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
1. Introduction and Motivation

2. Dynamically Adaptable I/O Semantics

3. Data Reduction Techniques

4. Conclusion and Outlook
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

Figure: HPC I/O stack
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

```java
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

![Diagram showing runtime for different file systems with varying number of nodes.]

**Figure:** Checkpointing into a shared file (one process per node)
Lustre vs. JULEA...

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

Dynamically Adaptable I/O Semantics

Data Reduction Techniques

Conclusion and Outlook

---

**JULEA: individual items**

(a)

**JULEA: individual items (batch)**

(b)

**JULEA: individual items (atomic)**

(c)

**JULEA: individual items (unsafe)**

(d)
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.

```xml
<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

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