JULEA: A Flexible Storage Framework for HPC

Workshop on Performance and Scalability of Storage Systems

Michael Kuhn

Research Group Scientific Computing
Department of Informatics
Universität Hamburg

2017-06-22

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 majors 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 young researchers and students

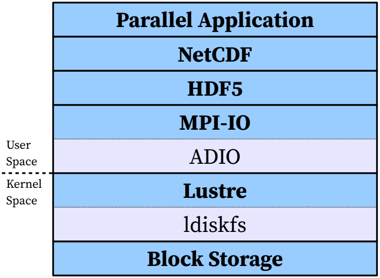
Motivation…

- Applications rely on high-level I/O libraries
  - Exchangeability of data is a primary concern
  - Self-describing data formats such as NetCDF and HDF5
- Multiple projects investigate integrating I/O libraries and file systems more closely (DAOS, ESiWACE etc.)
  - Hard to achieve with current file systems
  - Requires extensive changes
- Related research
  - HPC and big data convergence
  - Alternative file system interfaces

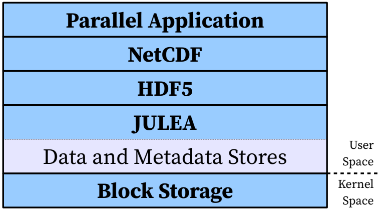
Motivation…

- 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



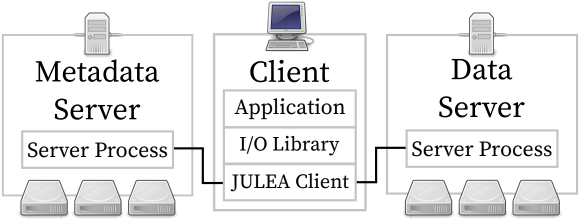
(b) Proposed I/O stack with JULEA

- 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 a 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
- Clients are completely 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
  - Allows multiple clients to co-exist and not interfere

Semantics

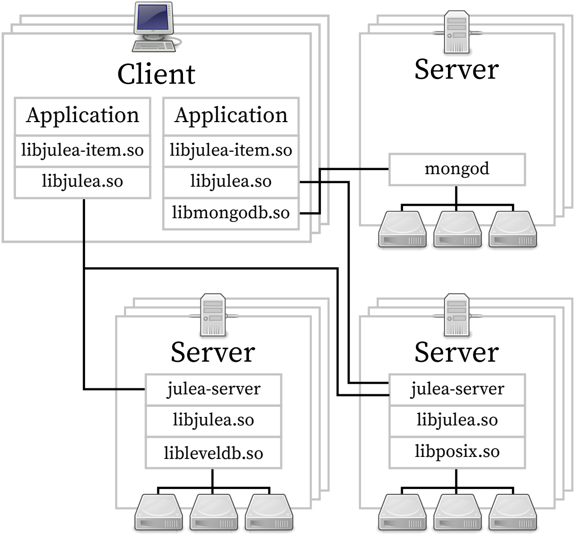
- Adapt file system to application instead of other way around
- Operations' semantics can be changed at runtime
  - Different categories: atomicity, concurrency, consistency, ordering, persistency and safety
- Possible to mix the settings for each of these categories
  - Not all combinations might produce 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)[1]
- 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.

[1] `https://github.com/wr-hamburg/julea`

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
- **mongodb**: uses MongoDB, maps key-value pairs to documents

Evaluation Setup

- Performance depends on the used data and metadata backends
  - Focus on some general performance aspects
- Two configurations:
  - **local**: desktop machine (Intel Xeon E3-1225v3) with a consumer SSD (Samsung SSD 840 EVO)
  - **remote**: two dual-socket nodes (Intel Xeon X5650) with an HDD (Seagate Barracuda 7200.12), connected via Gbit Ethernet

Performance Results

Storage	Backend	Operation	Perf. (local)	Perf. (remote)
Data	POSIX	Create	19,500 ops/s	3,600 ops/s
		Delete	29,500 ops/s	4,300 ops/s
		Status	39,500 ops/s	4,500 ops/s
	NULL	Create	49,000 ops/s	5,500 ops/s
		Delete	49,500 ops/s	5,000 ops/s
		Status	49,500 ops/s	4,900 ops/s
Metadata	LevelDB	Put	41,500 ops/s	4,300 ops/s
		Delete	43,000 ops/s	4,300 ops/s
	MongoDB	Put	7,500 ops/s	1,400 ops/s
		Delete	8,000 ops/s	1,500 ops/s

- Both: MongoDB much slower than LevelDB
- Remote: performance mainly limited by network

Performance Results…

Safety	Operation	Perf. (local)	Perf. (remote)
None	Put	225,000 ops/s	62,000 ops/s
	Delete	197,000 ops/s	59,500 ops/s
Network	Put	41,500 ops/s	4,300 ops/s
	Delete	43,000 ops/s	4,300 ops/s
Storage	Put	29,500 ops/s	3,800 ops/s
	Delete	31,000 ops/s	3,900 ops/s

- Network: operations require one round trip
- None: performance much higher due to pipelining

Future Work

- Basic storage framework and some initial backends finished
- 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 backend for LMDB
- Further improvements to JULEA's backend interface
  - Should remain stable in the foreseeable future
  - Provide a reliable base for third-party plugins

Summary

- JULEA provides a flexible storage framework
  - Contains necessary building blocks for storage systems
  - Facilitates rapid prototyping and evaluation
- Few dependencies and can be used without system-level access
  - Easy to use on clusters
- Runs completely in user space
  - Easy to debug and develop