outline!

Submission Type: Research

Abstract

1 Introduction

Cloud providers are increasingly reliant upon application-specific extensions to distributed storage system such as Ceph as is shown in Figure 1 by a marked increase in Ceph extensions used in production.

The construction of extensions is not limited to core 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].

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. 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.

If the growth illustrated in Figure 1 continues, the amount of low-level C++ will grow large. Unfortunately, much of this code is written assuming a performance profile defined by a combination of the hardware and software versions available at the time of development. As Ceph development continues at a fast pace, the cost of adapting to changing assumptions regarding performance will become significant.

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.

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.

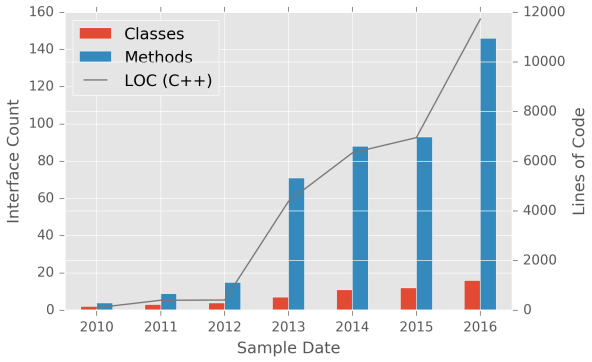


Figure 1: [source] Growth of officially supported, custom object interfaces in RADOS over 6 years. A *method* is a specific object interface and a *class* is a logical grouping of methods.

Category	Specialization	Methods
Locking	Shared Exclusive	6
Logging	Replica State Timestamped	3 4 4
GC	Reference 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.

2.1 Domain-specific Object Interfaces

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.

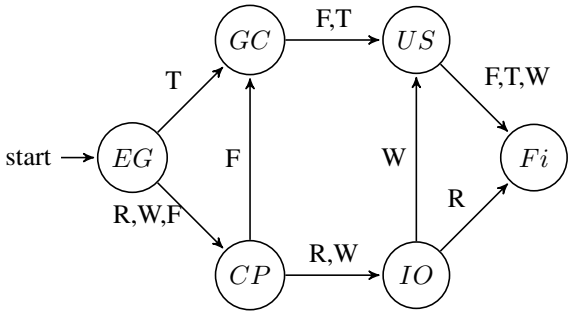


Figure 2: State transition diagram for read (R), write (W), fill (F), and trim (T) CORFU operaitons. The states epoch guard (EG), check position (CP), and update state (US) access metadata. The I/O performs a log entry read or write, and garbage collection (GC) marks entries for reclamation.

2.2 Motivating Example: CORFU Log

Short description of CORFU system [3]. It's a log! While CORFU relies on custom interfaces built into flash devices, an implementation in Ceph relies upon custom device interfaces to enforce its protocol.

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 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.

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.

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

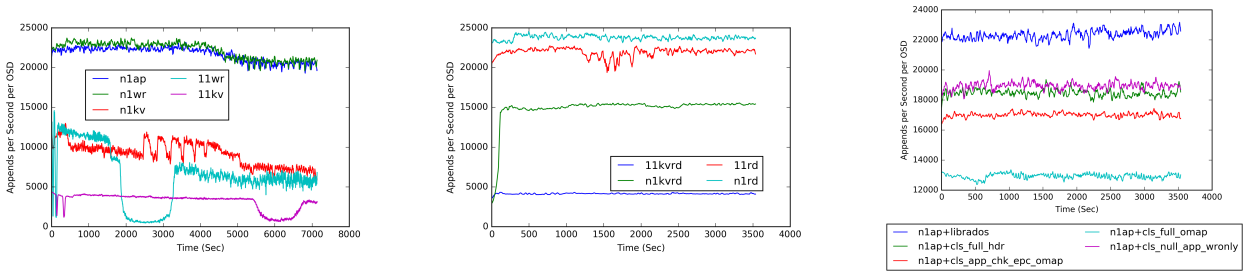
7 Conclusion

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.



(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.