In-Vivo Storage System Development

Noah Watkins*, Carlos Maltzahn, Scott Brandt

*UC Santa Cruz, *Inktank, Inc.*

Ian Pye

Cloudflare, Inc.

Adam Manzanares

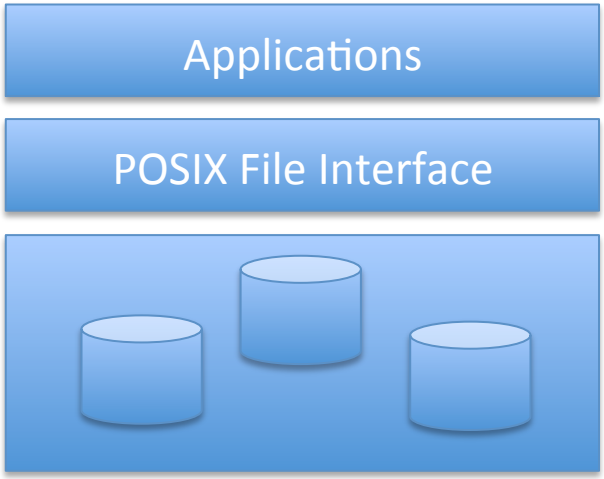
CSU Chico

Overview

- Co-designed storage interfaces
  - History
  - Benefits
  - Challenges
- Live evolution of interfaces
- System support

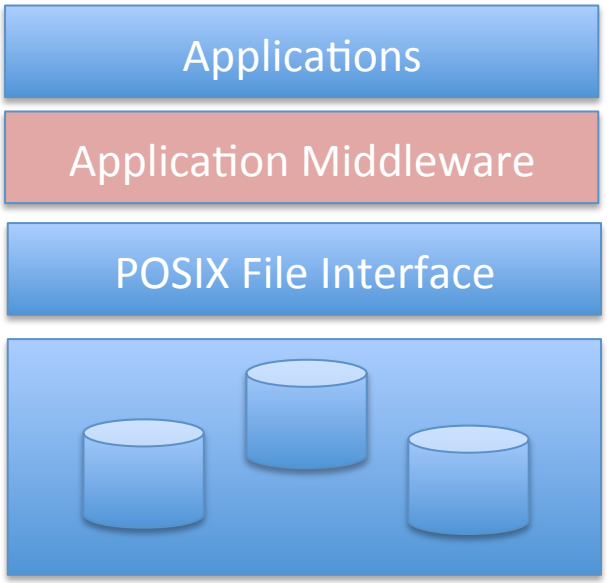
In the beginning…

- Storage systems are *treated* like big dumb boxes with fixed interfaces (POSIX file I/O)
- Application developers accept inconveniences



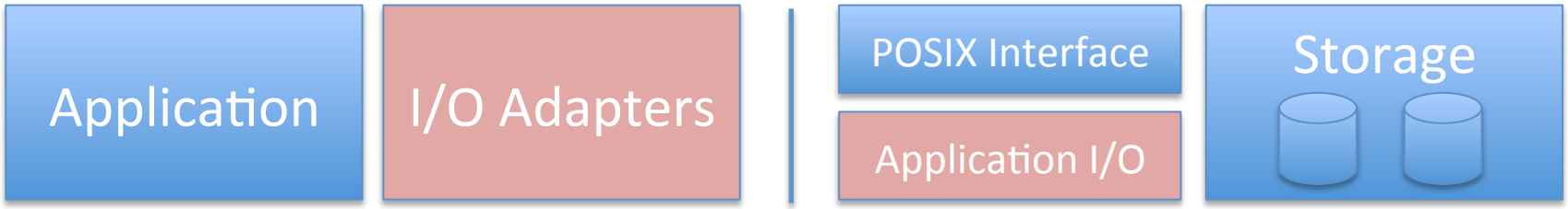
Middleware to the rescue…

- Applications extract I/O specific interfaces
- Share domain-specific access libraries



And throughout time…

- Proposed alternative / co-designed interfaces



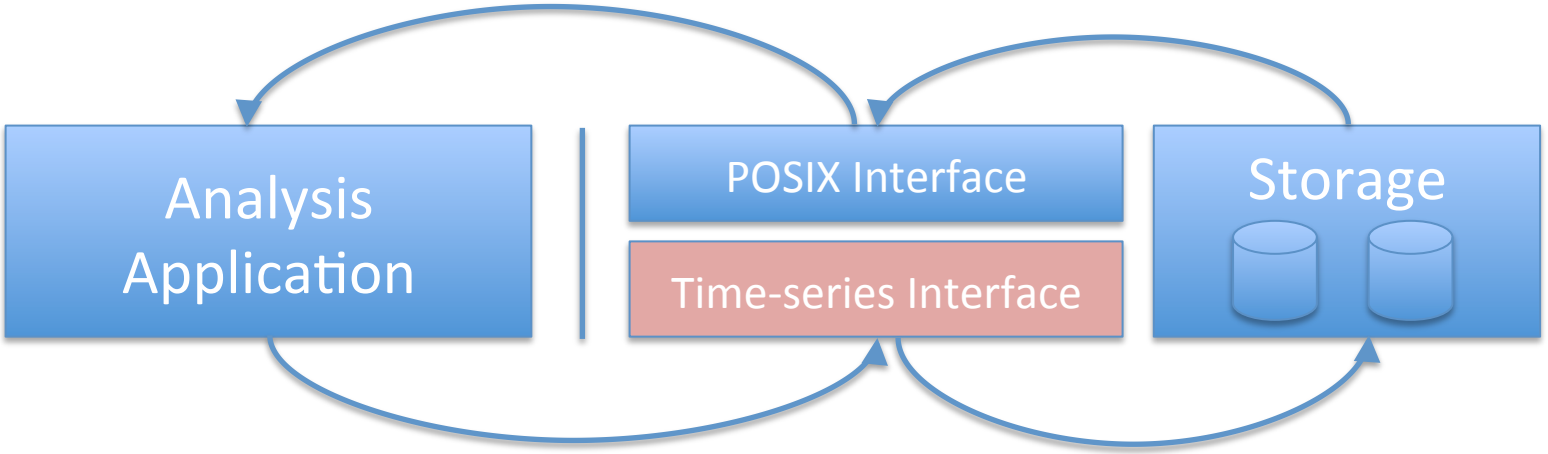
- Powerful, well-researched concept
  - Simplify design and increase understandability
  - Active storage-related work for the past 15 years
- Storage systems have huge inertial mass

The custom interface comeback

- Hadoop has been popularizing this
  - Structured storage, co-located processing
- DOE FastForward Project
  - Versioned object store, analysis shipping for 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?*

App/Storage Co-design Example

- Data source partitioned and stored
  - Time-series: access log, sensor, simulation output
- Storage system exports a time-series interface
- Co-design takes a holistic view
  - Each layer is tailored for the application



Obs. 1: Data and Interface are One

- Data and interfaces are inextricably tied
  - Interfaces define and interpret stored data
  - Evolution of each happens together
- Co-designed interfaces exist within storage
- Apps and interfaces must be synchronized
  - Complex, evolving storage system
- *Storage systems should play a role in the management of co-designed interfaces*

Obs. 2: Complicated software life-cycle

- Decoupled application and interface source
  - Evolve together under different control domains
- Interface conflicts arise
  - Multiple developers plus production versions
- Need isolated development environments
- *Storage system should provides isolation and life cycle management*

Ok, so what? Get a test cluster…

- Development conflicts aren't fatal
  - Use guidelines and best-practices
- Test performance before deployment
- Stage application and interface versions
- Problems
  - Over-provisioning costs $$$
  - Peculiarities of live data
  - Iterative development is difficult

In-Vivo Storage Development

- Interface development in a single system
- Evolution of the live system
- Management services
  - Interfaces
  - Ensure isolation
  - Facilitate software life-cycle

System Architecture

- Dynamic, extensible interfaces
- Interface developer environment (IDE)
  - Isolated workspaces
  - Software life cycle
- The IDE management service
  - Handles interface consistency
  - Integration

Extensible Storage Interfaces

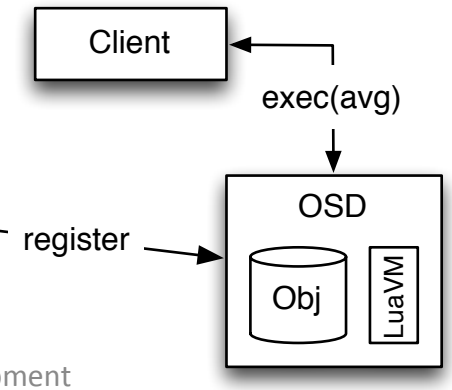
- Our focus is on object-based storage
- Interface defined by new system function
  - Capabilities depend on the system
    - Object model, atomicity
- New functionality is added with new code
  - Pragmatically, static interfaces won't help
  - Compiled extensions are difficult to manage
  - Static interfaces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

Dynamic Storage Interfaces

- Script-based solution dynamic
- Our prototype in RADOS uses Lua language
  - Built several interfaces: just shuffling data around
  - 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
  - Goal: like a working copy in Git/Subversion
  - Exists in, and is managed by, the storage system
- Logical isolation
  - Between workspaces and production views
  - Must be transparent and efficient

Workspace Isolation: Logical

- Interface conflict challenges
- Cached data naming conflicts (e.g. Avg())
  - Dev1: mean
  - Dev2: median
  - Both with cached value named ‘avg’

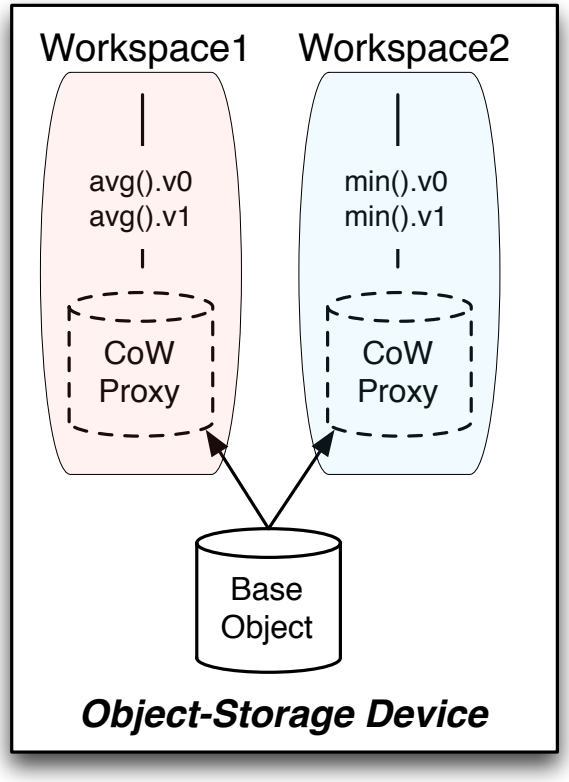
Interface-Average

```
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
```

- Data format transformation
  - Row vs. column storage

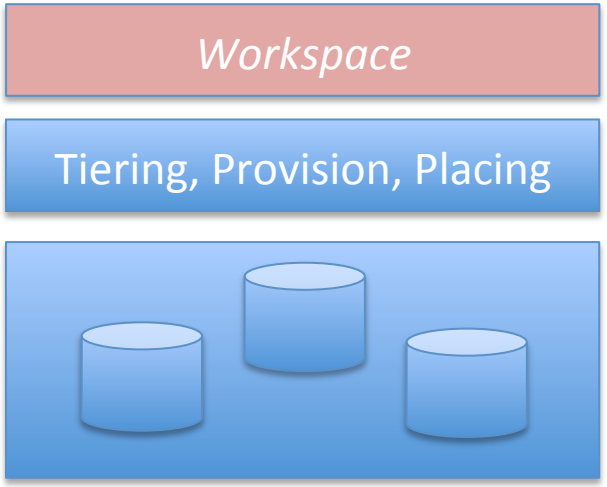
Workspace Isolation: Logical

- Transparent isolation for developers
  - Namespacing techniques
- Read Efficiency
  - Share when possible
  - Satisfied from base data
- Handling writes
  - CoW for data transformations



Workspace Isolation: Physical

- Existing provisioning for developers
- Integrate workspace with existing management abstractions
- Dependent on underlying storage system features

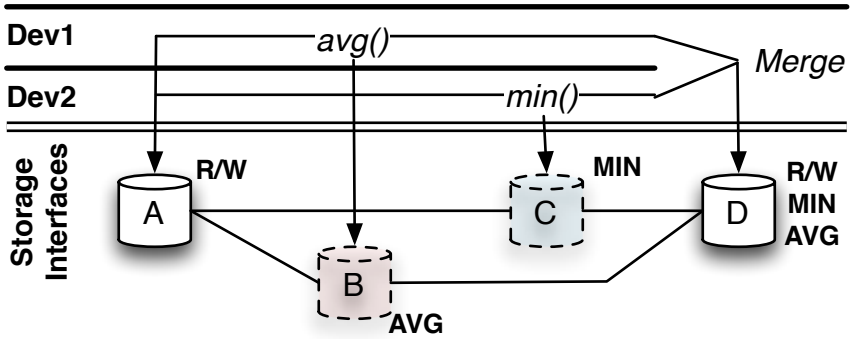


Workspace Isolation: Performance

- Production performance is high-value
  - Avoid violation of SLAs
  - Slow downs unacceptable
- Workspace-aware performance isolation
  - Insulate production
  - Inter-workspace performance policies
- Use existing solutions
  - Disk (Fahrrad, *Povzner, et. al, EuroSys '09*)
  - CPU (RBED, *Lin, et. al, RTSS '06*)

Workspace Management

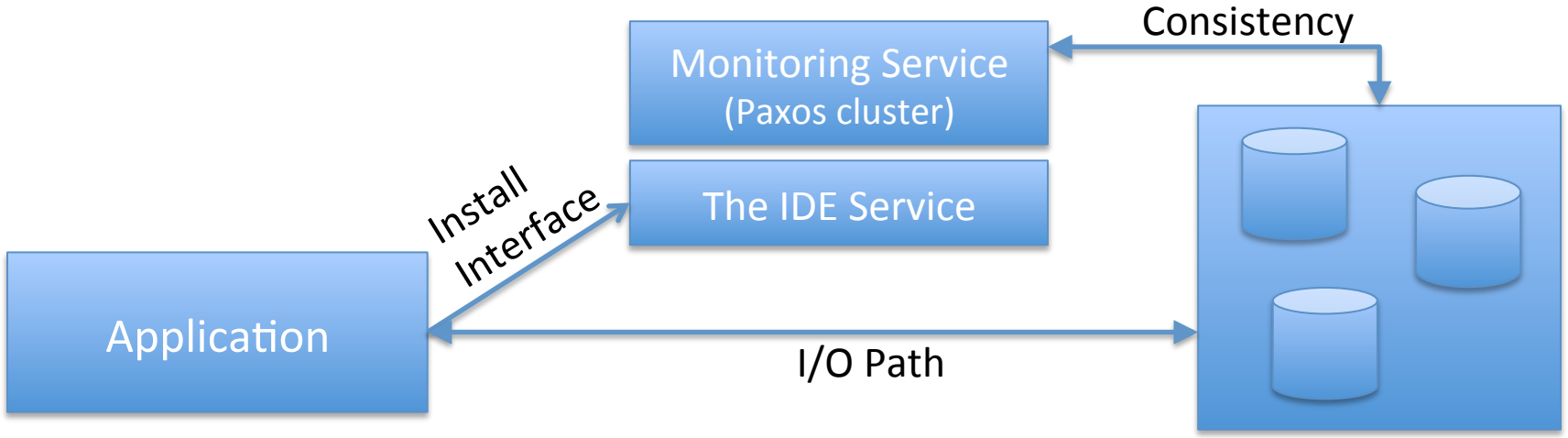
- Familiar interface management
  - Branch-and-merge workflow



- Isolation parameters can also be migrated
- Expensive transformations take care
  - Automatic transformation using hooks
- Workspace removal results in clean-up

The IDE Service: Consistency

- Interfaces are installed, updated, removed
- Large storage system loosely coupled
- Application should expect consistent views
  - Challenging in a dynamic, changing system



The IDE Service: Integration

- Interface discovery and app integration
- Similar challenge in RPC
- Expose a limited SCM interface (e.g. Git)
- Aligned with workspace API
  - Easy to embed in existing projects
  - Integrate with automated builds

Conclusion

- Storage system interface co-design is possible
  - System support: developer isolation, consistency
- Open-source, built inside Ceph
  - Fully scriptable, atomic object interfaces with Lua
  - http://github.com/ceph/cls-lua
- In-progress
  - Consistent interface versions across cluster
- Thanks!
  - Noah Watkins, jayhawk@cs.ucsc.edu