Malacology

A Programmable Storage System

[Sevilla et al. EuroSys '17]

Michael A. Sevilla, Noah Watkins, Ivo Jimenez, Peter Alvaro, Shel Finkelstein, Jeff LeFevre, Carlos Maltzahn
University of California, Santa Cruz
Narrow Waist to Storage → Redundant Functionality

1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation

My App

atomic ops
consensus
data access
batching
migration

Storage System

✓ consensus  ✓ batching
✓ atomic ops   ✓ data access
✓ migration

File, Block, Object
Narrow Waist to Storage → Redundant Functionality

My App
- consensus
- atomic operations
- batching
- migration

Storage System
- consensus
- atomic ops
- batching
- data access
- migration

1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation
Programmable Storage Systems...

1. … expand narrow waist to storage
2. … expose internal functionality as interfaces
3. … simplify the construction of new services
4. … facilitate app-specific stacks with less layers
Programmable Storage Systems...

1. … expand narrow waist to storage

2. … expose internal functionality as interfaces

3. … simplify the construction of new services

4. … facilitate app-specific stacks with less layers
Motivation: Breaking the Narrow Waist Model

Developers willing to break layers and use non-standard APIs
Exposing Internal Functionality as Interfaces

[Not a fixed set of interfaces…]
Exposing Internal Functionality as Interfaces

My App
✓ consensus
✓ atomic ops
✓ batching
✓ migration

[Not a fixed set of interfaces…]
Programmable Storage Systems...

1. … expand narrow waist to storage

2. … expose internal functionality as interfaces

3. … simplify the construction of new services

4. … facilitate app-specific stacks with less layers
Composing Interfaces to Build Higher-Level Services

1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation

2X less Source code shipped with Ceph

CORFU
[Balakrishnan, NSDI'12]

Data I/O  Shared Resource  File Type  Service Metadata  Load Balancing  durability

storage daemon monitor daemon metadata daemons

Mantle
[Sevilla, SC'15]
Composing Interfaces to Build Higher-Level Services

1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation
Programmable Storage Systems...

1. .... expand narrow waist to storage
2. .... expose internal functionality as interfaces
3. .... simplify the construction of new services
4. .... facilitate app-specific stacks with less layers
Application-Specific Trade-offs for the Sequencer

- Cost of throughput is latency
- Knob for configurable performance characteristics

8 requests have a latency of 700 ms

Higher is Better

Lower is Better

Cost of throughput is latency

Knob for configurable performance characteristics
Application-Specific Load Balancing for the Sequencer

customized for sequencer workload

- Load balancing improves performance (free)
- Programmable load balancing → app-specific software stacks

Customized to be Less Aggressive

<table>
<thead>
<tr>
<th>Throughput (ops/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3K</td>
</tr>
<tr>
<td>2K</td>
</tr>
<tr>
<td>1K</td>
</tr>
</tbody>
</table>

Time (sec)

0 50 100 150 200 250 300

- No Balancing
- CephFS
- Mantle
Conclusion

Programmable storage

→ less layers, more robust, app-specific

Principled approach to using interfaces is effective:

- CORFU: distributed shared commit log
- Mantle: 2X less code, shipped in Ceph [Release]

Future work

- explore existing services; find proper set of interfaces
  - www.programmability.us
  - https://github.com/michaelsevilla/malacology-popper
Thank you… Questions?

Malacology: A Programmable Storage System

Michael A. Sevilla, Noah Watkins, Ivo Jimenez, Peter Alvaro, Shel Finkelstein, Jeff LeFevre, Carlos Maltzahn
Extra
Related Work

Software defined networks (control/data plane)  [Kumar SIGCOMM'13]

Software-defined storage  [Stefanovici, FAST'16; Thereska SOSP'13]

Policy-based provisioning  [Carlson, Whitepaper'15; Gracia-Tinedo IEEE'16]

Operating System: application-specific stacks  [SOSP'95, OSDI'96; SOSP'95]

Programmable Storage: adding interfaces to expose functionality to get application-specific stacks; approach and applying it to storage
Conclusion: Programmable Storage Systems...

- expand the narrow waist to storage

- expose interfaces to internal functionality; we built:
  - CORFU: shared commit log
  - Mantle: file system metadata load balancer

- facilitate building application-specific software stacks

www.programmability.us
https://github.com/michaelsevilla/malacology-popper
Customizable Trade-offs for the Sequencer

1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation

- Higher is Better
- Lower is Better

Enhance!
Re-Implementing Mantle on Malacology

- New features:
  - 2X src code reduction
- Shipped in Ceph [Luminous]
Building a Distributed Service

1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation

- ZLog
- Atomic TXNs
- Serialization
- Batching
- Versioning
- Atomic TXNs
- Consensus
- Migration
- Transactions
- Batching/Serialization
- Migration
Traditional Storage APIs are Convenient… … but Narrow
## Storage Systems → Common Internal Abstractions

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Ceph</th>
</tr>
</thead>
<tbody>
<tr>
<td>consensus</td>
<td>monitor daemon</td>
</tr>
<tr>
<td>transaction</td>
<td>object server daemon</td>
</tr>
<tr>
<td>serialization</td>
<td></td>
</tr>
<tr>
<td>data access</td>
<td></td>
</tr>
<tr>
<td>migration</td>
<td>other daemons</td>
</tr>
<tr>
<td>persistence</td>
<td>(metadata server in this case)</td>
</tr>
</tbody>
</table>

**Functionality**
- consensus
- transaction
- serialization
- data access
- migration
- persistence

**Monitor Daemon**
- M

**Other Daemons**
- object server daemon
- other daemons (metadata server in this case)
## Storage Systems → Common Internal Abstractions

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Ceph</th>
<th>Malacology</th>
</tr>
</thead>
<tbody>
<tr>
<td>consensus</td>
<td>![M]</td>
<td>service metadata</td>
</tr>
<tr>
<td>transaction</td>
<td>![data I/O]</td>
<td>data I/O</td>
</tr>
<tr>
<td>serialization</td>
<td>![shared resource]</td>
<td>shared resource</td>
</tr>
<tr>
<td>data access</td>
<td>![file type]</td>
<td>file type</td>
</tr>
<tr>
<td>migration</td>
<td>![load balancing]</td>
<td>load balancing</td>
</tr>
<tr>
<td>persistence</td>
<td>![durability]</td>
<td>durability</td>
</tr>
</tbody>
</table>
Example: Service Metadata Interface

1. Problem

2. Solution

3. Higher-Level Services

4. Evaluation

- consensus
- transactions
- batching/serialization
- data access
- migration

My App

- consensus
Example: Service Metadata Interface

- authentication
- logging/debugging
- configs
- cluster maps
- metadata for higher-level service
  (e.g., binaries, version numbers)
Customizable Trade-offs for the Sequencer

1. Problem

2. Solution

3. Higher-Level Services

4. Evaluation

Throughput (ops/sec)

Bad Performance

100K

10K

100

1

10

100

1K

Batch Size

Quota

Diminishing Returns

Good Performance

Avg. Latency (us)

10^7

10^6

10^5

0

10

20
Narrow Waist to Storage → Redundant Functionality

1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation

Consensus → Data Access → Batching → Migration

Atomic Ops

Storage System

File, Block, Object

My App

 ✓ consensus
 ✓ atomic ops
 ✓ batching
 ✓ data access
 ✓ migration
1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation
1. Problem
2. Solution
3. Higher-Level Services
4. Evaluation
Narrow Waist to Storage $\rightarrow$ Redundant Functionality

My App

-consensus
-atomic ops
-batching
data access
-migration

Storage System

File, Block, Object