Efficient, Modular Metadata Management with Loris

Richard van Heuven van Staereling
Raja Appuswamy
David C. van Moolenbroek
Andrew S. Tanenbaum

Vrije Universiteit, Amsterdam

July 29, 2011

File systems as lightweight data stores

- File systems have remained data agnostic for several decades
  - Files are still unstructured sequence of bytes
  - Simple hierarchy-based organization of files
- Generality has enabled widespread adoption as:
  - Document stores in personal computing
  - Dedicated data and metadata stores in enterprise computing
  - Local node stores for cluster/parallel file systems in HPC
  - Local node stores for distributed file systems in DISC

Domain-specific metadata management: a growing trend

- The "Generalized FS – domain-specific metadata" gap
  - User-level metadata management systems bridge the gap
- Desktop and multimedia search applications (Personal computing)
  - Maintain application-specific indices
  - Provide attribute or tag-based query interface
- Enterprise search appliances (Enterprise computing)
  - Periodic, incremental crawling of metadata
  - Admin-friendly interface to assist in policy enforcement

Domain-specific metadata management (2)

- User-level provenance management subsystems (HPC)
  - Low impact, complete, automated provenance gathering
  - Provenance-friendly storage and query runtime subsystems
- Custom-built databases for housing metadata (DISC)
  - Databases optimized for metadata storage and retrieval
  - Avoid using inefficient local file systems as metadata stores

Domain-specific metadata management (2)

- User-level provenance management subsystems (HPC)
  - Low impact, complete, automated provenance gathering
  - Provenance-friendly storage and query runtime subsystems
- Custom-built databases for housing metadata (DISC)
  - Databases optimized for metadata storage and retrieval
  - Avoid using inefficient local file systems as metadata stores

Domain-specific metadata management:
a least common denominator functionality across application areas

Issues with existing metadata management solutions

- Stale query results
  - Outside mainline metadata modification path
  - Indices not maintained in real time
- Performance impact of file system crawling
  - Unoptimized metadata placement in local file systems
  - Resource-intensive index scans and updates
- Storage inefficiency
  - Unwarranted metadata duplication

File systems and metadata management

- If local file systems provide metadata management:
  - No polling/gathering will be required
  - No metadata duplication
  - Custom layout schemes for storing indexed metadata
- However, traditional file systems lack modularity
  - Integration of metadata management on a case-by-case basis
  - Impossible to plug in domain-specific naming systems

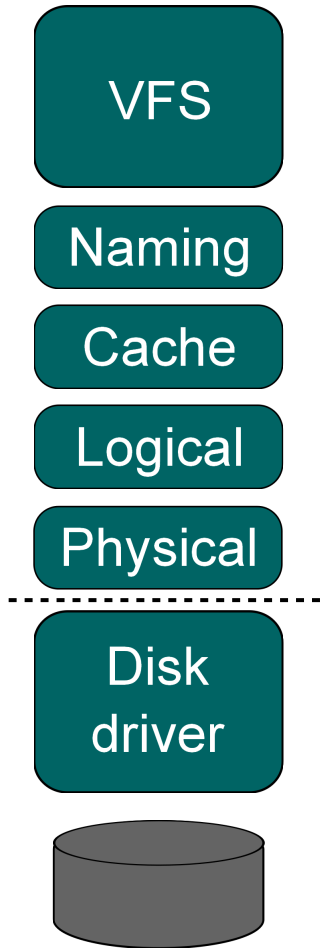
Context: the Loris Storage Stack

- Traditional stack also suffers from several other issues
  - Silent data corruption, RAID write hole
  - Lack of support for graceful degradation
  - Complicated device administration
  - Lack of support for integration of heterogeneous devices
- In prior work, we presented Loris
  - A modular redesign of the traditional storage stack

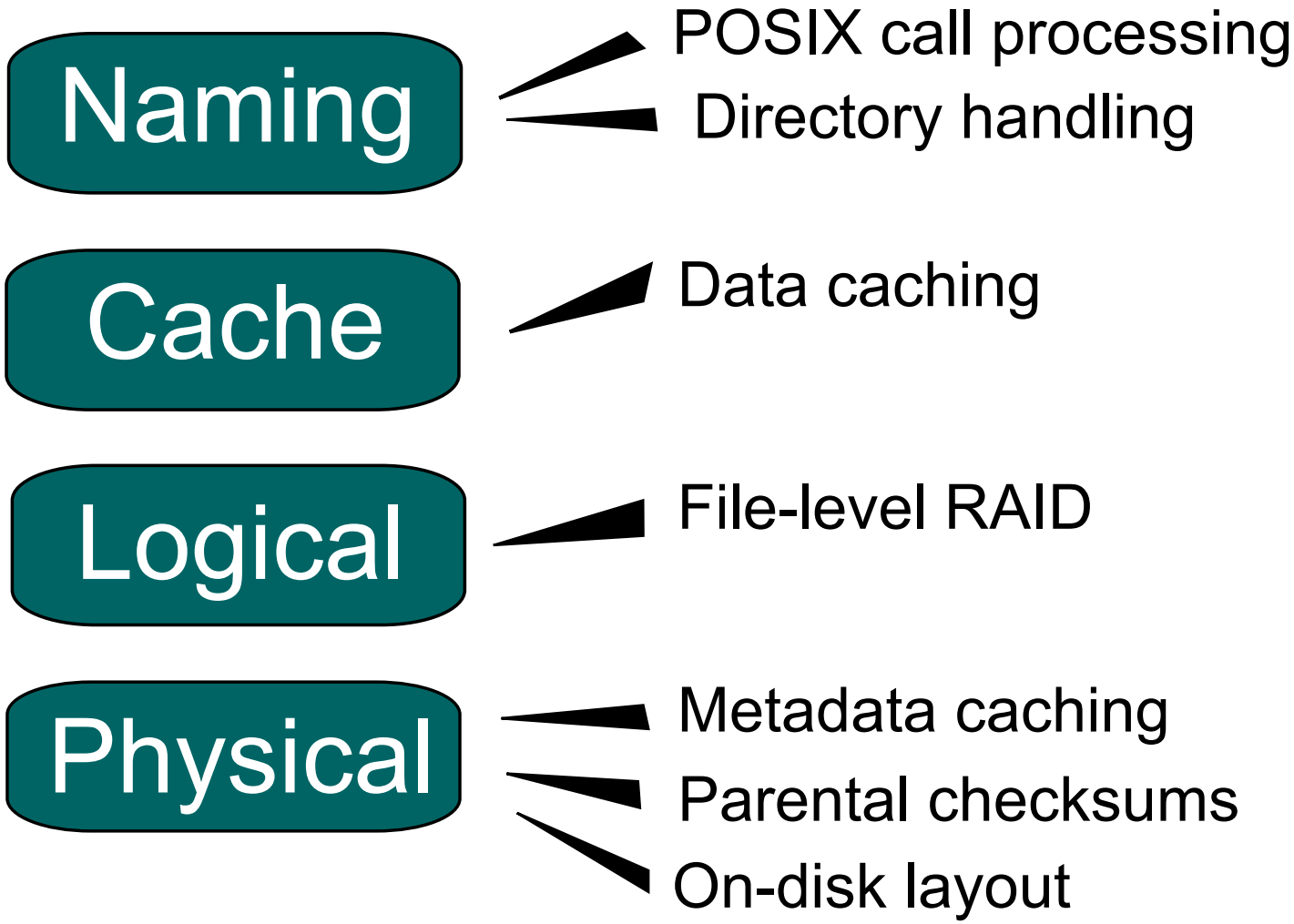
The Loris Storage Stack: layers and interfaces

- File-based interface between layers
  - Each file has a unique file identifier
  - Each file has a set of attributes
- File-oriented requests:

create	**truncate**
delete	**getattr**
read	**setattr**
write	**sync**



Loris: division of labor

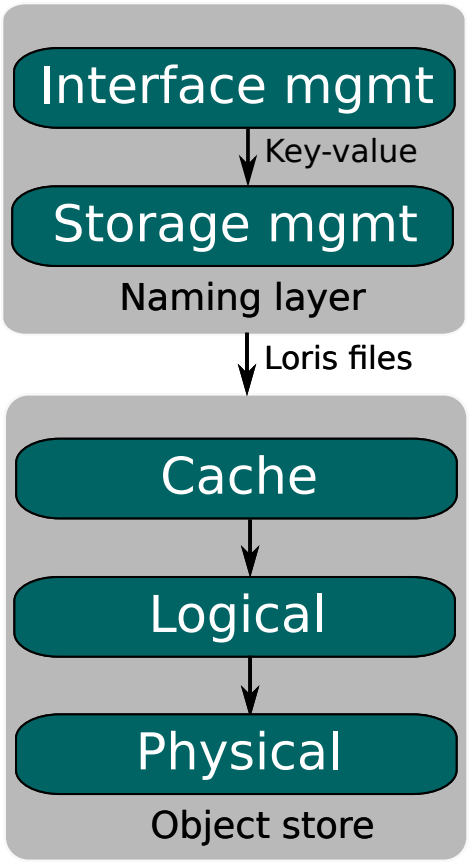


Loris as a customizable metadata management framework

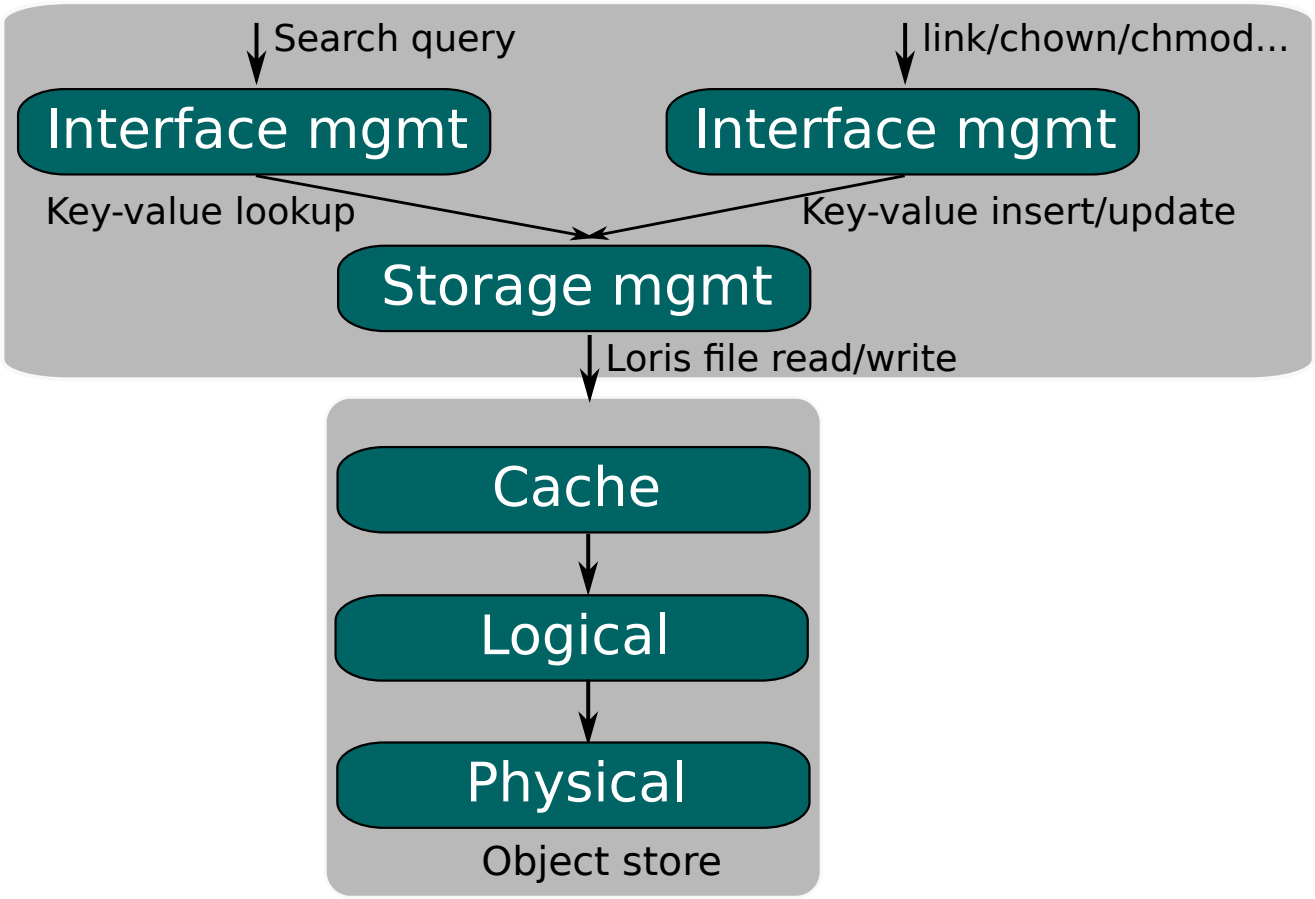
- Loris' naming layer views the lower layers as an object store
  - User-level metadata solutions view FS as object store
  - Metadata management is a straightforward extension
- Modular integration of metadata management
  - Can change naming modules without affecting other layers
- Each naming implementation in essence builds a database
  - Database files stored as Loris files
  - Domain-specific file formats used for packing metadata
  - Domain-specific query interfaces used for searching metadata

Our Loris-based metadata management solution

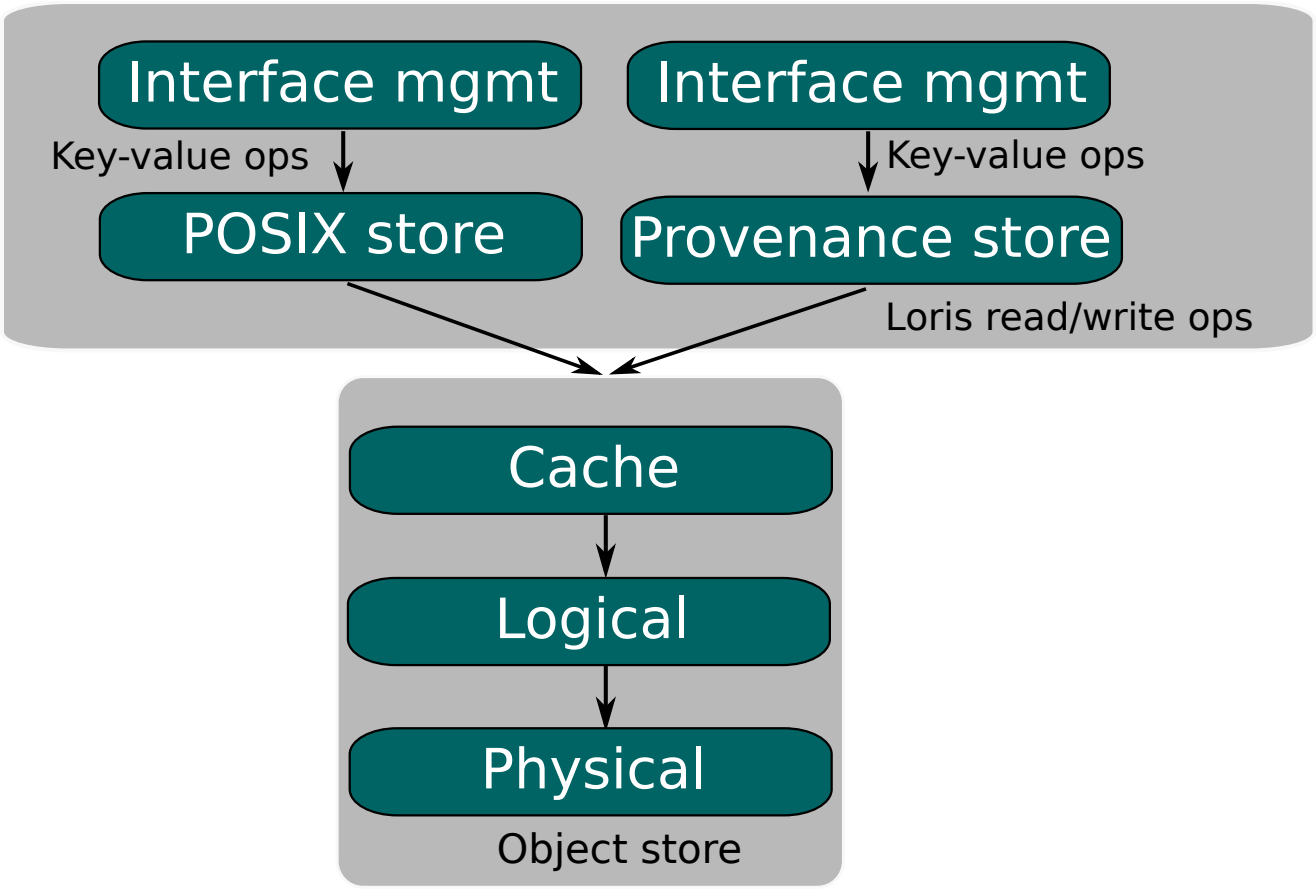
- Plug-in-based naming layer
  - Decomposed into two sublayers
- Storage management sublayer
  - Key-value store for metadata
  - Stores key-value pairs in domain-specific file formats
- Interface management sublayer
  - Mapping domain abstractions to key-value pairs (ex: Directories)
  - Domain-specific interfaces (ex: POSIX)



Abstraction boundaries and mapping (1)



Abstraction boundaries and mapping (2)



Our storage management sublayer

- Key-value pairs stored in write-optimized Log-Structured Merge trees
  - Multicomponent trees with in-memory and on-disk parts
  - In-memory components provide buffering
  - Immutable on-disk components created by batch flushing
- LSM trees have several advantages over other indexing trees
  - Random metadata updates converted into sequential writes
  - Key format can be used to control locality
  - Short-lived metadata dies in memory
- Our LSM data structures
  - AVL tree as the in-memory component
  - Densely-packed B+-trees as on-disk components

Our interface management sublayer : POSIX emulation

- All POSIX metadata maintained in a single LSM tree
  - Unified key structure for storing directories and attributes
  - < parentID, name, record type > is used as the key
  - Special mechanism for handling hard links

Key	**Value**
<0, /, f> <0, /, r>	atime=2011-01-01 . . . id=1 links=4 mode=drwxr-xr-x . . .
<1, etc, f> <1, etc, r>	atime=2011-01-02 . . . id=5 links=2 mode=drwxr-xr-x . . .
<1, tmp, f> <1, tmp, r>	atime=2011-01-03 . . . id=3 links=2 mode=drwxr-xr-x . . .
<3, prog.c, f> <3, prog.c, r>	atime=2011-01-01 . . . size=2000 id=10 links=1 mode=-rw-r–r– . . .
<3, t.txt, f> <3, t.txt, r>	atime=2011-01-03 . . . size=100 id=13 links=1 mode=-rw——- . . .
<5, rc, f> <5, rc, r>	atime=2011-01-02 . . . size=1024 id=20 links=1 mode=-rwx—— . . .

Table: Mapping for /, /etc, /tmp, /tmp/prog.c, /tmp/t.txt and /etc/rc

Our interface management sublayer : real-time Indexing

- LSM-tree-based indexing of attributes
  - Policy-based inclusion/exclusion of attributes
  - Index updates in LSM trees incur little overhead
  - Separate merge parameters for index and metadata trees
  - All attributes indexed in a single tree
  - Uses <attribute ID, value, fileid> as the key

Key	**Value**
<atime, 2011-01-02, 20>	
<atime, 2011-01-03, 13>	
⋮	⋮
<size, 100, 13>	
<size, 1024, 20>	
⋮	⋮

Our interface management sublayer: attribute-based search

- Using typed virtual directories as query interface
  - Read-only directories created on the fly
  - Different plugins can be used to generate entries
  - Example: version virtual directory
- Attribute-based search virtual directory plugin
  - Query term is a combination of attributes/conditions
  - Conjunctive queries map onto hierarchies
  - Examples: cd [uid $=$ 100]/[size $>$ 1048576]
- Query evaluated using the auxiliary attribute index

Evaluation

- 31% speedup with Postmark
- 3-52% speedup with application benchmark
  - Copies src, build, find and grep, rm etcetera.
- Indexed search is 25x faster than the find utility
  - Find all files modified in the last N days (200,000 files)
  - Find all files with size $>$ 1 GB (200,000 files)
- Real-time indexing incurs moderate (10-15%) overhead
  - With both Postmark and application-level benchmarks while indexing seven frequently updated attributes

Conclusion

- Ad hoc, domain-specific metadata management solutions suffer from serious limitations
- Lack of modularity in traditional file systems complicates integration of metadata management
- Loris provides a modular, flexible framework for implementing such solutions
- Our naming layer design provides
  - High-performance metadata storage using LSM trees
  - Customizable, real-time indexing of attributes
  - Search-friendly, attribute-based interface in addition to the traditional POSIX interface