I/O Semantics for Future Storage Systems

Michael Kuhn, Julian Kunkel

Research Group Scientific Computing
Department of Informatics
University of Hamburg

2015-04-29

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

HPC I/O Stack

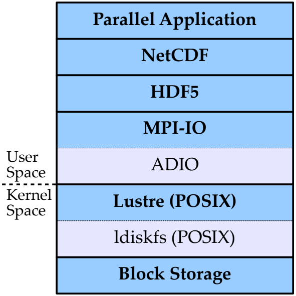


Figure: HPC I/O stack

- Complex interactions
  - Optimizations on each layer
- Applications use structured data
  - Matrices, vectors etc.
  - Often in the form of time series
- A POSIX file is a byte stream
  - All high-level information is lost
  - Data types, required semantics

Syntax and Semantics

- Even high-level interfaces have no knowledge about the applications' I/O requirements
  - Optimizations are often based on heuristics
- Semantical information can provide needed knowledge
- Interfaces comprise two parts
  - Syntax defines available operations
  - Semantics defines operations' behavior
  - Both are typically static

I/O Semantics

- POSIX features very strict consistency requirements
  - Changes have to be visible to other clients immediately
  - I/O is intended to be atomic
  - Effectively prohibits client-side caching
- MPI-IO's consistency requirements are less strict
  - Changes are immediately visible only to the process itself
  - Correctly handles non-overlapping or non-concurrent writes
- File systems force POSIX semantics upon higher layers

I/O Semantics…

- I/O semantics can only be changed in a limited fashion
  - `strictatime`, `relatime` and `noatime` change the file system's behavior regarding the last access timestamp
  - `posix_fadvise` allows announcing the access pattern
  - MPI-IO's atomic mode for stricter consistency semantics
- Provided facilities are often restricted
  - Usually only possible at file open or mount time
  - Often apply to the whole file

Feature Wishlist

- Allow developers to specify application requirements
  - Offer fine-grained control
  - Leverage semantical information across the whole stack
- Adapt file system to the applications' requirements
  - Perform optimizations that are applicable to the semantics
- High level of abstraction
  - Less focus on technical aspects
  - Closer to application semantics

New I/O Stack

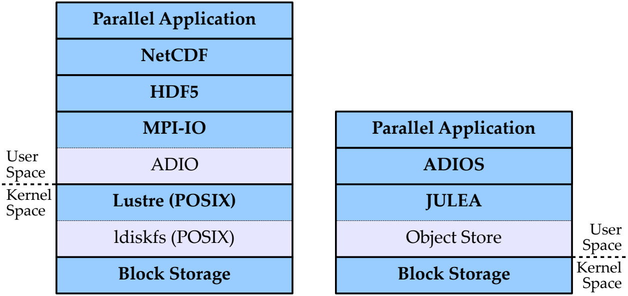


Figure: HPC and JULEA I/O stacks

- Easier to analyze, concentration into a single layer
- Pass semantical information into the file system

Features

- Semantics are dynamically adaptable according to the applications' I/O requirements
  - Developers can specify coarse-grained ("checkpoint") or fine-grained requirements ("strict consistency semantics")
  - File system can tune operations for specific applications
- All accesses to the file systems are performed via batches
  - Each batch can consist of multiple operations
  - Combine different kinds of operations within one batch

Interface

```
batch = new Batch(POSIX_SEMANTICS);

store = julea.create("test store", batch);
collection = store.create("test collection", batch);
item = collection.create("test item", batch);
item.write(..., batch);

batch.execute();
```

Listing 1: Executing multiple operations in one batch

- Namespace is split into stores, collections and items
- Provide a defined point for the operations' execution
  - Traditional approaches can only guess

Semantics

- Many important aspects of the semantics can be changed
  - Performance-related: atomicity, concurrency, consistency, ordering, persistency and safety
  - Further ideas: redundancy, security, transformation
- Templates for easy use and established semantics
  - Default: Concurrent non-overlapping operations
  - POSIX: Provided for backwards compatibility
  - Temporary (local): Allow transparent use of advanced technologies such as node-local buffers

Semantics…

- Atomicity: Whether accesses should be executed atomically
  - Avoid locking for multi-server operations
- Concurrency: Whether concurrent accesses will take place and how the access pattern will look like
  - Efficiently handle different patterns without heuristics
- Consistency: When clients will see modifications of others
  - Enable/disable client-side read caching

Semantics…

- Ordering: Whether batch operations can be reordered
  - Group operations for more efficient access
- Persistency: When modifications will be visible globally
  - Enable/disable client-side write caching
- Safety: When data/metadata will be durable
  - Guarantee state of data/metadata after execution

Lustre vs. JULEA

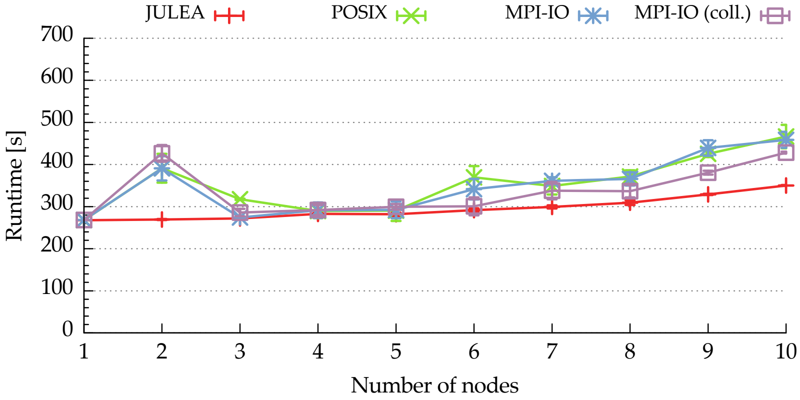


Figure: Checkpointing into a shared file (one process per node)

Lustre vs. JULEA…

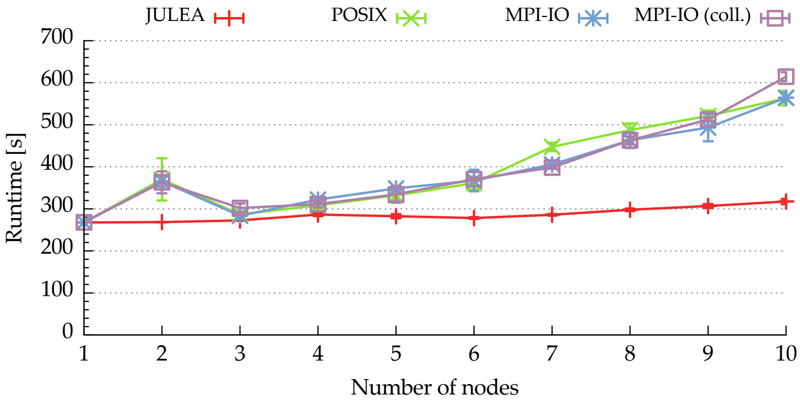
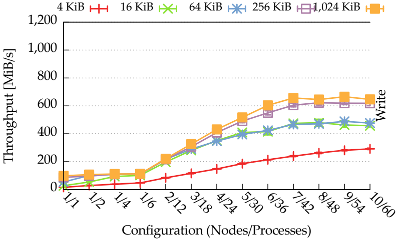
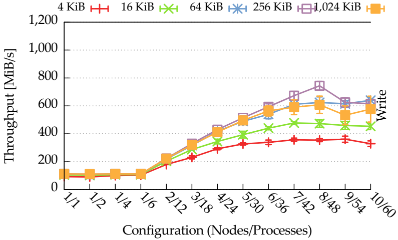


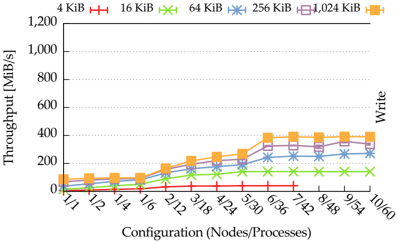
Figure: Checkpointing into a shared file (six processes per node)



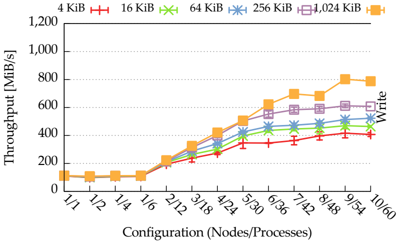
(a) JULEA: individual items



(b) JULEA: individual items (batch)



(c) JULEA: individual items (atomic)



(d) JULEA: individual items (unsafe)

Application Compatibility

- Combine with ADIOS
  - Abstract description of applications' I/O using XML
- Already offers some helpful information
  - `adios_start_calculation`
  - `adios_stop_calculation`
  - `adios_end_iteration`

Application Compatibility. . .

- Extend with further semantical information
  - Required I/O semantics
  - Compressibility etc.

```
<adios-config host-language="C">
  ...
  <semantics group="checkpoint" concurrency="non-overlapping"/>
  <semantics group="buffer" template="temporary-local"/>
</adios-config>
```

Listing 2: ADIOS extensions

Gap Between Computation and Storage

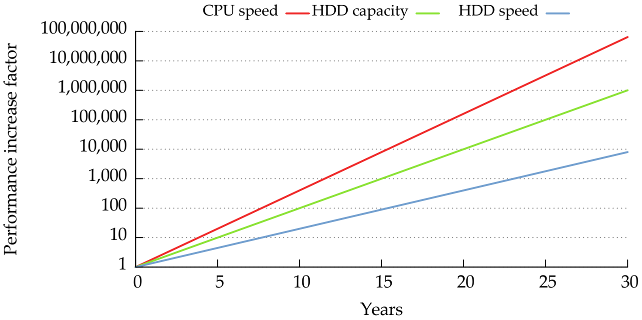


Figure: Development of CPU speed, HDD capacity and HDD speed

- I/O is becoming an increasingly important problem
  - Data can be produced faster but it becomes harder to store it
- Consequence: Spend more money on storage

Example: DKRZ

	2009	2015	Factor
Performance	150 TF/s	3 PF/s	20x
Nodes	264	2,500	9.5x
Node performance	0.6 TF/s	1.2 TF/s	2x
System memory	20 TB	170 TB	8.5x
Storage capacity	5.6 PB	45 PB	**8x**
Storage throughput	30 GB/s	400 GB/s	13.3x
Disk drives	7,200	8,500	1.2x
Archive capacity	53 PB	335 PB	**6.3x**
Archive throughput	9.6 GB/s	21 GB/s	**2.2x**
Power consumption	1.6 MW	1.4 MW	0.9x
Investment	30 M€	30 M€	1x

Data Reduction

- Several approaches for data reduction
  - Compression, deduplication and recomputation
- Deduplication needs too much main memory
  - More than the current supercomputer is equipped with
- Recomputation might become viable in the future
  - Cost for computation and storage currently in balance
  - Preservation is problematic

Data Reduction...

- Compression is promising
  - 33 % savings with negligible overhead
- Currently only static approaches for compression
  - One compression algorithm per file system
- Use semantical information to improve compression
  - Working on adaptive compression
  - More efficient application-specific compression

Conclusion

- POSIX is portable but inflexible
  - No way to relax semantics
- Static approaches are only suitable for a subset of use cases
  - Other file systems are also limited to their semantics
- New approach offers solutions
  - Adapt semantics according to the application requirements
  - Use semantical information across the complete I/O stack

Outlook

- Dynamically adaptable semantics for established interfaces
  - Semantics suited for modern HPC applications
  - Common set of configurable parameters
- Extend ADIOS to support more semantical information
  - Adapt I/O semantics according to application requirements
  - Exploit information for advanced data reduction