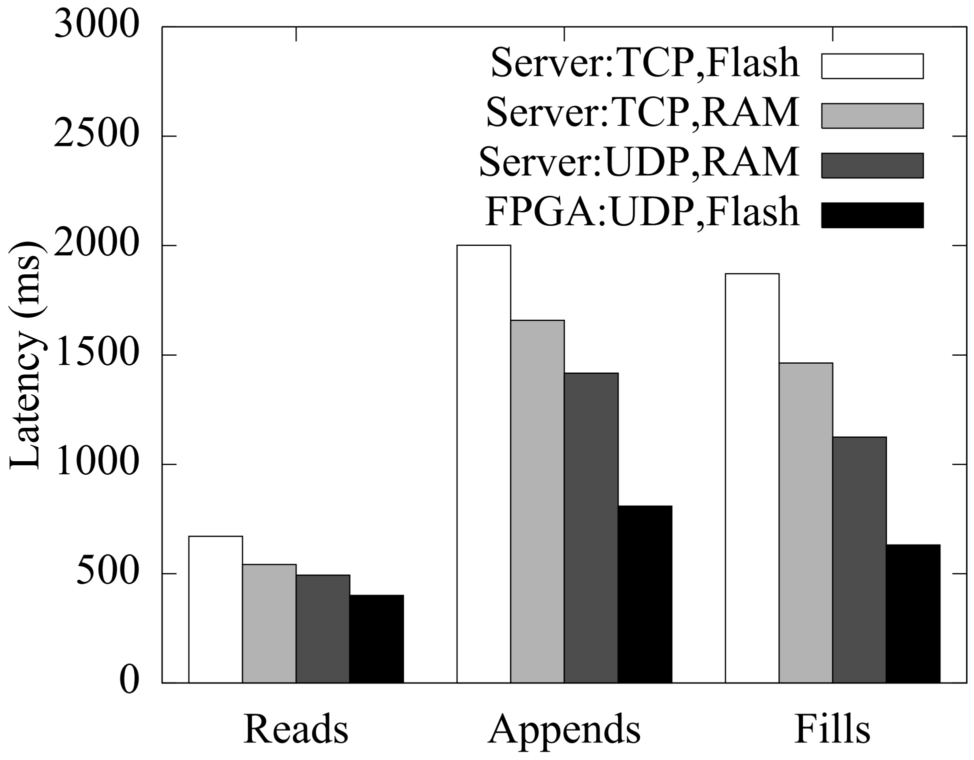
- plays the log forward to execute commands
- proposes new commands by appending them to log

Problem?

- With N servers running T commands/sec, the CORFU log see…
- N * T reads/sec.
- Probably would be solved by multicasting the log to servers

Evaluation - Latency

- Server: TCP, Flash means
  - server-attached flash unit that r/w on SSD
  - clients connect over TCP/IP
- The ordering of read/append/fill?
  - append/fill -> chain replica
- The latency of CORFU is very low

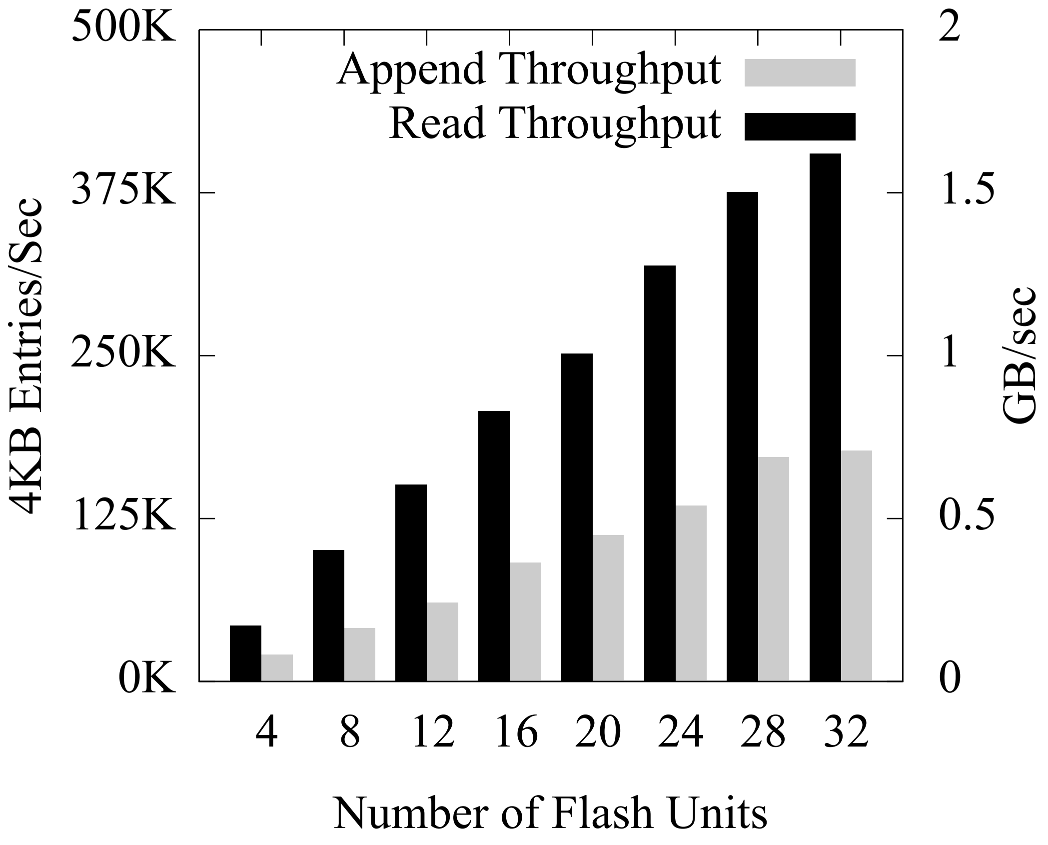


Latency for CORFU operations on different flash-unit configurations

Evaluation - Throughput

- High Throughput

- Scalability
  - nice scalability
  - appends' bottleneck: sequencer



Throughput for random reads and appends

Evaluation - Replication

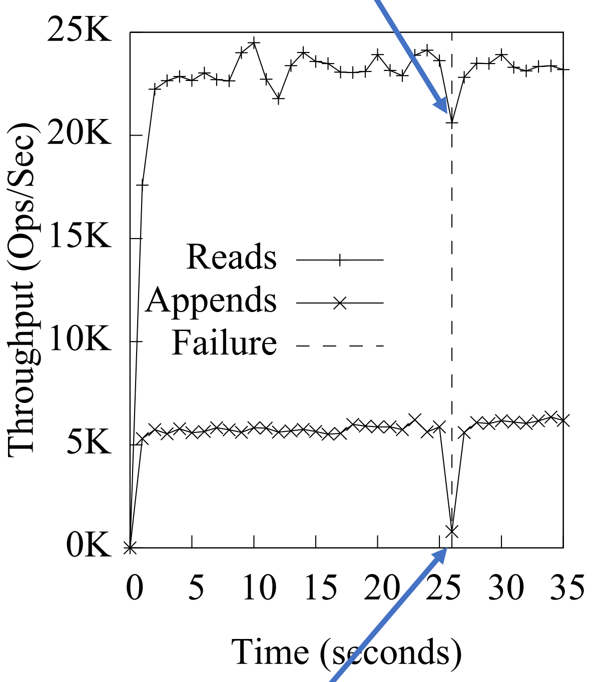
Throughputs:

- appending clients waits
- reading clients continue on alive replicas

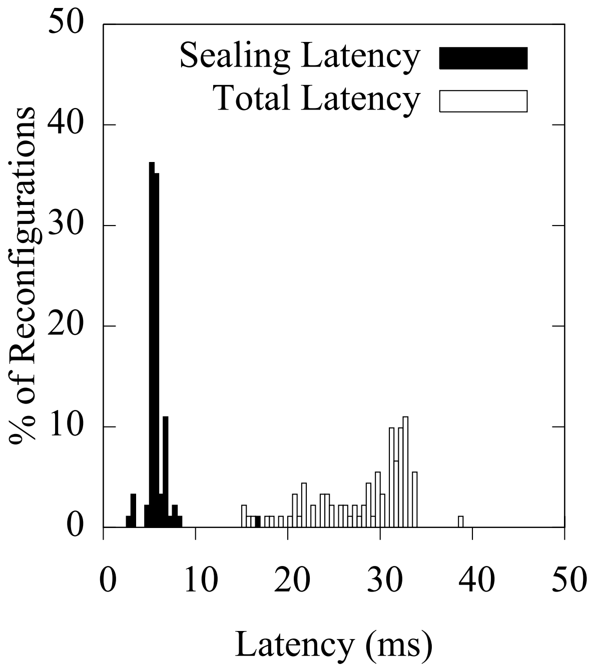
Latency:

- most of sealing latency < 10ms
- most of reconfiguration latency < 35ms

read throughput at failure



write throughput at failure



Conclusion

CORFU

- Organizes a cluster of flash drives as a shared log
- Features atomicity and durability
- Applicable in various distributed system problems

Take-away:

- The big-picture of designing a system
- Handling the tricky points with distributed system knowledge
  - e.g., replication using chain, sealing by keeping an epoch number

Ending

- Thank you for listening!
- Some details not covered
  - e.g., other applications of CORFU, like CORFU-Store
- Questions/corrections/discussions welcome!