DataWarp Transparent Cache: Data Path Implementation

CUG 2018

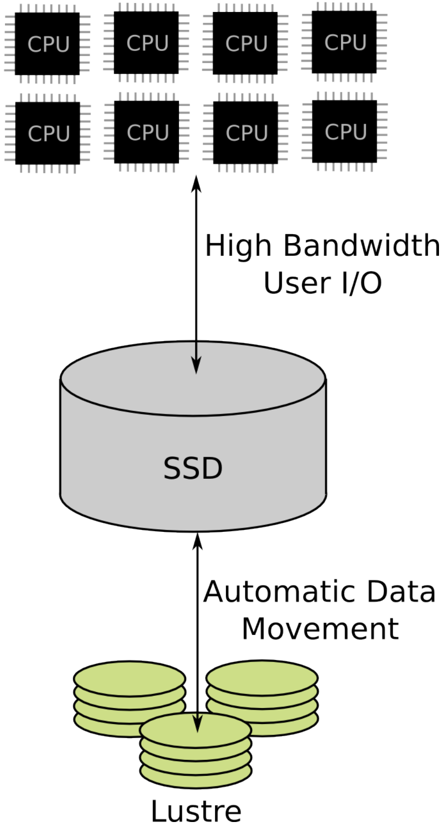
Matt Richerson, Cray Inc.

Agenda

- Overview of DataWarp
- Implementation of the DataWarp scratch data path
- Expanding DataWarp for the transparent cache data path

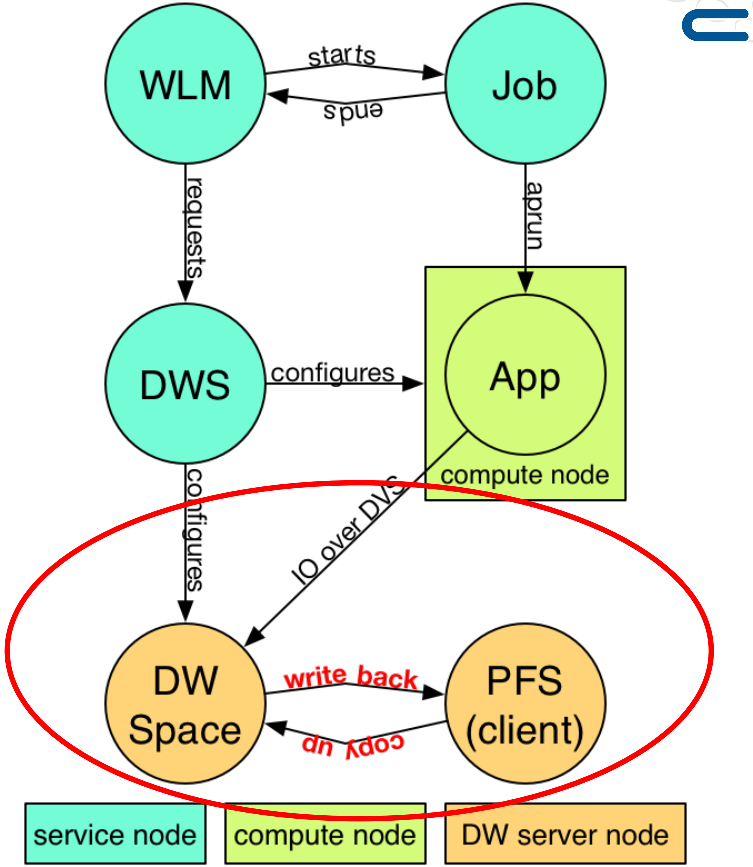
Transparent Cache Overview

- SSDs on service nodes serve as a cache layer between compute nodes and PFS
- DataWarp automatically moves data between SSDs and PFS



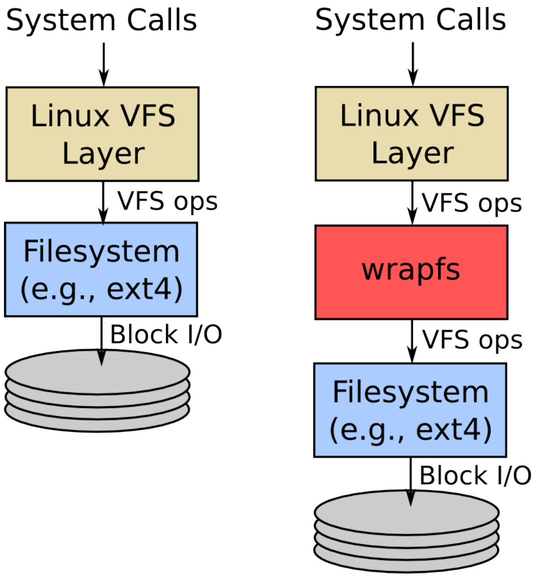
DataWarp Components

- DataWarp has both user space and kernel space components
- Focus is on the kernel level components
- Cray Inc. developed filesystems



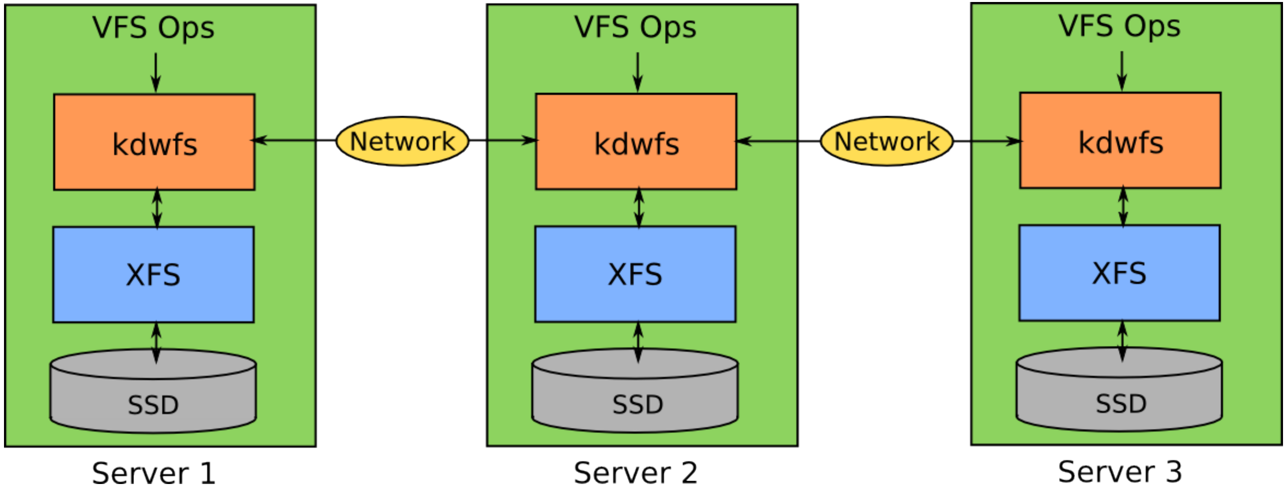
Stackable Filesystems

- **Linux VFS layer allows filesystems to be stacked**
- **VFS operations are the API**
  - Stackable filesystem must appear as a normal filesystem to kernel VFS
  - Stackable filesystem must appear as the kernel VFS to lower filesystem
- **wrapfs is a GPL pass through stackable filesystem**



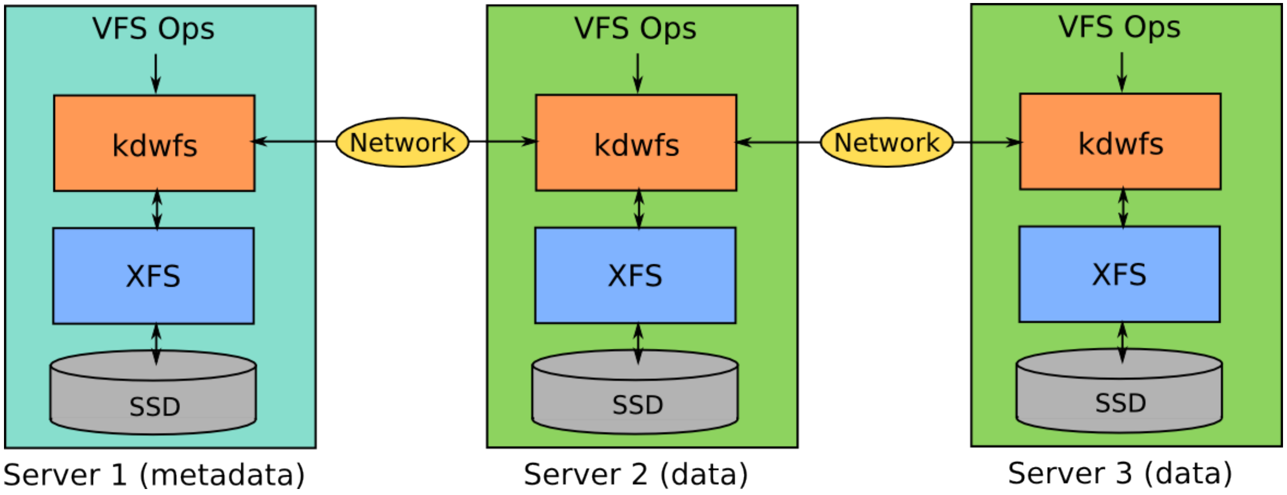
DataWarp Scratch Filesystem

- kdwfs – simple distributed filesystem
- Based on wrapfs



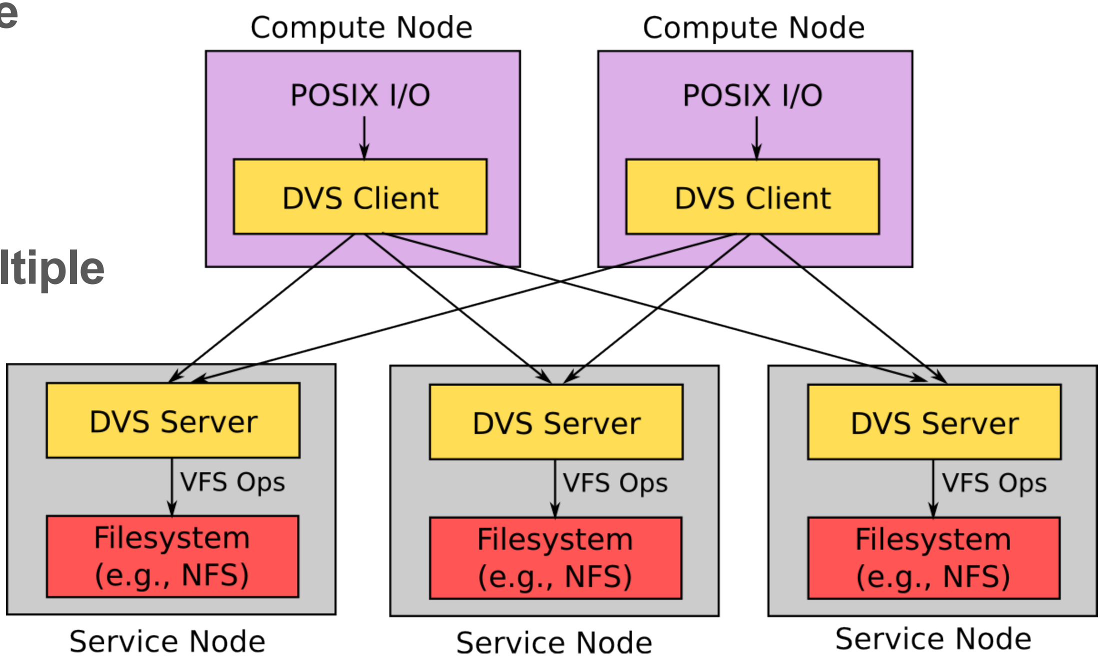
DataWarp Scratch Filesystem (cont.)

- Metadata and data are separated
- Single metadata server
- Data is striped across multiple SSDs



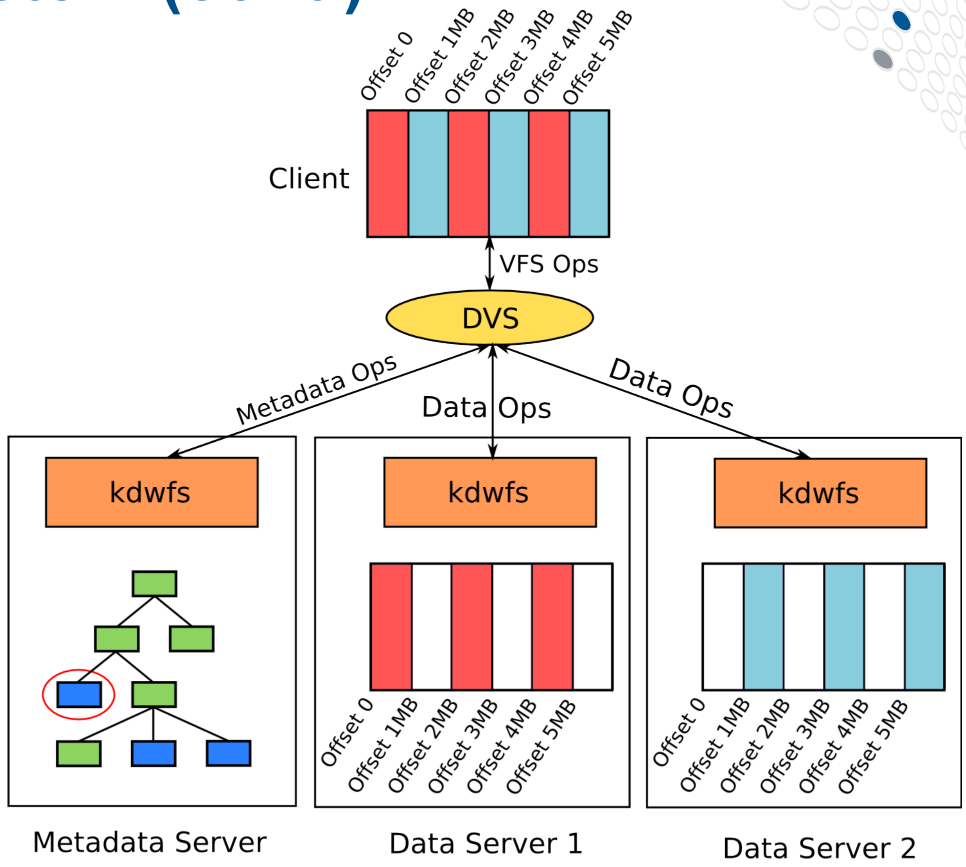
DataWarp Scratch Filesystem (DVS)

- DVS is an I/O forwarder
- POSIX filesystem interface
  - Some limitations
- DVS servers interact with underlying filesystem
- Data can be striped to multiple servers
- Scalable



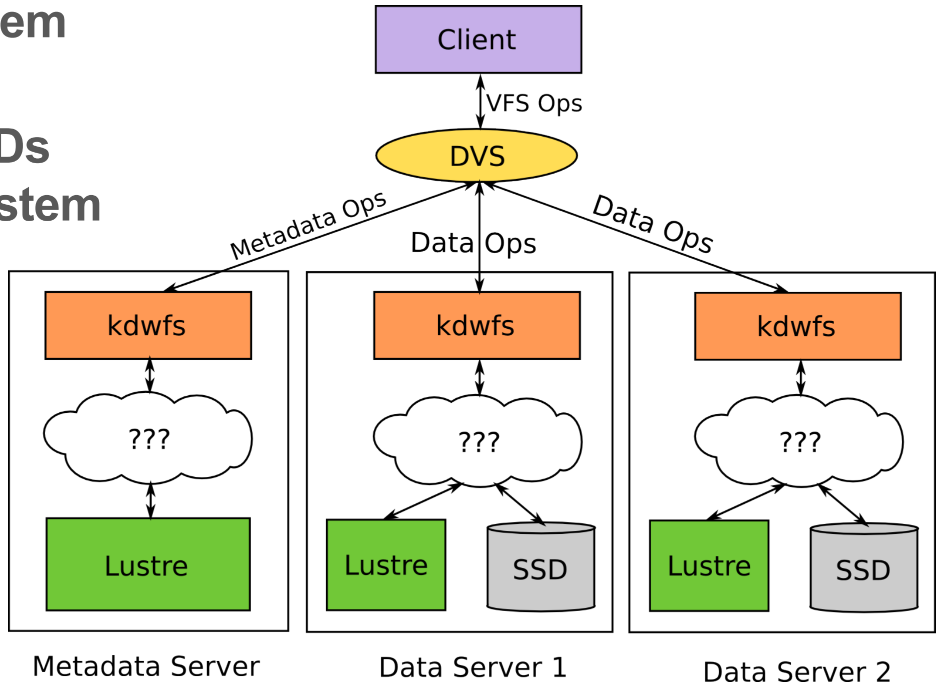
DataWarp Scratch Filesystem (cont.)

- **Each metadata inode has one or more data objects**
- **DVS handles I/O forwarding between computes and DW servers**
  - Metadata and data operations target correct server
  - Data is striped based on set block size
- **kdwfs handles communication between DW servers**



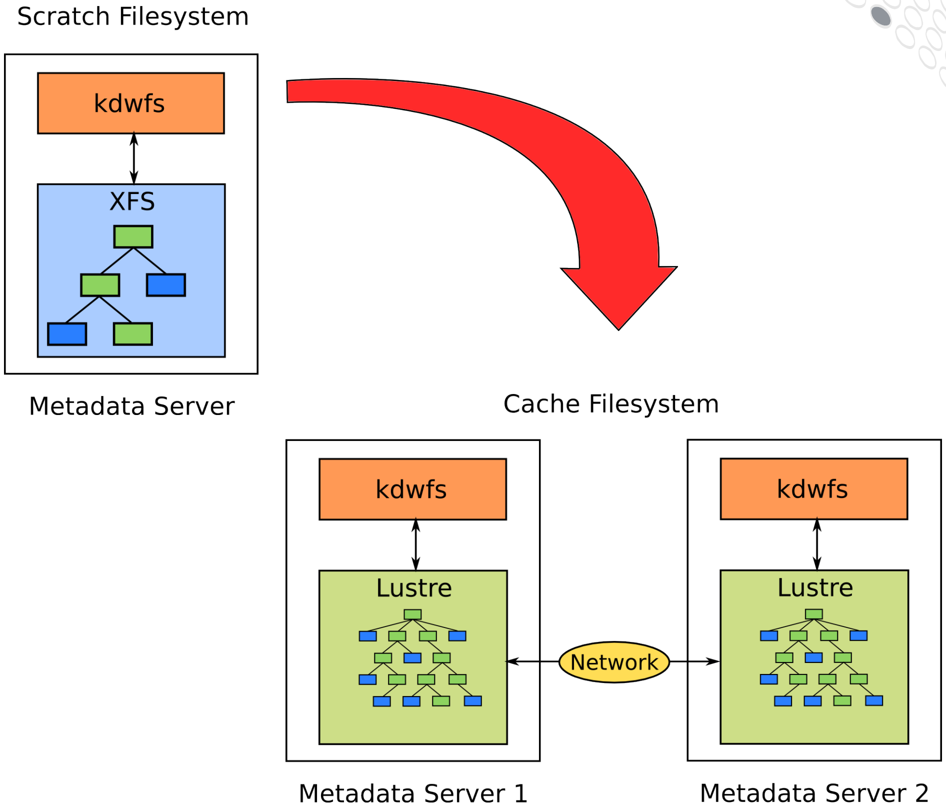
DataWarp Cache Filesystem

- Extension of the scratch filesystem infrastructure
- Use DVS and kdwfs to stitch SSDs together into a distributed filesystem
- How does Lustre get tied in?



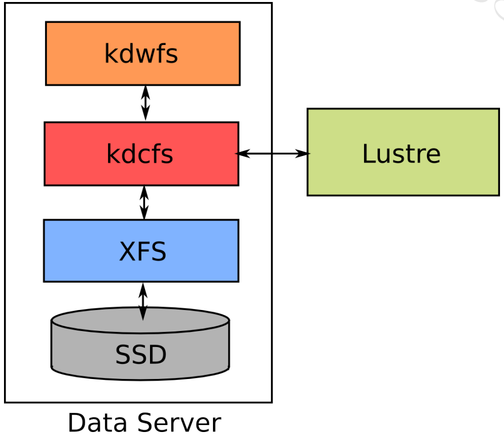
DataWarp Cache Filesystem (metadata)

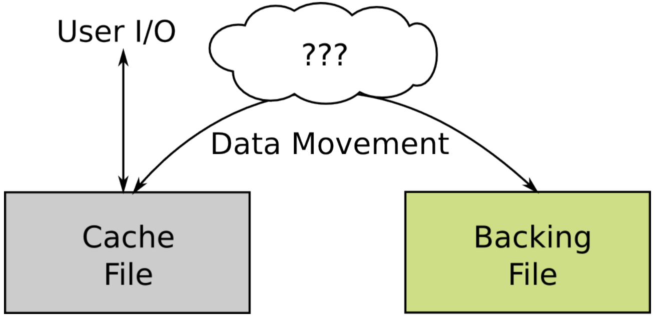
- **kdwfs on metadata server is stacked on top of PFS (Lustre)**
  - PFS client is DW server
  - DW metadata directory tree is identical to PFS
- **PFS keeps coherency between clients**
  - Multiple DW metadata servers are possible
  - DW metadata servers have to handle remote changes



DataWarp Cache Filesystem (data)

- **Data is still striped to multiple data objects**
- **Data objects read and write to SSD**
- **DW manages data movement**
  - SSD holds cache file
  - PFS holds backing file
- **kdcfs (kernel data caching filesystem)**
  - Based on wrapfs
  - Node local
  - File handles
  - I/O targets cache file



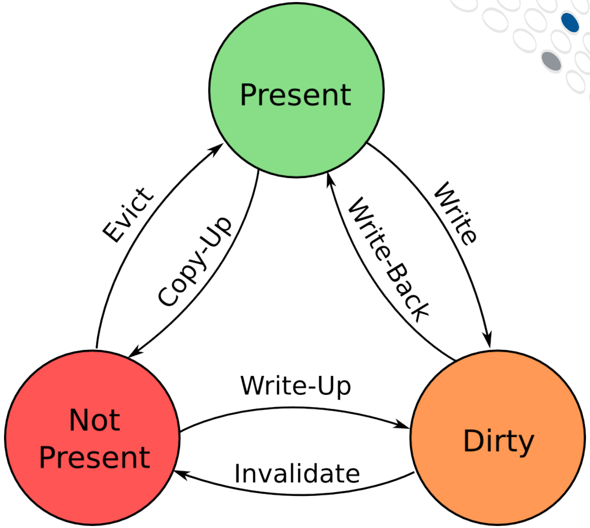


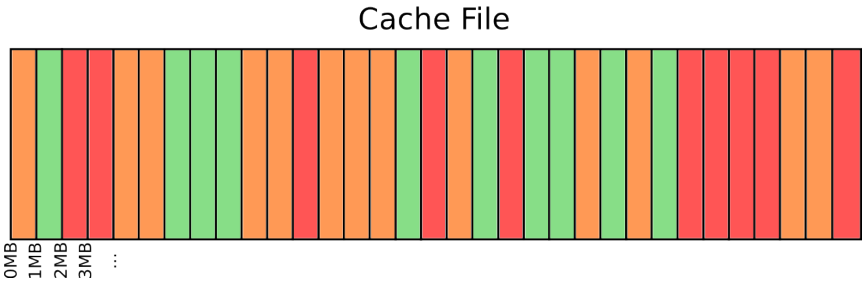
kdcfs Internals (cache operations)

- **Cache operations modify cache file data**
  - Implemented as a pool of worker threads
- **Cache operations: copy-up, write-back, evict, invalidate**
  - Copy-up – Copy data from PFS to cache file
  - Write-back – Copy data from cache file to PFS
  - Evict – Deallocate a clean region from the cache file
  - Invalidate – Deallocate a dirty or clean region from the cache file
- **fallocate() is used to allocate and deallocate space in the cache file**

kdcfs Internals (extents)

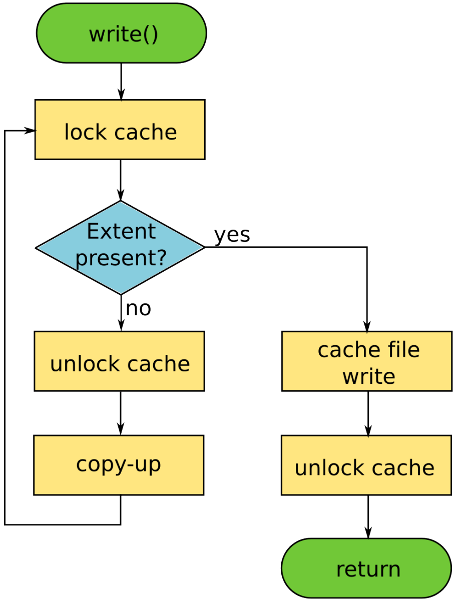
- **Cache file is logically divided into extents**
  - Default size is 1MB
- **Data in each extent is handled independently**
- **Extent states:**
  - Not-present – Cache data is older than PFS data
  - Present – Cache data is the same as PFS data
  - Dirty – Cache data is newer than PFS data
- **Per inode extent states are tracked in an in-memory tree**

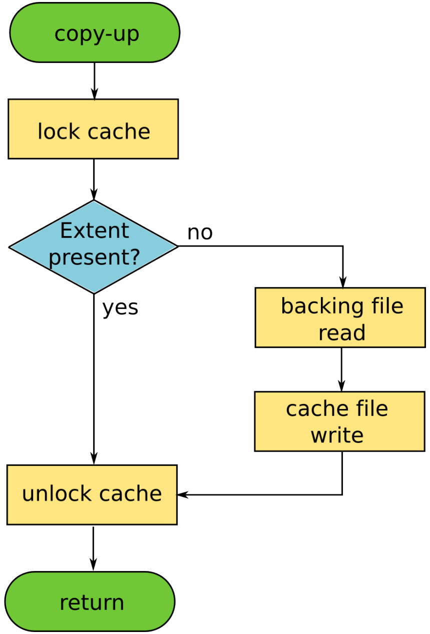




kdcfs Internals (example)

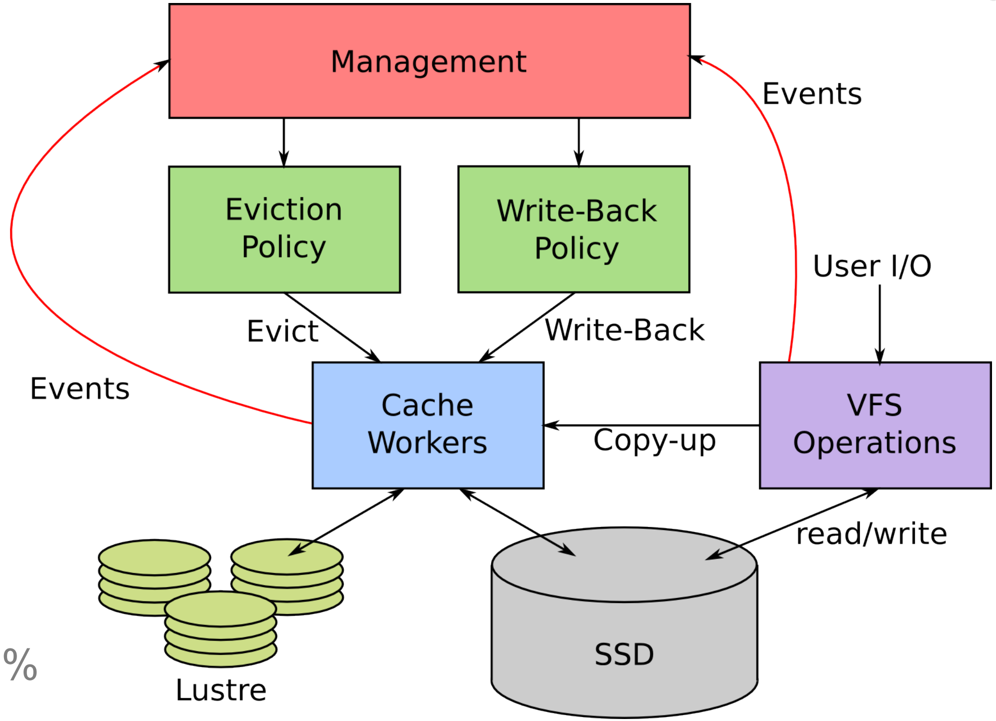
- **VFS operations can trigger cache work**
  - Example: write() results in copy-up





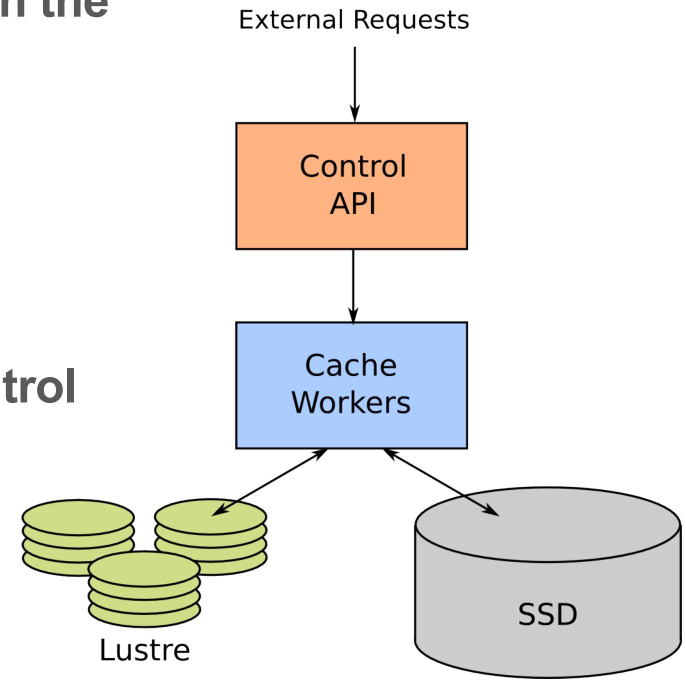
kdcfs Internals (management)

- **Management sub-component monitors cache events**
- **Management policies for write-back and evict**
  - Policies are swappable
  - Separate for write-back and eviction
- **File LRU policy for write-back**
  - High water 50% low water 0%
- **File LRU policy for eviction**
  - High water 100% low water 95%



kdcfs Internals (control API)

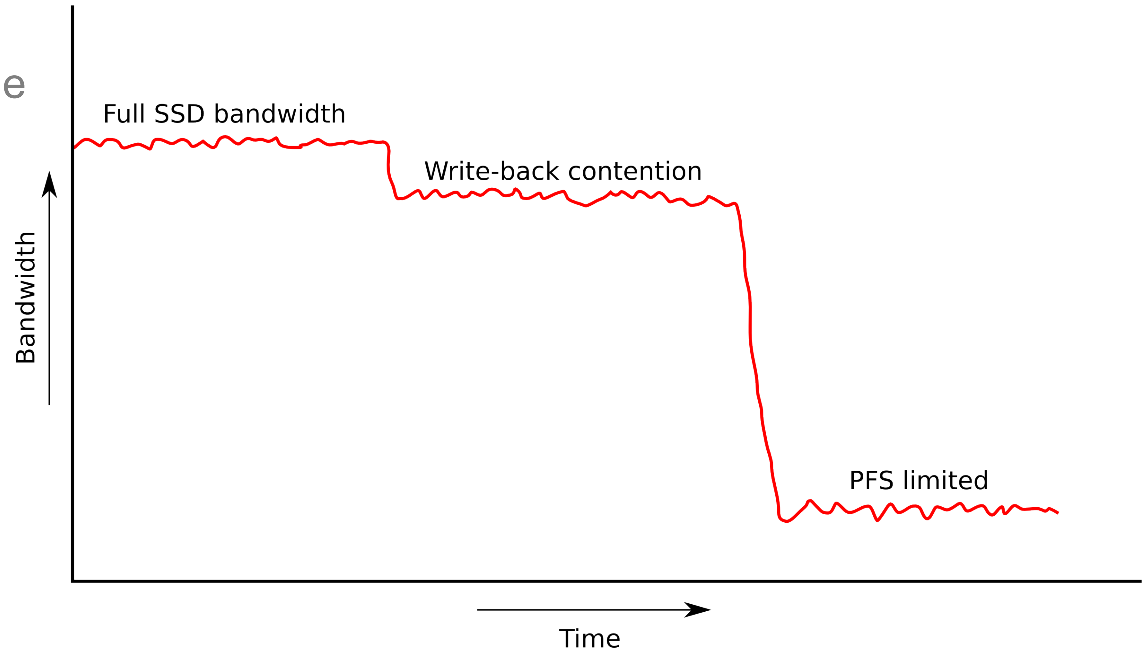
- **External components can influence data in the cache**
- **Ioctl() interface**
  - Individual files
  - Mount point
- **DWS flushes data at job end**
- **Exported to compute nodes as cache control API (future release)**



Transparent Cache Performance

- **Cache size is important**
  - PFS interaction is slow
  - Files larger than the cache are limited by PFS
- **Good workloads**
  - Bursty writes
  - Read after write
  - Multiple reads

Relative Write Bandwidth to Empty Cache



Summary

- **DataWarp transparent cache builds on existing scratch filesystem**
  - Scratch already has good performance
  - Increased stability
  - Fixes and features benefit both modes
- **Metadata operations go through Lustre**
  - All files within the mount are accessible
- **Filesystem implementation allows easy integration of different caching policies**
- **Cache control API will allow manual data movement similar to scratch**

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q&A

Matt Richerson
mattr@cray.com