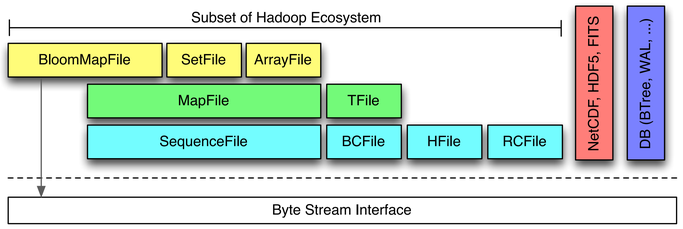
DataMods: Generalizing File System Services

Noah Watkins and Carlos Maltzahn
Systems Research Lab — UC Santa Cruz

Introduction

Scaling Applications

- Arrival of "Big Data" is pushing limits of storage systems
- Applications require scalability, and are growing more complex
- Existing interfaces (e.g. POSIX) are a roadblock to scaling



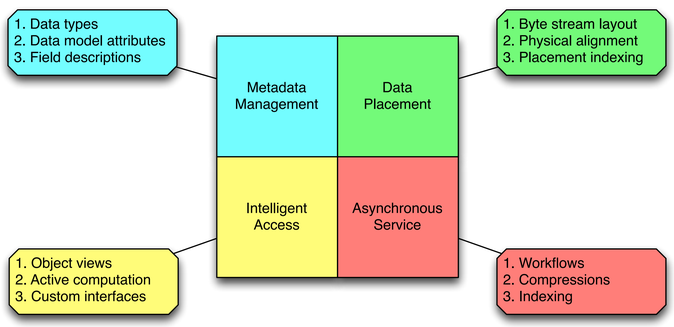
New Interfaces

- Storage systems take years to build and certify
- Scalable, open-source systems are in development *now*
- It is time to invest in alternative interface without fear of lock-in

Middleware Services

- Applications interact with complex data models
- File systems are used to share and persist application data
- Middleware maps the complex (application) to the simple (byte-stream)

Middleware Functionality



Storage System Services

Scalable Meta-data Management

- Indexing and file system hierarchy
- Fixed-size inode eliminates block lists

File Services and Operations

- Control over file layout (striping strategy)

Distributed Object Storage

- Local storage, cache, multi-core CPU, RAM
- Object behavior and interface defined by *class*

Recovery and Fault-tolerance

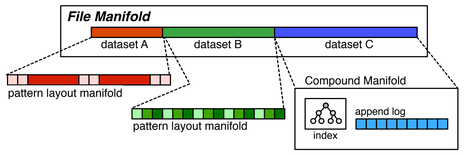
- Transparent handling of fault-tolerance (scalable shuffling)
- Scrubbing is an asynchronous background task

Data Model Modules

- Middleware duplicates services found in storage systems
- Expose storage system services with convenient abstractions

File Manifold

- Generalization of metadata storage and placement services
- Complex heterogeneous byte streams with custom striping strategy
- Container for complex data organizations



Active and Typed Storage

- Advanced interfaces that go beyond binary objects
- Programming model with well-defined performance costs
- Construction of domain-specific interfaces and processing routines

Asynchronous Services

- Middleware perform indexing, compression, and workflows
- Work must be performed online while files are opened
- Many tasks are amenable to offline, asynchronous completion

Use Case: Checkpoint/Restart

Motivation

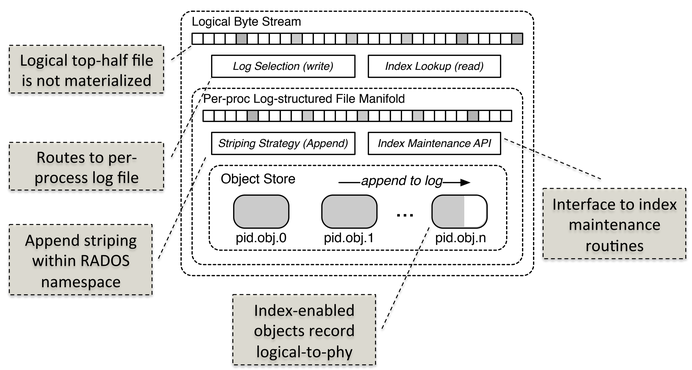
- Large-scale, long running computations need fault-tolerance
- Periodically checkpoint state to file system
- Many processes write to one file (N-1) or dedicated files (N-N)

Parallel Log-structured File System (Bent:SC09)

- Middleware transforms N-1 workloads into N-N transparently
- Global index is maintained that records *every* write
- Constructing and compressing the index online has overhead

PLFS File Manifold

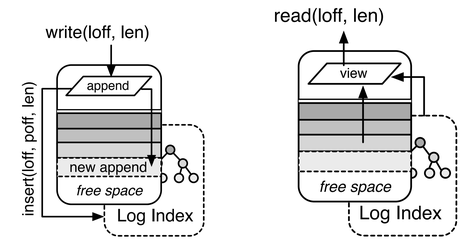
- Hierarchical manifold with logical file at top level
- Log-structured files form the lower level



Use Case: Checkpoint/Restart

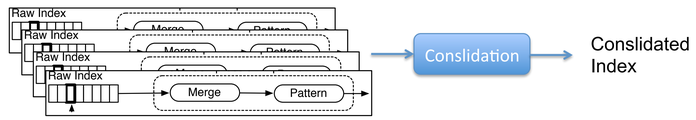
Automatic Indexing

- Storage system is able to observe and record all writes
- Low-level objects automatically index and append writes
- Building this object type is trivial, and operation is light-weight
- Data and temporal locality for index and payload I/O



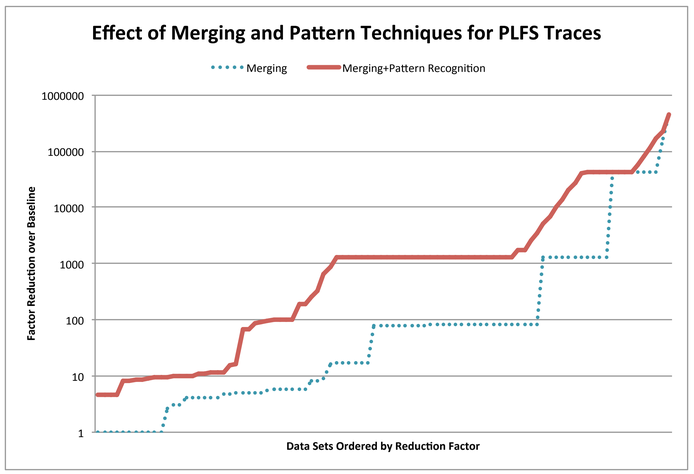
Offline Index Compression

- After a checkpoint, the global index is fragmented
- Two forms of compression reduce the size of the index
- Consolidation combines all of the fragments for efficient read I/O



Compression Performance

- Applied compression techniques to 92 published traces from LANL
- Compression of several orders of magnitude are possible



Future Work

- Formalizing the DataMods abstraction
- Applying techniques to other domains (e.g. Hadoop, Visualization)

jayhawk@cs.ucsc.edu