outline!

Submission Type: Research

Abstract

1 Introduction

Storage systems are increasingly providing features that take advantage of application-specific knowledge to achieve optimizations and provide unique services. However, this trend is leading to the creation of a large amount of low-level software extensions that will be difficult to maintain as system software and hardware evolve.

The widely deployed Ceph distributed storage system is an example of a storage system that supports application-specific extensions in the form of custom I/O interfaces to objects within the underlying RADOS object storage system [5, 6]. Organizations are increasingly reliant upon these extensions as is shown in Figure 1 by a marked increase in the number of object operations that are packaged are part of the Ceph distribution. In addition to the growth of the quantity of operations, Figure 1 also depicts the amount of low-level C++ written to implement the set object operations. Unfortunately, this code is written assuming a performance profile defined by the combination of the hardware and software versions available at the time of development. As Ceph continues to evolve at a fast pace, the cost of adapting application-specific codes to changing assumptions regarding performance may become significant.

This trend is also not limited to Ceph. AWS Lambda, Redis Lua, Redis Modules, KV-Drives, and Rhea, are all recent examples of storage systems embracing the power of including application-specific semantics.

Previous work in active storage has focused on the utility of moving computation closer to data in an effort to reduce data movement and take advantage of cpu and i/o parallelism, and gave little to no attention to security or performance isolation concerns. In contrast, the interfaces being deployed today focus on data management, indexing, and physical format.

We propose the introduction of a declarative language for building object based interfaces that allows a storage system to meet the needs of an application throughout the development process without requiring rewrites.

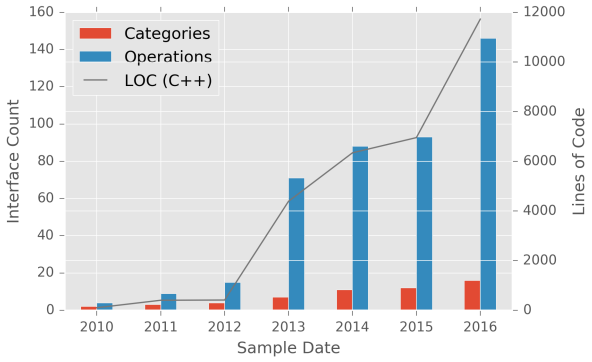


Figure 1: [source] Growth of officially supported, custom object interfaces in RADOS over 6 years. A *method* is a specific object interface and a *class* is a logical grouping of methods.

2 Background

In this section we briefly describe Ceph and object-based interface development. We will use CORFU as a motivating example of an interface, and then show the challenges with the current low-level mechanisms for defining interfaces.

2.1 Domain-specific Object Interfaces

And the construction of object operations is not limited to core Ceph developers; the development community has been receptive to contributions with the recent inclusion by CERN developers of an extension for performing limited numeric operations on object data [1], and all While Ceph has not yet reached the point of directly exposing these features to users, the recent inclusion of a mechanism for dynamically defining extensions using Lua [2] suggests that aspects of this may soon appear.

Ceph allows the construction of domain specific interfaces. There are many different object interfaces in production today. An example of the spectrum of interface types is shown in Table 1. Object classes are developed as a transactional composition of native storage interfaces.

2.2 Motivating Example: CORFU

The primary motivating example we will use in this paper is the CORFU distributed shared-log designed to pro-

Category	Specialization	Methods
Locking	Shared Exclusive	6
Logging	Replica State Timestamped	3 4 4
Garbage Collection	Ref. Counting	4
Metadata	RBD RGW User Version	37 27 5 5

Table 1: A variety of RADOS object storage classes exist that expose reusable interfaces to applications.

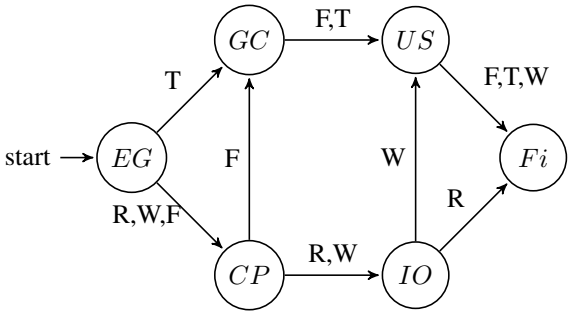


Figure 2: State transition diagram for read (R), write (W), fill (F), and trim (T) CORFU operaitons. The states epoch guard (EG), check position (CP), and update state (US) access metadata. The I/O performs a log entry read or write, and garbage collection (GC) marks entries for reclamation.

vide high-performance serialization across a set of flash storage devices [3]. The shared-log is a powerful abstraction useful when building distributed systems and applications, but common implementations such as Paxos or Raft funnel I/O through a single node limiting the throughput of log operations [4]. The CORFU protocol addresses this limitation by de-coupling log entry storage from log metadata, making use of a centeralized, volatile, in-memory *sequencer service* that assigns log positions to clients that are appending to the log. Since the sequencer is centeralized serialization is trivial, and the use of non-durable state allows the sequencer service to operate at very high rates.

While a full description of the CORFU protocol is beyond the scope of this paper, it is important to highlight the usage of application-specific flash I/O interfaces in CORFU that are used to guarantee correctness during failure. Each flash device in CORFU exposes a 64-bit write-once address space consisting of the primary interfaces *write(pos, data)* and *read(pos)*, as well as *fill(pos)* and *trim(pos)* that invalidate and reclaim log entries, respectively. All I/O operations initiated by clients are tagged with an *epoch* value, and flash devices are expected to reject client requests requests tagged with an old epoch value. To facilitate recovery or handle system reconfiguration in CORFU, the storage devices are also required to support a *seal(epoch)* command that marks the latest epoch and returns the maximum position written to that device. The seal interface is used following the failure of a sequencer to calculate the tail of the log that the sequencer should use to repopulate its in-memory state.

Programmable Storage and CORFU Two aspects of CORFU make its design attractive in the context of the Ceph storage system. First, CORFU assumes a cluster of flash devices because log-centric systems tend to have a larger percentage of random reads making it difficult to achieve high-performance with spinning disks. However, the speed of the underlying storage does not affect correctness. Thus, in a software-defined storage system such as Ceph a single implementation can transparently take advantage of any software or hardware upgrades, and make use of data management features such as tiering in RADOS, allowing users to freely choose between media types such as SSD, spinning disks, or emerging NVRAM technologies.

The second property of CORFU relevant in the context of Ceph is the dependency CORFU places on the storage device interface. As was previously discussed, in order to enforce constraints defined within the CORFU protocol, storage devices are expected to expose a domain-specific interface specially designed for the protocol. While the authors of the CORFU paper describe both a host-based and FPGA-based solution, RADOS directly supports the concept of custom storage interfaces to its objects that can be used to achieve the transactional semantics necessary to implement the CORFU storage interfaces. This feature is called RADOS object classes, and is one of the most powerful features offered by RADOS for designing storage applications.

A CORFU Object Interface The state-machine shown in Figure 2 shows the composition of actions for each component of the CORFU interface. For example, each operations begins with an *epoch guard*, and a *write* operation will consult an index and perform a bulk I/O operation. The primary concern when mapping an interface onto Ceph is deciding how each native interface is used in the interface composition.

2.3 Physical Design

There is a finite set of strategies for mapping the CORFU interfaces onto the current set of native I/O interfaces used to construct object interfaces. Table 2 shows the design space for mapping the CORFU interfaces onto Ceph. In

Map	I/O	Entry Size	Addressing	Metadata
1:1	KV	Flex	Ceph	KV/BS
	BS	Flex	Ceph/VFS	KV/BS
N:1	KV	Flex	KV/BS	
	WR	Fixed	VFS	KV/BS
	AP	Flex	KV/BS	KV/BS

Table 2: The high-level design space of mapping CORFU log entry storage onto the RADOS object storage system.

order to select the best mapping we performed a parameter sweet over the design space.

Figure 3a shows the result of a parameter sweep which clearly shows that an N-1 mapping based on the bytestream interface provides superiour performance. However, the other two graphs show the same interfaces running on an old version of Ceph that show the same decision would not have been the optimal choice ***note: for the outline these are not the real graphs we'll be using***.

3 Programming Model

In this section we are going to be describing our way of creating interfaces.

4 Other Interfaces

Here we show the derivation of two other interfaces that are in production in Ceph today to demonstrate the generality of our interface.

5 Evaluation

6 Related Work

7 Conclusion

References

[1] Merged pull request for cls_numops. https://github.com/ceph/ceph/pull/4869. Accessed: 2016-05-12.

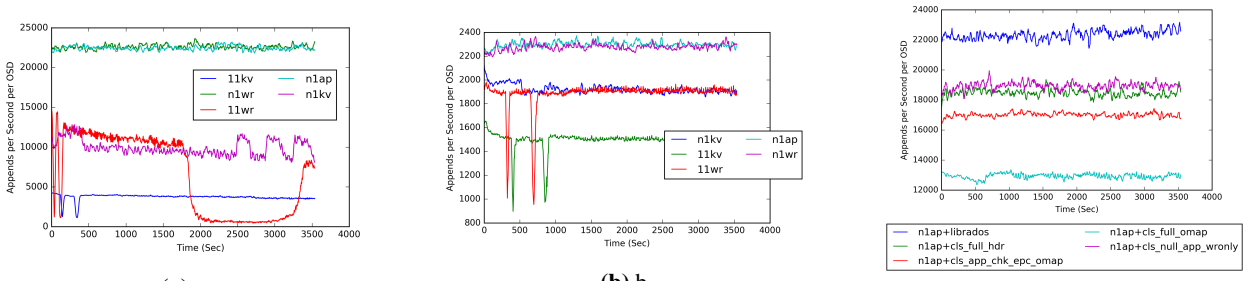
[2] Pull request for cls_lua. https://github.com/ceph/ceph/pull/7338. Accessed: 2016-05-12.

[3] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobber, Michael Wei, and John D. Davis. Corfu: A shared log design for flash clusters. In *NSDI'12*, San Jose, CA, April 2012.

[4] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[5] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *OSDI'06*, Seattle, WA, November 2006.

[6] Sage A. Weil, Andrew Leung, Scott A. Brandt, and Carlos Maltzahn. Rados: A fast, scalable, and reliable storage service for petabyte-scale storage clusters. Reno, NV, November 2007.



(a) a

(b) b

(c) c

Figure 3: Shown here are the graphs and such that demonstrate that the same physical design choices are not the same between differing version of Ceph even on the same hardware.