Towards a Unified Object Storage Foundation for Scalable Storage Systems

Authors: Cengiz Karakoyunlu, Dries Kimpe,
Philip Carns, Kevin Harms, Robert Ross, Lee Ward

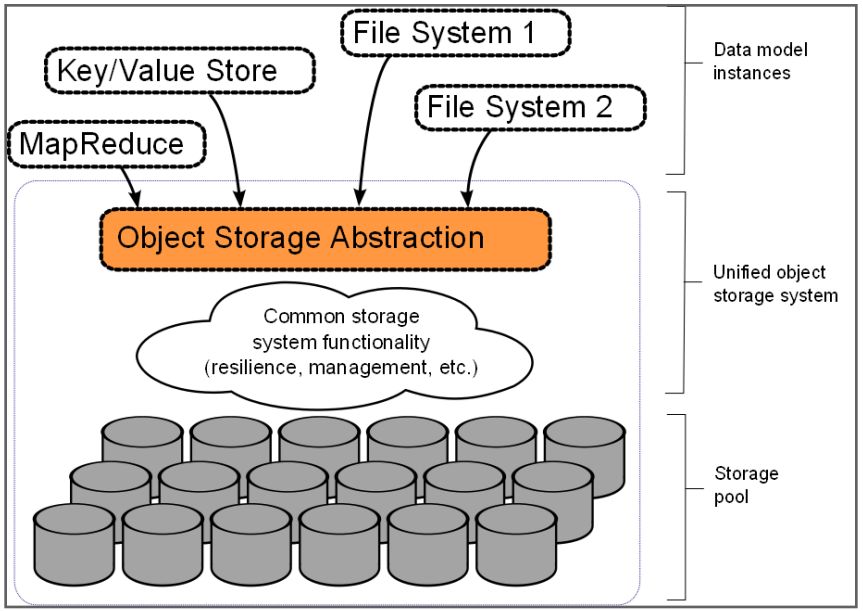
Presenter: Cengiz Karakoyunlu
cengiz.k@uconn.edu
September 27, 2013

What is object-based storage?

- Popular alternative to traditional block-based storage
- Stores and accesses data in ***objects,*** logical collection of bytes with numerical identifiers
- Easy data management
- Decouples storage systems from underlying hardware resources
- Various data models can be built on top of object-based storage
- Typically implemented as a software interface, although featured as a device level interface

Why do we need a new object-storage interface?

- Large scale object-storage systems are generally tailored to specific use cases
- Cannot easily reuse them in different use cases
- Difficult to maintain a common storage pool for different applications
- ***Proposing*** Advanced Storage Group (ASG) interface;
  - Unifies the features necessary to meet the requirements of common data models
  - Provides a foundation for common storage use cases



Common data model requirements

	Shared				Distinguishing				
	High Performance	Scalability	Fault Tolerance	Concurrent Read Access	Concurrent Write Access	Synchronization Primitives	Atomicity	Compute Storage Locality	Record Oriented Access
Parallel File System	✓	✓	✓	✓	✓	✓	✓		
Cloud Object Storage	✓	✓	✓	✓			✓		
MapReduce	✓	✓	✓	✓				✓	
Key/Value Store	✓	✓	✓	✓	✓	✓	✓		✓

Common storage use case (I)

POSIX Directory

- Create , remove, lookup or rename an entry, update metadata of an entry
- Atomic operations
- Existing object-storage systems typically use additional services (metadata servers) to support POSIX directory operations

Common storage use case (II)

Column-Oriented Key/Value Store

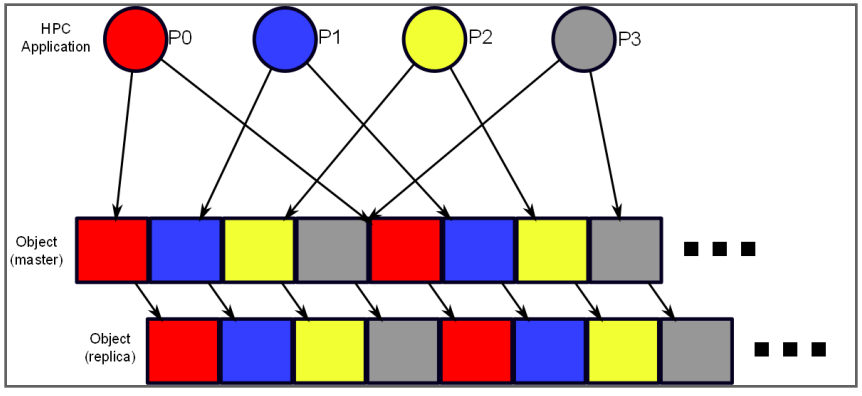
- Each entry is stored in a column
- Each row stores the same data field of an entry
- Shard represents collection of rows

		Column 0	Column 1	Column 2	Column 3
Shard 1	**Row 0**	Alice	Bob	Brad	Charles
	Row 1	Smith			Springfield
Shard 2	**Row 0**	111-1111		144-1144	321-4321

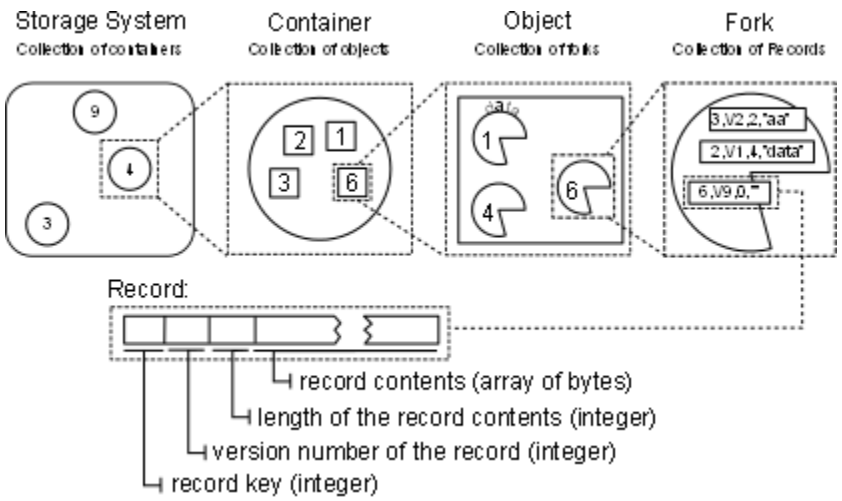
Common storage use case (III)

HPC Application Checkpoint

- HPC applications periodically write checkpoint data
- Existing checkpointing methods
  - N-N
    - Each application writes to a separate checkpoint file
    - Metadata overhead
  - N-1
    - Each application writes to a unique checkpoint file
    - High concurrency

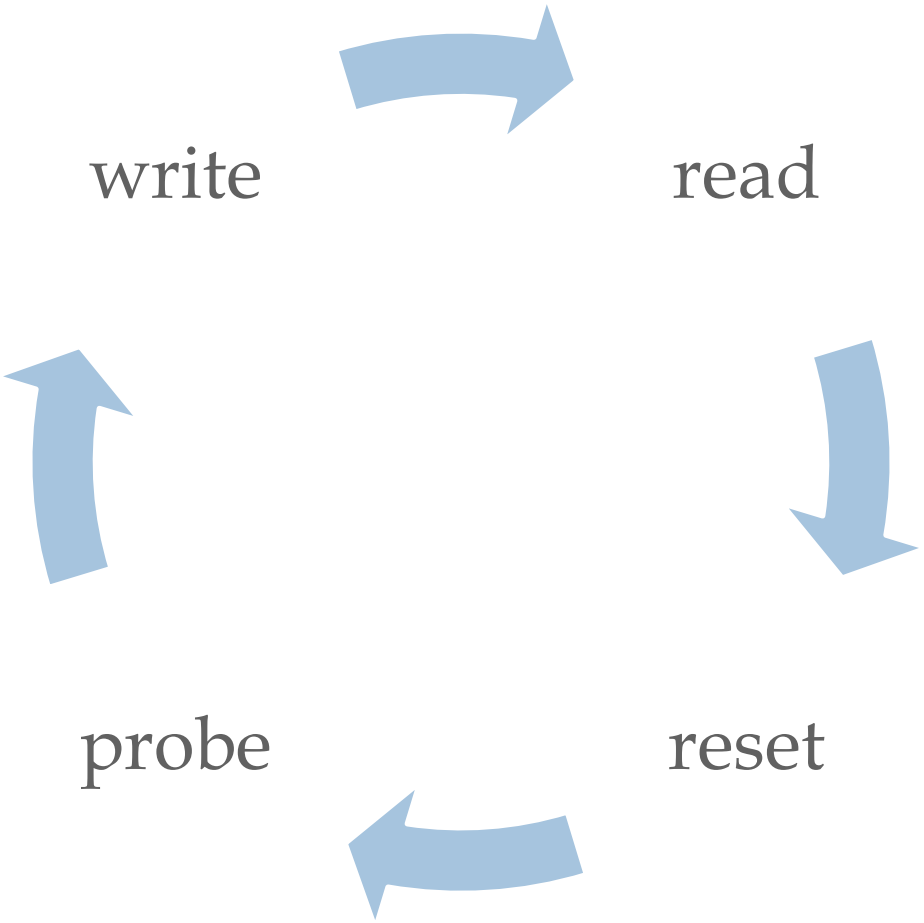


ASG Storage Model Architecture



- Records may contain zero-length data
- Forks allow to store related data together
- Containers partition the system into logical units
- ASG entity identifiers are not global
- 2^{64} records in a fork, 2^{64} forks in an object, 2^{64} objects in a container

ASG Storage Model Primitives



write

- Stores data in a sequential range of records
- Overwrites existing data
- Input arguments
  - Container, object, fork and record ids
  - Local buffer
  - Range of records
  - Number of bytes going to each record
  - Conditional flags
  - Version number
- Returns
  - Size of written data
  - New version number
- Example;
  - write (1, 1, 1, 2, 2, 2, “data”, UNTIL, 3)

Conditional flags for write

- NONE
  - Write should succeed without checking version number or conditional flags
- ALL
  - Write should only succeed if the given version number is greater than all the version numbers in the specified range
- UNTIL
  - Write should **continue** until it finds a record with a version number greater than or equal to the given version number
- AUTO
  - Given version number is not important
  - New data is written with the highest version number in the given range plus one
- Conditional flags can be combined

read

- Retrieves data from a sequential range of records
- Input arguments
  - Container, object, fork and record ids
  - Local buffer
  - Range of records
  - Conditional flags
  - Version number
    - Cannot be used to retrieve older versions
    - Only used for conditional execution
  - Returns
    - Number of records read
    - Version number information
- Example;
  - read (1, 1, 1, 2, 2, local_buffer, UNTIL, 3)

Conditional flags for read

- NONE
  - Read should succeed without checking version number or conditional flags
- ALL
  - Read should only succeed if the given version number is greater than all the version numbers in the specified range
- UNTIL
  - Read should **continue** until it finds a record with a version number greater than or equal to the given version number
- Conditional flags can be combined

reset

- Resets an entity back to its original condition (version 0, no data)
- Can operate on containers, objects, forks and records
- Input arguments
  - Container, object, fork and record ids
  - Range of records may be specified
  - Conditional flags
- Returns
  - Number of entities reset
- Example;
  - reset (1, 1, 1, 2, 2, ALL, 5)

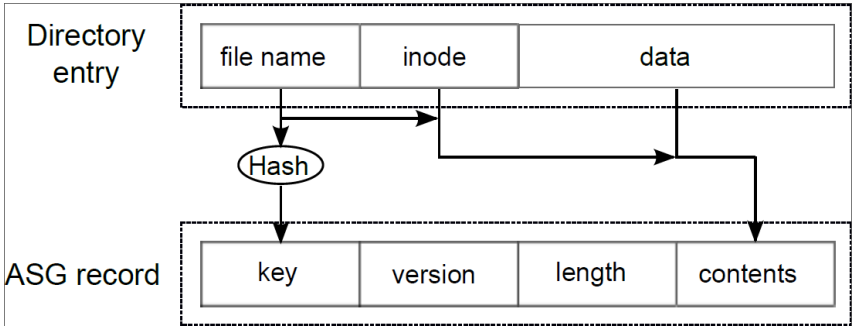
probe

- Returns information about a set of matching items
- Can be called on the entire system, containers, objects or forks
- Input arguments
  - Container, object, fork or record ids
  - Entity id to start with
  - Local buffer to store information
  - Maximum number of items to retrieve
- Returned information contains
  - Id of the first container, object, fork or record
  - Number of containers, objects, forks or records
  - Total number of records
  - Record version numbers
- Example;
  - probe_system(2, local_buffer, 8)

How do we meet common data model requirements?

	Shared				Distinguishing				
	High Performance	Scalability	Fault Tolerance	Concurrent Read Access	Concurrent Write Access	Synchronization Primitives	Atomicity	Compute Storage Locality	Record Oriented Access
Unified byte stream and key&value storage									✓
Eliminating object attributes		✓							
Record versioning	✓				✓				
Conditional operations			✓			✓	✓		
Independently addressable records				✓	✓				✓
Fork structure	✓		✓						
Server location								✓	

How to use ASG for common storage models? (I)



- Directory entries are represented with ASG records
- Independently addressable records and conditional operations prevent duplicate directory entries and ensure atomicity
- While creating a entry ASG *write()* checks for zero version number
- ASG *reset()* checks the version number while removing an entry
- To update the metadata of an entry, ASG *write()* checks for non-zero version number
- While renaming, ASG *write()* does not use conditional flags to overwrite new entry if it already exists
- ASG *probe()* keeps track of existing version numbers to identify entries modified while reading a directory

How to use ASG for common storage models? (II)

- Any value in the database table can be references by an *object-fork-record* triple
- All records within a row are stored in the same object
- All records within a column are stored in the same fork
- An entire row or column can be created or removed atomically
- Without ASG features, and additional mapping index is required to access rows and columns
- Since ASG records can have zero-length data, there can be empty cells in the database

		Column:fork 0	**Column:fork 1**	**Column:fork 2**	**Column:fork 3**
Shard:object 1	**Row:record 0**	Alice	Bob	Brad	Charles
	Row:record 1	Smith			Springfield
Shard:object 2	**Row:record 0**	111-1111		144-1144	321-4321

How to use ASG for common storage models? (III)

- ASG object-fork-record structure and explicit location control feature enable to implement HPC checkpointing methods
- Existing checkpointing methods
  - N-N
    - ASG storage model exposes the location information of any entity to higher-level applications
    - Applications can use the location information to balance the metadata load across the system without talking to an additional server
    - Object attributes are eliminated in the ASG storage model that further simplifies metadata management
  - N-1
    - Conditional operations and versioning are useful to order writes to a shared checkpoint file
    - Applications can concurrently and atomically write to a shared checkpoint file
    - No need to use any locking methods

Related Work

Existing work	Feature
NASD	Variable-length objects replacing fixed-length traditional blocks
OSD+	Adds dedicated directory objects on top of T10
Panasas File System Lustre Ceph	Built on object-based storage
Ursa Minor	Supports versioned writes based on timestamps
Datamods	Extends existing storage system services to support complex data models
OSC's PVFS-OSD	Maps PVFS on top of an object storage emulation
VSAM	Supports both fixed and variable length records
NTFS	Forks are similar to ASG records
Amazon SimpleDB Amazon DynamoDB Redix Hyperdex	Support for conditional operations

Basis for our work	Feature
TOSD	Atomicity, versioning and commutativity
Goodell et al.	Extended POSIX API with data objects
Carns et al.	Optimistic coordination

Conclusion & Future Work

- ASG storage model unifies the features necessary to support common data models
- Common storage use cases can be implemented on top of the ASG storage model using its structures and primitives
- More use cases can be supported using the ASG storage model
  - Fault tolerance
    - Recovery after a system failure can be more straight-forward using the versioning feature of the ASG storage model
- Even more complex storage systems can be built on top of the ASG storage model

Acknowledgments

- We would like to thank Matthew Curry, Geoff Danielson, Ruth Klundt and Justin Wozniak.
- This material is based upon work supported by, or in part by U.S. Department of Energy's Oak Ridge National Laboratory and included the Extreme Scale Systems Center, located at ORNL and funded by the DoD in part by contract number 4000111689 "Novel Software Storage Architectures". This work also was supported by U.S. Department of Energy, under contracts DE-AC02-06CH11357.