In-Flight Data Management for Distributed Storage Systems

Kendrick Boyd Carlos Maltzahn
boydk@lawrence.edu carlosm@soe.ucsc.edu
Storage Systems Research Center (SSRC) at UCSC

This work was completed as part of UCSC's SURF-IT summer undergraduate research program, an NSD CISE REU Site. This material is based upon work supported by the National Science Foundation under Grant No. CCF-0552688.

Background

To handle petabytes of data and billions files ranging from bytes to terabytes in size, storage systems (like Ceph*) may use object based storage.
Object Based Storage:

- separates metadata and data on different machines
  - data is stored on Object Storage Devices (OSDs)
  - metadata is stored on Metadata Servers
- files divided into objects and striped on several OSDs

* Ceph is an object based storage system being developed by SSRC

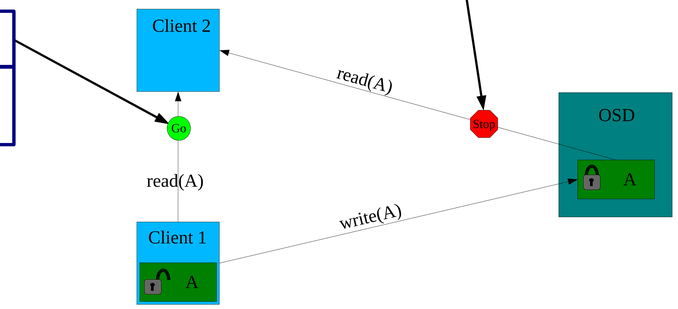
Problem

Read must wait until Client 1 closes and transmits A back to OSD -> long wait

Data that is “in-flight” is being modified by a client, like file A, and has very slow access times for other clients.

Solution

Most recent data is always immediately available from Client 1



Assumptions

- locking by file region
- object based storage system
- fastest network in cluster is between clients
- client caches file when opened as read only or readwrite

Solution Requirements

- last write immediately seen by all clients
- reduced latency for opening in-flight regions
- scalable
- minimal data loss due to client failures

In-Flight Data Design Space

We have found the circled options to best fulfill the specified requirements.

Forwarding – options about when a request is forwarded to a client

Which requests?	(read only)	(read/write)
Forward to which client caches?	read only	(read/write)
Maximum sharing tree height?	finite constant	(infinite)
When?	transmission only	(open to close)

Coherence – options pertaining to coherence of caches

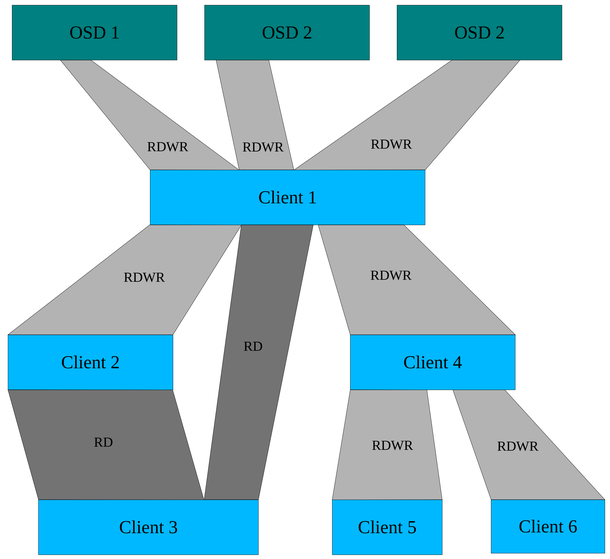
Local caches updated?	(yes)		no
Updated how?	(invalidate with callbacks)	update every write	
Changes available to other clients after?	(write to local cache)		file close

Notes

- Only 1 client has true write capabilities for a region at a time.
- If client 2 opens region A for readwrite and obtains the region from client 1 then client 1 cannot modify region A until client 2 has closed A.

Sharing Trees

Clients writing to clients create a hierarchy of client caches, shown below.



Client Failures

Clusters of 1000s of clients have frequent client failures, thus in-flight data management must:

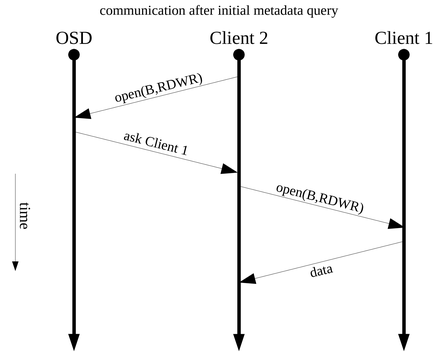
- discover client failures
- rebuild sharing trees without failed client
- recover maximum in-flight data
- storage operations continue normally after recovery

Idea: failure recovery method

- client that receives lock directly from OSD is authority for that region and is root of sharing tree
- clients with a readwrite cache store entire tree below itself and path to root
- all writes lazily replicated up tree to root
- non-root client failure
  - root redirects around failed client
  - little or no data loss due to lazy writes up tree
- root client failure
  - OSD rebuilds sharing tree by finding the 2nd tier clients
  - some data loss expected
- reduces traffic at OSD except for root client failure

Example 1

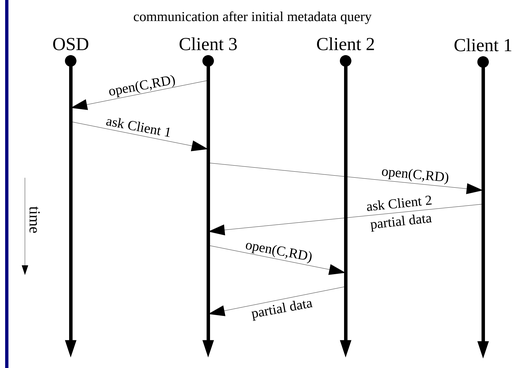
Initial State:	Client 1 has region A opened for readwrite.
Action:	Client 2 opens region B for readwrite. B⊂A



Final State:	Client 2 has region B in local cache for writing. Client 1 cannot make changes to region B. Client 1 will be notified of any changes to region B.

Example 2

Initial State:	Client 1 has region A opened for readwrite. Client 2 has region B opened for readwrite. B⊂A.
Action:	Client 3 opens region C for read. C overlaps A and B



Final State:	Client 3 has region C in local cache for reading. Client 3 will be notified of any changes to region C.

Future Work

- establish how file size, last modified and last accessed are handled
- investigate other algorithms for recovery after client failure
- implement in Ceph and benchmark