Towards new storage interfaces – chance or curse?

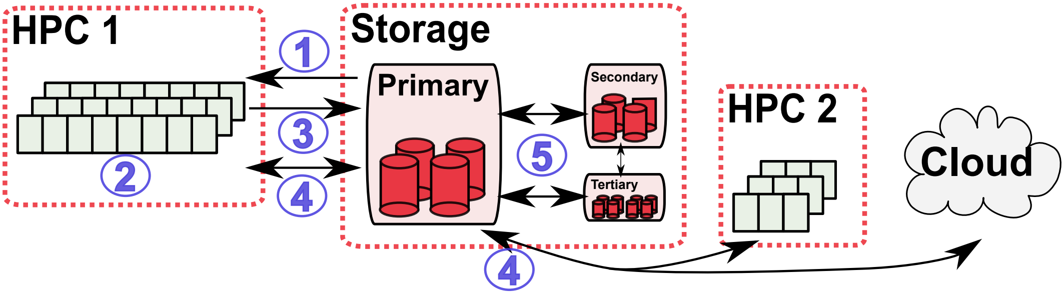
Julian M. Kunkel (DKRZ)

2017-09-26

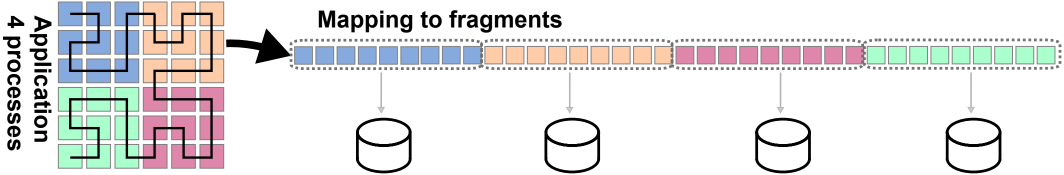
Outline

1. HPC Storage Landscape
2. Thoughts
3. Better Interfaces?
4. Community APIs

HPC Storage Usage: Workflows

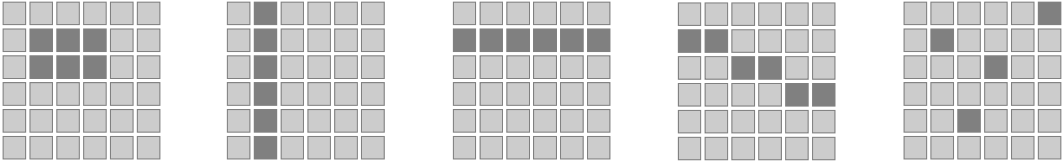


Mapping of a 2D field from a parallel application to storage



Mapping for Pre-Post

User-defined analysis of ND datasets leads to various patterns



User Perspective: Accessing Data

Multitude of data models

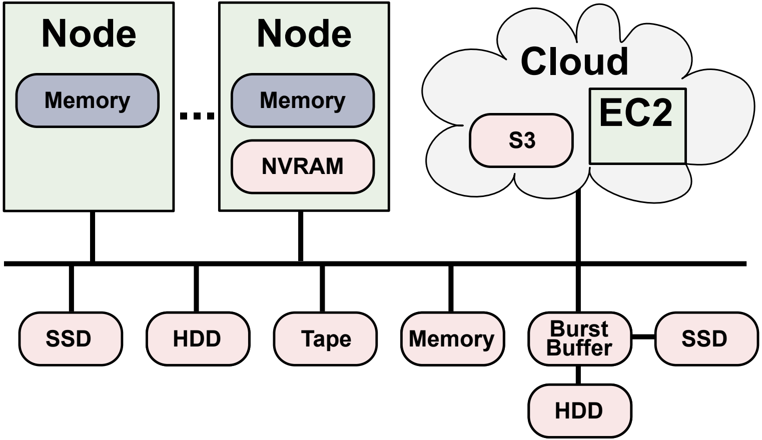
- POSIX File: Array of bytes
- HDF5: Container like a file system
  - Dataset: N-D array of a (derived) datatype
  - Rich metadata, different APIs (tables)
- Database: structured (+arrays)
- NoSQL: document, key-value, graph, tuple

Choosing the right interface is difficult – workflow may involve several

Properties / qualities

- Namespace: Hierarchical, flat, relational
- Access: Imperative, declarative, implicit (`mmap()`)
- Concurrency: Blocking vs. non-blocking
- Consistency semantics: Visibility and durability of modifications

Storage Landscape of Future Systems



HPC system with compute nodes and storage

Outline

1 HPC Storage Landscape

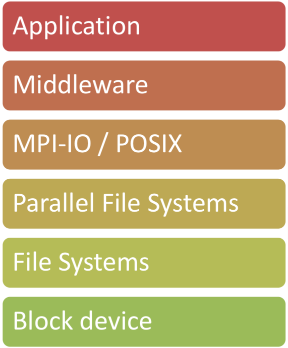
2 Thoughts
- Storage stack
- Performance Optimization

3 Better Interfaces?

4 Community APIs

Peeking at the Current I/O Stack – System Perspective

- Coexistence of access paradigms
  - File (POSIX, ADIOS, HDF5), SQL, NoSQL
- Semantical information is lost through layers
  - Suboptimal performance
- Reimplementation of features across stack
  - Unpredictable interactions
  - Wasted ressources
- Restricted (performance) portability
  - Optimizing each layer for each system?



Example I/O stack

Limitations of the current software stack

Platform

1. Zoo of interfaces
2. Low-level storage APIs
3. Loss of semantical information
4. Interference of applications / lack of coordination
5. All data treated identically

Software

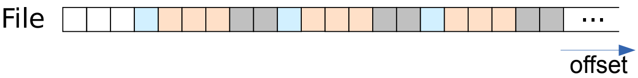
1. Explicit workflows
2. Unclear access patterns (users, sites)
3. No performance awareness
4. Lack of technological knowledge (from users, for tweaking)
5. Manual tiering (or with policies)

Semantical Gap of File Access (1)

Applications work with (semi)structured data

- Vectors, matrices, n-Dimensional data

A file is just a sequence of bytes!



Applications/Programmers must serialize data into a flat namespace

- Uneasy handling of complex data types
- Mapping is performance-critical (on HDDs)
- Vertical data access unpractical (e.g. to to pick a slice of multiple files)

Semantical Gap of File Access (2)

Information hidden from file systems

- Data types
- Data semantics
- Value of data
- Type: Checkpoint, computed, original, logfile
- Data lifecycle: production, usage, deletion

Characteristics can even vary within a file, e.g. for metadata

Storage systems could use this information for

- Improving performance: Automatic tiering, caching, replication
- Simplifying management: ILM, offering alternative data views
- Correctness: Ensuring data consistency

Performance Tweaks

- There are many options to tune the I/O-stack
  - API: `posix_fadvise()`, HDF5 properties, open flags, cache size
  - Via command line: `lfs setstripe`
  - Setup/initialization of a storage system
  - Mounting options and procfs settings
- Many options are of technical nature
  - Performance gain/loss depend on hardware, software
  - Specific to file system, API (MPI, POSIX, HDF5)
  - Many types of hints/tweaks are not portable
- Performance loss forces us to use these optimization

Performance Tweaks

- There are many options to tune the I/O-stack
  - API: `posix_fadvise()`, HDF5 properties, open flags, cache size
  - Via command line: `lfs setstripe`
  - Setup/initialization of a storage system
  - Mounting options and procfs settings
- Many options are of technical nature
  - Performance gain/loss depend on hardware, software
  - Specific to file system, API (MPI, POSIX, HDF5)
  - Many types of hints/tweaks are not portable
- Performance loss forces us to use these optimization

Usually we are losing system performance!

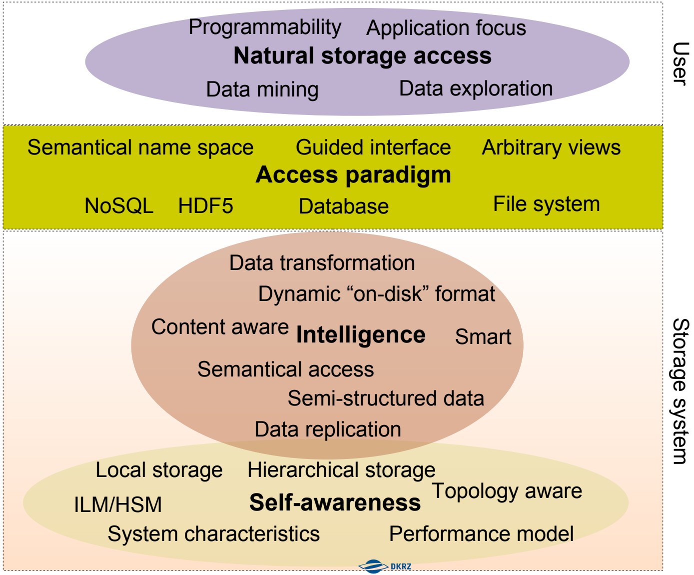
Critical Discussion

Questions from the users' perspective

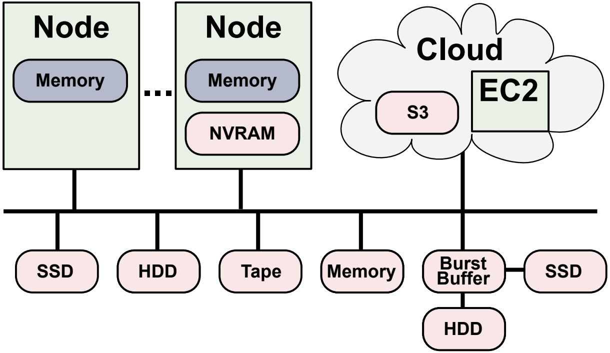
- Why do I have to organize the file format?
  - It's like taking care of the memory layout of C-structs
- Why do I have to convert data between storage paradigms?
- Why must I provide system specific performance hints?
  - It's like telling the compiler to unroll a loop exactly 4 times
- Why can't I rely on a correct implementation of the consistency model?
  - Parallel file systems have performance issues with most models
- Why is a file system not offering the consistency model I need?
  - My application knows the required level of synchronization

Would you rather like to code an actual application?

Personal Vision: Towards Intelligent Storage Systems and Interfaces



Coexistence of Storage Systems vs. Tiering



- We shall be able to use all storage technologies concurrently
  - Without explicit migration etc. put data where it fits
  - Administrators just add a new technology (e.g., SSD pool) and users benefit
  - Should be steered by a standard and open interface
  - Open ecosystem for any vendor...

Additional Responsibilities of Storage System

- Mapping of data structures
- Flexible semantics
- Compute offloading, see success of big data tools
- Tight integration of workflows
- Advanced performance assessment
- Namespace based on metadata
- Management tools
- ...

Outline

1 HPC Storage Landscape

2 Thoughts

3 Better Interfaces?
- Guided Interfaces
- Compression Example
- SCIL
- ESDM

4 Community APIs

Exascale10 Initiative Term: Guided Interfaces

Guiding vs. automatism vs. technical hints

Users provide additional information to guide an intelligent system.

The I/O stack may exploit this information or not.

Systems could improve over time by using the information better.

Information which could be provided by users

- Data types
- Semantics
- Relations between data
- Lifecycle (especially usage)

Several issues have been addressed in different access paradigms.

Also some behavioral hints exist: open() flags, fadvise(), ...

Compression Research: Involvement

- **Scientific Compression Library** (SCIL)
  - Separates concern of **data accuracy** and **choice of algorithms**
  - Users specify necessary accuracy and performance parameters
  - Metacompression library makes the choice of algorithms
  - Supports also new algorithms
  - Ongoing: standardization of useful compression quantities
- Development of algorithms for lossless compression
  - MAFISC: suite of preconditioners for HDF5, pack data optimally, reduces climate data by additional 10-20%, simple filters are sufficient
- Cost-benefit analysis: e.g., for long-term storage MAFISC pays of
- Analysis of compression characteristics for earth-science related data sets
  - Lossless LZMA yields best ratio but is very slow, LZ4fast outperforms BLOSC
  - Lossy: GRIB+JPEG2000 vs. MAFSISC and proprietary software
- Method for system-wide determination of ratio/performance
  - Script suite to scan data centers...

SCIL: Supported User-Space Quantities

Quantities defining the residual (error):

absolute tolerance: compressed can become true value ± absolute tolerance

relative tolerance: percentage the compressed value can deviate from true value

relative error finest tolerance: value defining the abs tol error for rel compression for values around 0

significant digits: number of significant decimal digits

significant bits: number of significant decimals in bits

field conservation: limits the sum (mean) of field's change

Quantities defining the performance behavior:

compression throughput

decompression throughput

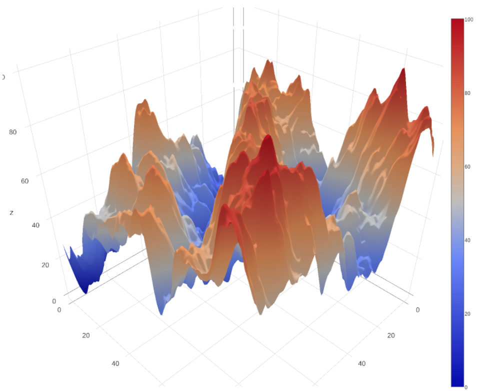
- in MiB or GiB, or relative to network or storage speed

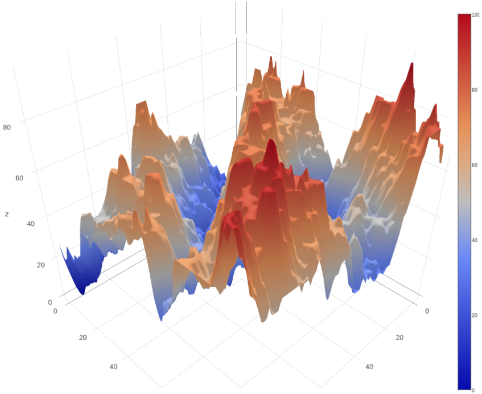
Aim to standardize user-space quantities across compressors!

See `https://www.vi4io.org/std/compression`

SCIL Provides Typical Synthetic Data

Example: Simplex (options 206, 2D: 100x100 points)





Right picture compressed with Sigbits 3bits (ratio 11.3:1)

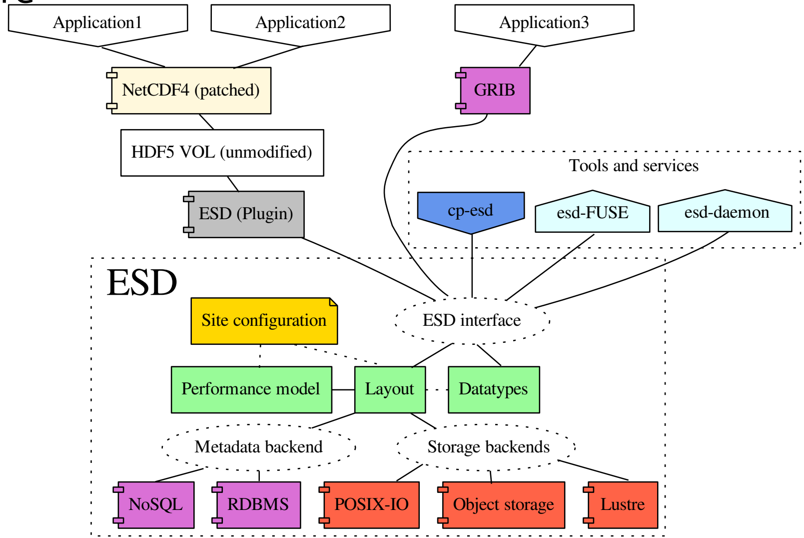
Ongoing Activity: Earth-Science Data Middleware

Part of the ESiWACE Center of Excellence in H2020

Design Goals of the Earth System Data Middleware

1. Understand application data structures and scientific metadata
2. Flexible mapping of data to multiple storage backends
3. Placement based on site-configuration + performance model
4. Site-specific optimized data layout schemes
5. Relaxed access semantics, tailored to scientific data generation
6. A configurable namespace based on scientific metadata

Architecture



Outline

1 HPC Storage Landscape

2 Thoughts

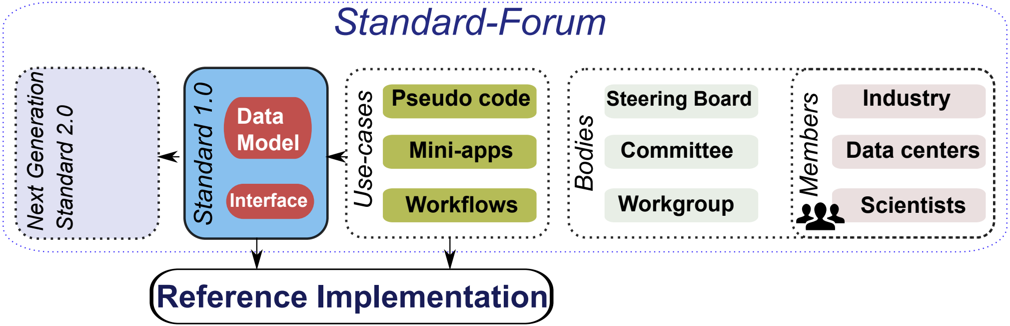
3 Better Interfaces?

4 Community APIs

- Community

A Potential Approach in the Community: Following MPI

- The **standardization** of a high-level *data model & interface*
  - Targeting data intensive and HPC workloads
  - Lifting semantic access to a new level
- Development of a reference implementation of a **smart runtime system**
  - Implementing key features
- Demonstration of benefits on socially relevant data-intense apps



API Key Features

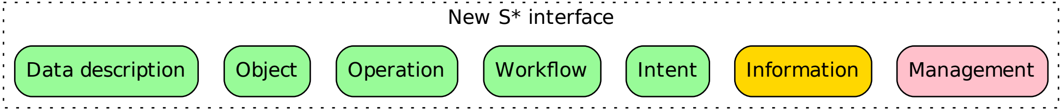
- High-level data model for HPC
  - Storage understands data structures vs. byte array
  - Relaxed consistency
- Semantic namespace
  - Organize based on domain-specific metadata (instead of file system)
  - Support domain-specific opperations and addressing schemes
- Integrated processing capabilities
  - Offload data-intensive compute to storage system
  - In-situ/In-transit workflows
- Workflow management
  - Managed data-driven workflow
- Performance-portability
  - Guided interfaces: Intents vs. technical hints
- Enhanced data management features
  - Embedded performance analysis
  - Resilience, import/export, ...

API development

Development of the data model

- Establishing a Forum similarly to MPI
- Define data model for HPC
  - Must be beneficial for Big Data + Desktop, too
- Open board: encourage community collaboration

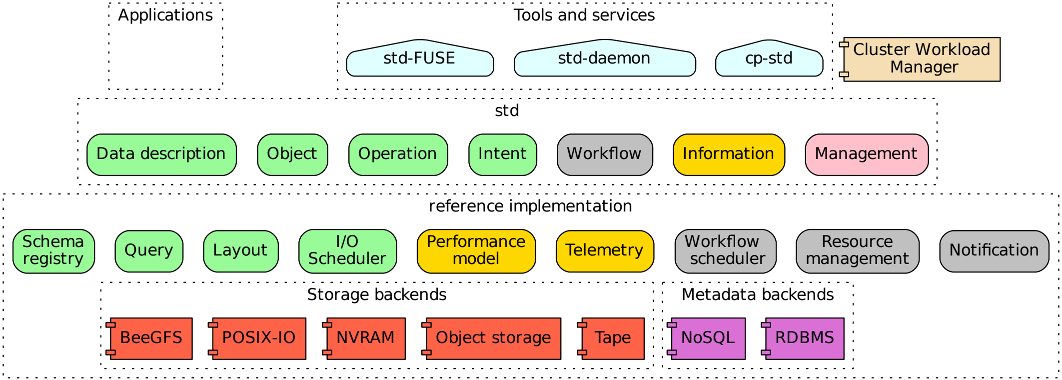
Current Draft



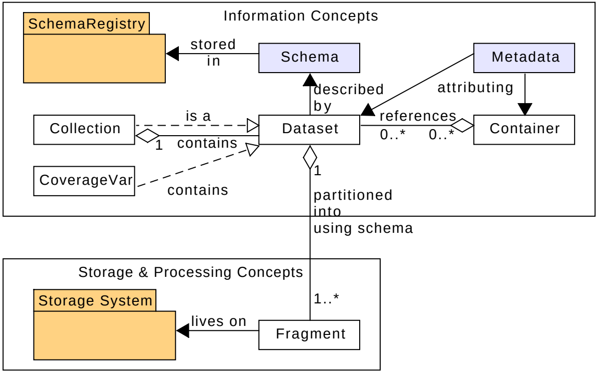
Reference Implementation: Goals

- Semantic access
  - Search and access based on metadata
- Self-aware
  - Understand performance characteristics
- Automatic layouting + smart data replication
  - Adapt data layout during runtime
- Managed workflows
  - Scheduler considers compute and I/O requirements
- Compatibility

Architecture Draft



Data Model


Information Concepts
SchemaRegistry
stored in
Schema
Metadata
described by
attributing
Collection
is a
Dataset
references
0..*
0..*
Container
1
contains
CoverageVar
contains
1
partitioned into using schema
Storage & Processing Concepts
Storage System
1..*
lives on
Fragment

Processing Model