Accelerating Storage System Research Through a Common Framework

6th International LSDMA Symposium

Michael Kuhn

Research Group Scientific Computing
Department of Informatics
Universität Hamburg

2017-08-29

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

About Us: Scientific Computing

- Analysis of parallel I/O
- I/O & energy tracing tools
- Middleware optimization
- Alternative I/O interfaces
- Data reduction techniques
- Cost & energy efficiency

We are an Intel Parallel Computing Center for Lustre
(“Enhanced Adaptive Compression in Lustre”)

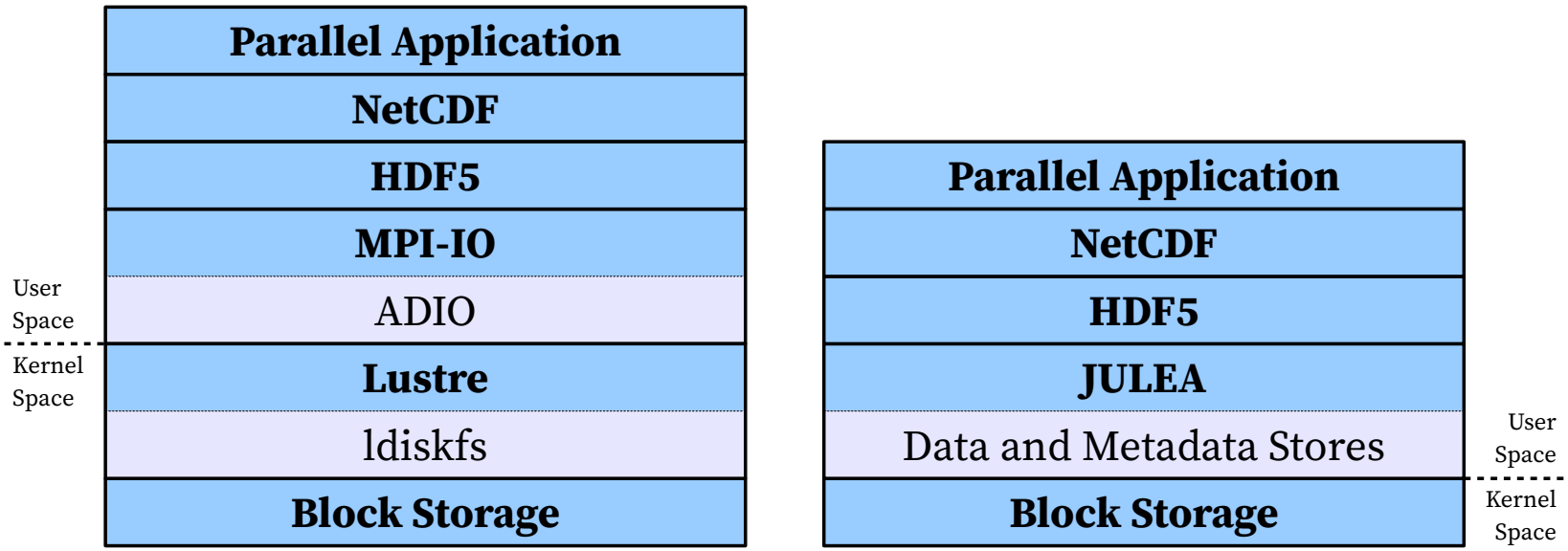
Motivation

- Hard to try new file system approaches
  - Changes to many different components required
  - File systems are typically monolithic in design
- Single interface, set of semantics and storage backend
  - Portability is an important factor
- Two major problems:
  1. Many specialized solutions for particular problems
     - Often based on existing file systems, seldom contributed back
  2. Necessary to have complete understanding of the file systems
     - Unnecessary hurdle for early-stage researchers and students

Motivation…

- Many projects implement basic functionality from scratch
  - Communication, distribution, backends etc.
- Possible solution is a flexible storage framework
  - Rapid prototyping of new ideas
  - Plugins for interface, storage backend and semantics
- JULEA is such a framework
  - Supports plugins that are configurable at runtime
  - Provides a convenient framework for research and teaching
  - Existing solutions have different focuses

Overview



(a) I/O stack commonly found in HPC

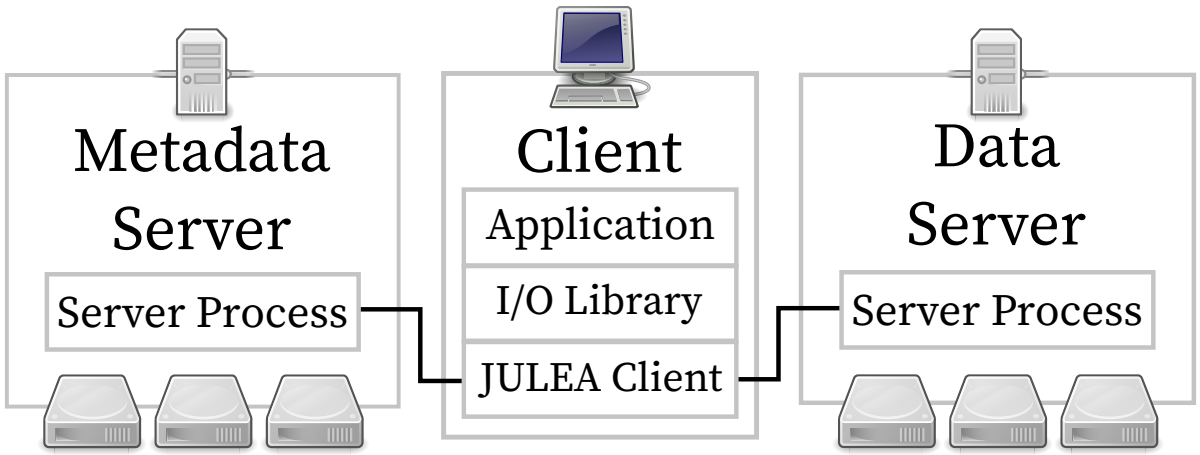
(b) Proposed I/O stack with JULEA

- JULEA runs completely in user space
- High-level libraries and applications can use it directly

Overview…

- Possible to offer arbitrary interfaces to applications
  - Traditional file system interfaces and completely new ones
- Servers are able to use many existing storage technologies
  - Support for multiple backends to foster experimentation
- Both clients and backends are easy to integrate and exchange
  - Can be changed at runtime through configuration file
- Dynamically adaptable semantics for all I/O operations
  - For example, POSIX and MPI-IO on a per-operation basis

Overview…



- Applications can use one or more JULEA clients
  - Clients can be used either directly by applications or by adapting I/O libraries to make use of them
- Servers are split into data and metadata servers
  - Allows tuning the servers for their respective access patterns

Clients

- File systems typically offer a single interface
  - Interwoven with the rest of the file system architecture
- JULEA clients are unrestricted regarding their interfaces
  - User space, therefore arbitrary interfaces can be provided
  - Typically problematic for kernel space file systems due to VFS
- Useful for both applications and I/O libraries
  - For instance, HDF5 directly on top of JULEA

Backends

- Separated into data and metadata backends
  - Additionally, client and server backends
- Data backends manage objects
  - Influenced by file systems (Lustre and OrangeFS), object stores (Ceph's RADOS) and I/O interfaces (MPI-IO)
- Metadata backends manage key-value pairs
  - Influenced by database (SQLite and MongoDB) and key-value (LevelDB and LMDB) solutions
- Backends support namespaces
  - Multiple clients can co-exist and not interfere with each other

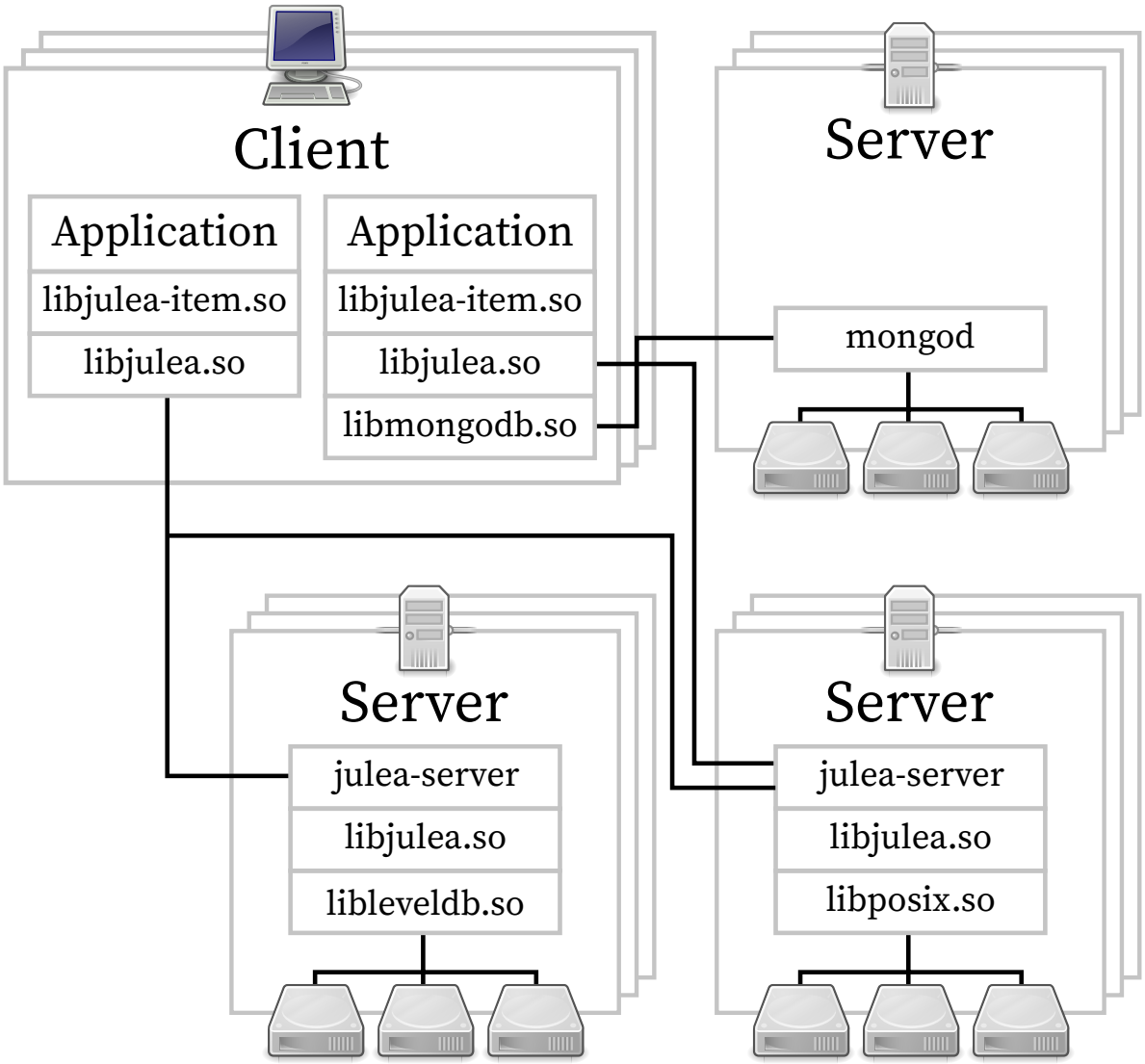
Semantics

- Adapt file system to application instead of other way around
- Operations' semantics can be changed at runtime
  - Different categories: atomicity, concurrency, consistency, ordering, persistency and safety
- Possible to mix the settings for each of these categories
  - Not all combinations might produce reasonable results
- Templates to emulate existing semantics such as POSIX
- Clients can fix appropriate semantics or give control to users

Implementation

- Modern C11 code
  - Automatic cleanup of variables etc.
- Open source (LGPL 3.0 or later)
  - Available at `https://github.com/wr-hamburg/julea`
- Only two mandatory dependencies
  - GLib for data structures, libbson for (de)serialization
- Clients are provided in the form of shared libraries
  - Allow applications to use multiple clients at the same time
- Server can function as both a data and metadata server
- Integrated support for tracing, unit tests etc.

Implementation…



Implementation…

- **object**: direct access to JULEA's data store
  - Able to access arbitrary namespaces
  - Provides abstractions for other clients
- **kv**: direct access to JULEA's metadata store
  - Able to access arbitrary namespaces
  - Provides abstractions for other clients
- **item**: cloud-like interface
  - Collections and items with flat hierarchy
- **posix**: POSIX file system using FUSE

Implementation…

- **posix**: compatibility with existing POSIX file systems, certain functionalities are duplicated
- **gio**: uses the GIO library that supports multiple backends of its own (including POSIX, FTP and SSH)
- **lexos**: uses LEXOS to provide a light-weight data store
- **null**: intended for performance measurements of the overall I/O stack, discards all incoming data
- **leveldb**: uses LevelDB for metadata storage
- **lmdb**: uses LMDB for metadata storage
- **mongodb**: uses MongoDB, maps key-value pairs to documents
- **sqlite**: uses SQLite, maps key-value pairs to rows

Coupled Storage Systems

- File systems currently in use offer a POSIX interface
  - File data is an opaque byte stream
- Self-describing data formats such as HDF5 are widely used
  - Data structure is encoded in the files themselves
- Two different types of metadata
  - **File system metadata**: stored on metadata servers
  - **File metadata**: stored within files on data servers
- Opening a file and reading data:
  1. Accessing file system metadata to identify data servers
  2. Reading file metadata to determine portions of the file
  3. Reading actual data from the file system

Coupled Storage Systems…

- Strict separation leads to inefficient file access
  - Maintaining file metadata requires access to shared file
  - File metadata is distributed across data servers
  - Data servers are commonly optimized for streaming I/O
- Approach: Couple data formats and storage systems
  - Give I/O libraries access to low-level functionality
  - Allow storage system to better understand data structure
- Example: native HDF5 support for JULEA
  - Map file metadata to JULEA's metadata backends
  - Groups, attributes etc. stored on metadata servers
  - Proof-of-concept implementation with SQLite

Coupled Storage Systems…

- Currently, opaque key-value pairs
  - Clients can encode arbitrary values
  - Servers do not know their structure
- Introduce schemas for metadata
  - Allow clients to specify schemas that the server can use
  - Query file system metadata and file metadata
- Allow complex queries without scanning the whole file system
  1. Return all files larger than 128 KiB

```
SELECT *
FROM files
WHERE file_size > 131072
```

  2. Return average file size

```
SELECT AVG(file_size)
FROM files
```

Future Work

- Implement an HDF5 VOL plugin
  - Map data to objects and metadata to key-value pairs
- Further extend JULEA's backend support
  - Data backend for Ceph's RADOS, metadata schema support
- Further improvements to JULEA's backend interface
  - Should remain stable in the foreseeable future
  - Provide a reliable base for third-party plugins
- Improve dynamically adaptable semantics
  - Refine based on real-world requirements
  - Implement support for additional semantics

Summary

- JULEA provides a flexible storage framework
  - Contains necessary building blocks for storage systems
  - Facilitates rapid prototyping and evaluation
- Runs completely in user space and has few dependencies
  - Easy to debug and develop
  - Possible to use on clusters without root access
- Feedback and contributions are always welcome
  - `https://github.com/wr-hamburg/julea`