In-Vivo Storage System Development

Noah Watkins*, Carlos Maltzahn, Scott Brandt

*UC Santa Cruz, *Inktank, Inc.*

Ian Pye

Cloudflare, Inc.

Adam Manzanares

CSU Chico

In the beginning…

- Storage systems are big dumb boxes with fixed interfaces
- Applications build all their smarts on top, no matter how inconvenient that might be
- Add Diagram
  - Basic illustration of the above concept

And throughout time…

- Alternative I/O interface proposed
  - Co-design
- Add example diagram
  - Structured interface to storage
- This is a powerful, well-researched concept
  - Active storage and custom interfaces enable
  - Reduce data transfer, exploit parallelism, simplify
- But moving a giant storage system (and established users) is hard!

And we settled on middleware…

- New thing? Build some middleware
- Instead of co-design
- Interesting stat is the number of libraries and middleware listed on wikipedia. Lots!

The custom interface comeback

- Hadoop has been popularizing this
  - Customizable platform, structured storage
- DOE FastForward Project
  - Analysis shipping in Lustre
- Heavily used in Ceph products
  - Atomicity guarantees, structured storage
- Open-source systems avoid vendor lock-in
- All the pieces seem to be in place. What gives?
  - *How do we actually build this stuff?*

Observation 1: Data and Interface are One

- The interfaces and data are tied together
- From this it follows that the storage system should play a key role in managing the interfaces

Example of Co-Design

- Click streams, logs, sensor, sci. simulation
- Read-mostly data
- Example diagram
  - Time ordered data partitioned into objects
- Customized interfaces are built on each object
- Both storage system and application must evolve together
- If we change one we need to change the other

Observation 2: Software life-cycle is difficult

- Application source is decoupled from interfaces
- Example: production plus 2 developers
- Eventually merge interfaces into production
- Isolation expected by developers isn't there

Observation 3: Deployment is difficult

- Applications are decoupled from the interfaces they depend on
- Consistency is hard to ensure in a dynamic system

Ok, so what? Get a test cluster

- Avoid production performance surprises
- Conflicts aren't fatal, just use developer guidelines to avoid conflicts
- Stage all the new changes ready to go
- Costs $$$
- Migration to production is shot in the dark
  - Peculiarities of live data

In-Vivo Storage Development

- Single system
- Live evolution
- System manages interfaces and ensures isolation
- Facilitates software life-cycle

Architecture

- Dynamic, extensible interfaces
- Interface developer environment
  - Workspace
  - Isolation
- An IDE service
  - Handles interface consistency
  - Etc…

Extensible Storage Interfaces

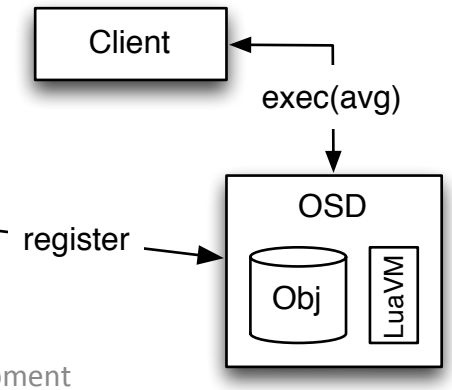
- Our focus is on object-based storage
- Interface defined by new system function
  - Capabilities depend on the system
    - Object model, atomicity
- New functionality is added with new code
  - Pragmatically, static interfaces won't help
  - Compiled extensions are difficult to manage
  - Static interfaces undermine iterative development

Dynamic Storage Interfaces

- Script-based solution dynamic / fast enough
  - Just shuffling data around
- Our prototype in RADOS uses Lua language
  - 90% the speed of C
- New interfaces are small code fragments

Interface-Average

```
function avg(attr)
    key = "avg." + attr
    val = cache.get(key)
    if not val then
        val = ComputeAverage(attr)
        cache.put(key, val)
    done
    return val
end
```

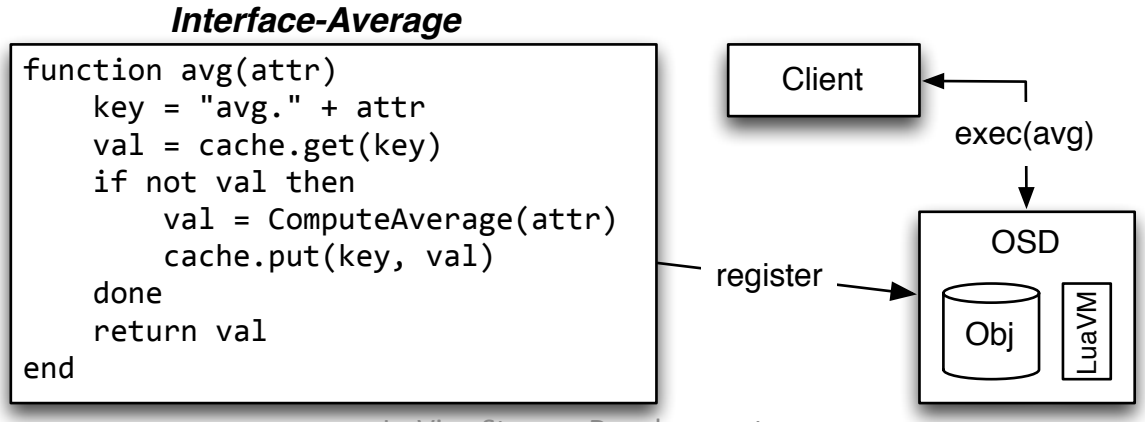


Interface Development Environment

- *Workspace* is the unit of developer isolation
  - Like a working copy in Git/Subversion
  - Exists in, and is managed by, the storage system
- Diagram
  - Storage cluster with workspace existing orthogonally to native partitioning entities like a pool.

Workspace Isolation

- Logical Isolation
  - Between workspaces and production views
  - Must be transparent and efficient
- Interfaces may cache data, use indexes, etc…
  - Transparent namespacing provides isolation



Workspace Isolation

- Efficiency
  - Reads satisfied from base data
  - CoW for data transformations
- Physical Isolation
  - Cluster partitioning
  - Data placement
  - Tiering
  - Integrate with underlying mechanisms (e.g.Pool)

Workspace Isolation

- Performance Isolation
  - Production performance should be insulated
  - Inter-workspace performance policies
- Use existing solutions
  - Disk (Fahrrad)
  - CPU (RBED)

Workspace Management

- Dropped or merged with production
- Name collisions are identified
  - Resolution is not automatic, but managed
- Isolation parameters can also be migrated
- Expensive transformations take care
  - May want to migrate all interface to format
  - Handled automatically using migration routine
- Workspace removal results in clean-up

The IDE Service

- Interfaces change in a changing cluster
  - Propogation etc…
- Application should expect consistent views
- Existing services handle data with similar requirements
  - Paxos service managing cluster state
  - Distributes and ensures interface consistency

The IDE Service

- Integration
- Need to resolve interfaces in the storage system with applications

Conclusion