Computer Science Department

Challenges and Approaches for Extreme Data Processing

Limitless Storage
Limitless Possibilities

https://hps.vi4io.org

Julian M. Kunkel

Universität zu Köln

2019-01-14

LIMITLESS **POTENTIAL** | LIMITLESS **OPPORTUNITIES** | LIMITLESS **IMPACT**

Outline

High-Performance Computing (HPC)

Definitions

- HPC: Field providing massive compute resources for a computational task
  - Task needs too much memory or time for a normal computer
  - ⇒ Enabler of complex scientific simulations, e.g., weather, astronomy
- Supercomputer: aggregates power of many compute devices

Example Supercomputers

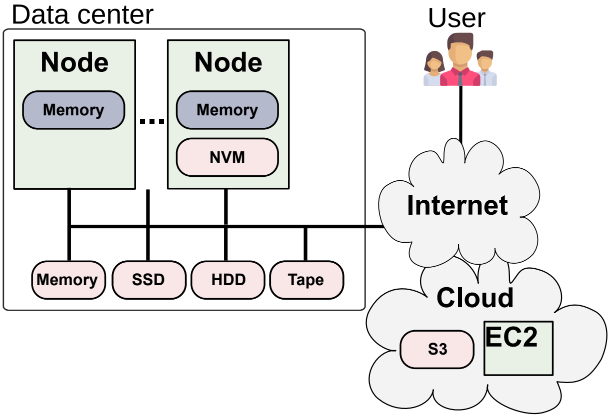
DKRZ: Mistral

- Compute: 3,000 dual socket nodes
  - Linpack: 3 Petaflop/s (10^{15})
- Storage: 52 Petabyte
  - 10k HDDs, 300 servers
  - Cost: 6 M€

RRZK Cologne: CHEOPS

- Compute: 831 nodes (dual/quad)
  - Linpack: 86 Teraflop/s
  - Storage: 500 Terabyte

Supercomputers & Data Centers



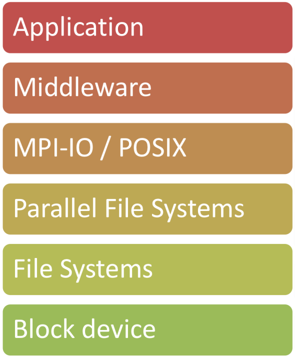
Juwels Cluster

Copyright: Forschungszentrum Jülich

A View on The I/O Stack

- Parallel application
  - Is distributed across many nodes
  - Has a specific access pattern for I/O
  - May use several interfaces
    File (POSIX, ADIOS, HDF5), SQL, NoSQL
- Middleware provides high-level access
- POSIX: ultimately file system access
  - Provides a hierarchical namespace and “file” interface
- Parallel file system: Lustre, GPFS, PVFS2
  - Parallel: multiple processes can access data concurrently
- File system: EXT4, XFS, NTFS
- Operating system: (orthogonal aspect)

The layers provide optimization strategies and tunables



Example I/O stack

Challenges

- Achieving high performance
- Understanding observed behavior (and performance)
  - The I/O hardware/software stack is very complex – even for experts
- Tuning system settings and configurations
- Enabling performance portability
- Managing files and (data-intense) workflows
- Utilizing heterogenous storage landscapes

These are opportunities for tools and method development!

- Diagnosing causes, predicting performance, prescribing settings
- Smarter ways of data handling

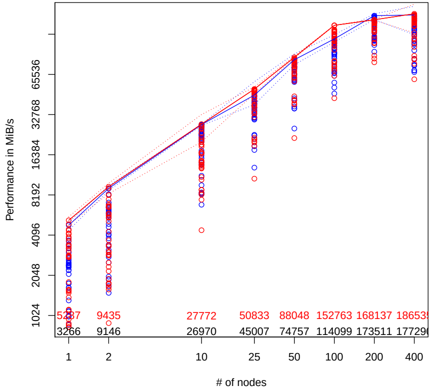
The Performance Challenge

- DKRZ file systems offer about 700 GiB/s throughput
  - However, I/O operations are typically inefficient: Achieving 10% of peak is good
- Influences on I/O performance
  - Application's access pattern and usage of storage interfaces
  - Network congestion
  - Slow storage media (tape, HDD, SSD)
  - Concurrent activity – shared nature of I/O
  - Tunable optimizations deal with characteristics of storage media
  - These factors lead to complex interactions and non-linear behavior

Illustration of Performance Variability

University of Reading

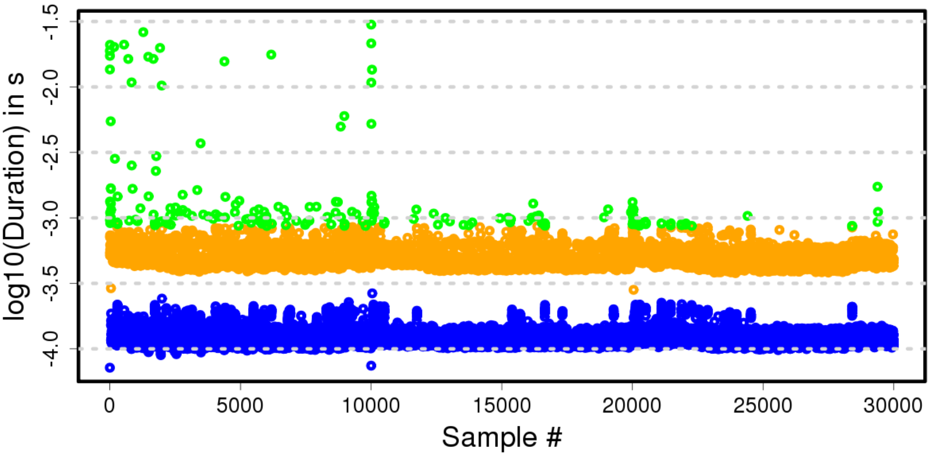
- Best-case benchmark: optimal application I/O
  - Independent I/O with 10 MiB chunks of data
  - Real-world I/O is sparse and behaves worse
- Configurations vary:
  - Number of nodes the benchmark is run
  - Processes per node
  - Read/Write accesses
  - Tunable: stripe size, stripe count
- Optimal performance:
  - Small configuration: 6 GiB/s per node
  - Large configurations: 1.25 GiB/s per node
- Best setting depends on configuration!



A point represents one configuration

Illustration of Performance Variability (2)

- Rerunning the same operation (access size, ...) leads to performance variation
- Individual measurements – 256 KiB sequential reads (outliers purged)



Outline

University of Reading

Research Activities & Interest

University of Reading

High-performance storage for HPC

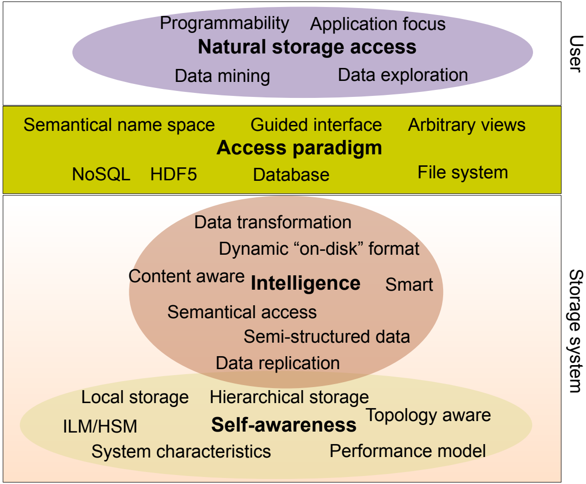
- Efficient I/O
  - Performance analysis methods, tools and benchmarks
  - Optimizing parallel file systems and middleware
  - Modeling of performance and costs
  - Tuning of I/O: Prescribing settings
  - Management of workflows
- Data reduction: compression library, algorithms, methods
- Interfaces: towards domain-specific solutions and novel interfaces

Other research interests

- Application of big data analytics (e.g., for humanities, medicince)
- Domain-specific languages (for Icosahedral climate models)
- Cost-efficiency for data centers in general

Personal Vision: Towards Intelligent Storage Systems and Interfaces

University of Reading



Support Activities

- Community building
  - Bootstrapped: The Virtual Institute for I/O https://www.vi4io.org
  - Supporting: European Open File System (EOFS) https://www.eofs.eu/
  - Organizing: Various I/O workshops
- Awareness: co-created the IO-500 list http://io-500.org
- Beyond teaching:
  - Online teaching platform for C-Programming (ICP project)
  - A **HPC certification program** https://hpc-certification.org
- Standardization:
  - Compression interfaces (AIMES project)
  - Next-Generation I/O Interfaces (https://ngi.vi4io.org)

Outline

Performance Analysis

University of Reading

Problem

Assessing observed time for I/O is difficult.

What best-case performance can we expect?

Support for analysis – my involvement

- Models and simulation
  - Trivial models: using throughput + latency
  - PIOSimHD: MPI application + storage system simulator
- Tools to capture and analyze system statistics and I/O activities
  - HDTrace – tracing tool for parallel I/O (+ PVFS2)
  - SIOX – tool to capture I/O on various levels
  - Grafana – Online monitoring for DKRZ (support)
- Benchmarks – on various levels, e.g., Metadata (md-workbench, IOR)
- **Statistic model** to determine likely cause based on time

I/O Modeling and Diagnosing Causes with Statistics

Issue

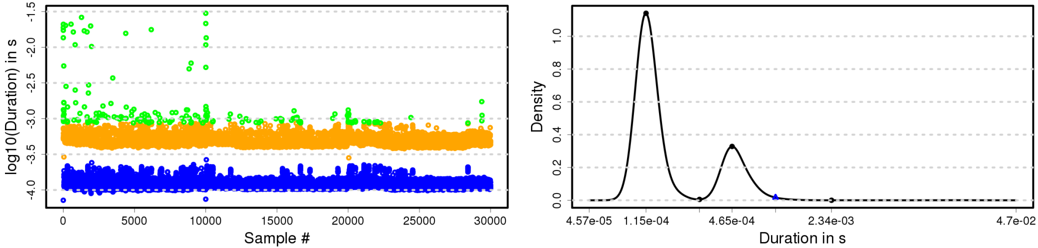
- Measuring the same operation repeatedly results in different runtime
- Reasons:
  - Sometimes a certain optimization is triggered, shortening the I/O path
  - Example strategies: read-ahead, write-behind
- Consequence: Non-linear access performance, time also depends on access size
- It is difficult to assess performance of even repeated measurements!

Goal

- Predict likely reason/cause-of-effect by just analyzing runtime
- Estimate best-case time, if optimizations would work as intended

Comparing Density Plot with the Individual Data Points

University of Reading

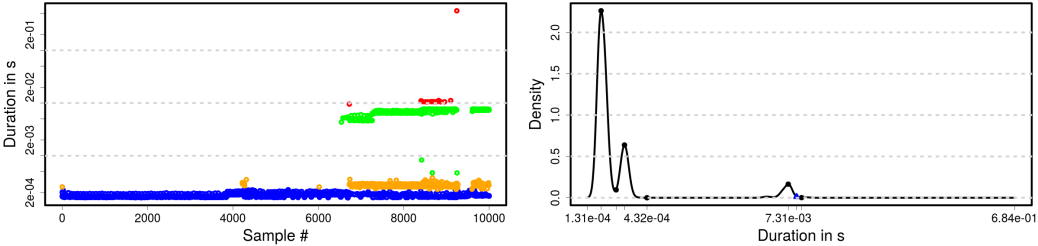


Duration for sequential reads with 256 KiB accesses (off0 mem layout)

Algorithm for determining classes (color schemes)

- Create density plot with Gaussian kernel density estimator
- Find minima and maxima in the plot
- Assign one class for all points between minima and maxima
- Rightmost hill is followed by cutoff (blue) close to zero ⇒ outliers (unexpected slow)

Write Operations



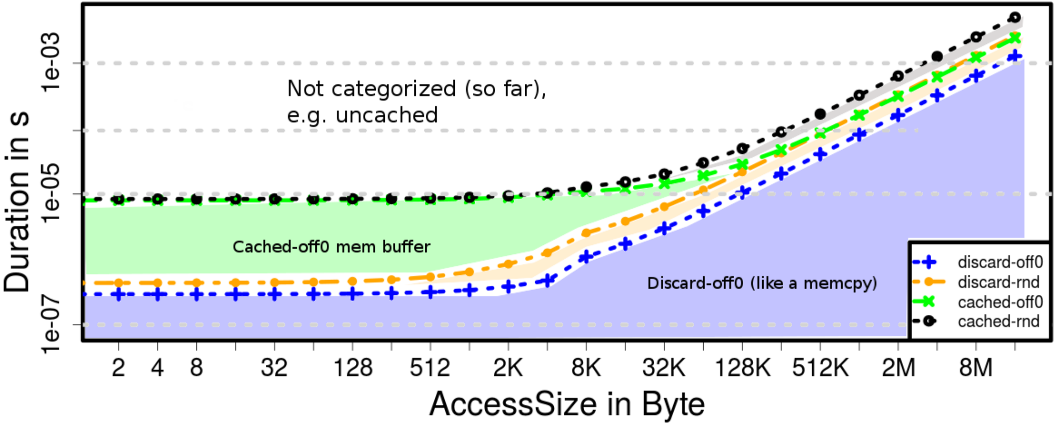
Results for one write run with sequential 256 KiB accesses (off0 mem layout).

Known optimizations for write

- Write-behind: cache data first in memory, then write back
- Write back is expected to be much slower

This behavior can be seen in the figure !

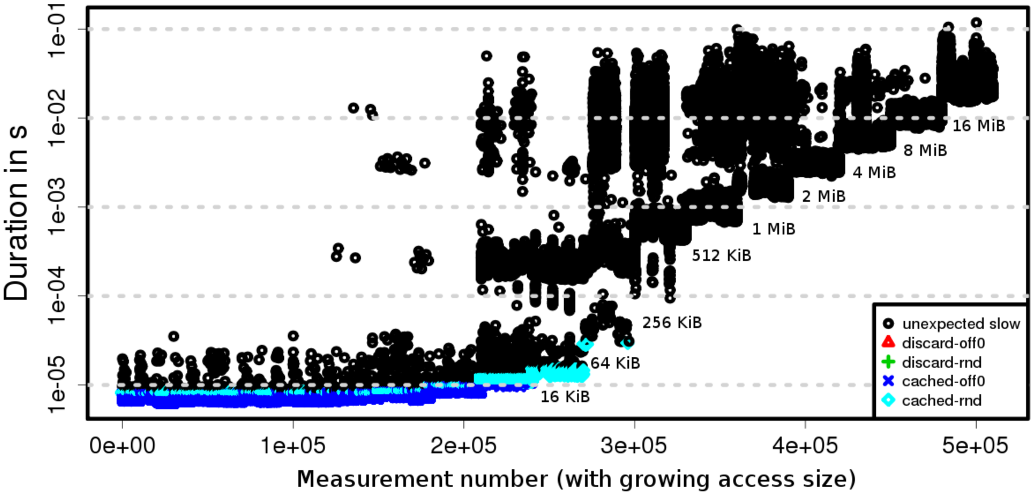
Resulting Performance Models for Read



Read models predicting caching and memory location.

Using the Model to Identify Anomalies

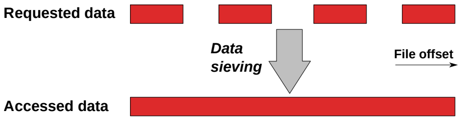
Using the model, the figure for reverse access shows slow-down (by read-ahead)



Outline

Prescriptive Analysis: Learning Best-Practises for DKRZ

- Performance benefit of I/O optimizations is non-trival to predict
- Non-contiguous I/O supports data-sieving optimization
  - Transforms non-sequential I/O to large contiguous I/O
  - Tunable with MPI hints: enabled/disabled, buffer size
  - Benefit depends on system AND application



- Data sieving is difficult to parameterize
  - What should be recommended from a data center's perspective?

Measured Data

- Simple single threaded benchmark, vary access granularity and hole size
- Captured on DKRZ porting system for Mistral
- Vary Lustre stripe settings
  - 128 KiB or 2 MiB
  - 1 stripe or 2 stripes
- Vary data sieving
  - Off or On (4 MiB)
- Vary block and hole size (similar to before)
- 408 different configurations (up to 10 repeats each)
  - Mean arithmetic performance is 245 MiB/s
  - Mean can serve as baseline “model”

System-Wide Defaults

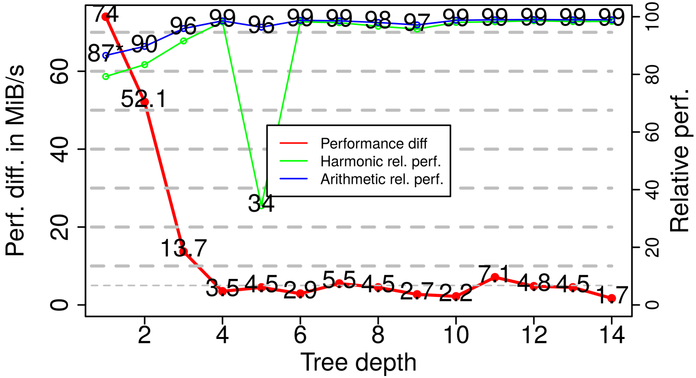
- Comparing a default choice with the best choice
- All default choices achieve 50-70% arithmethic mean performance
- Picking the best default default for stripe count/size: 2 servers, 128 KiB
  - 70% arithmetic mean performance
  - 16% harmonic mean performance ⇒ some bad choices result in very slow performance

Default Choice			Best	Worst	Arithmethic Mean			Harmonic Mean	
Servers	Stripe	Sieving	Freq.	Freq.	Rel.	Abs.	Loss	Rel.	Abs.
1	128 K	Off	20	35	58.4%	200.1	102.1	9.0%	0.09
1	2 MiB	Off	45	39	60.7%	261.5	103.7	9.0%	0.09
2	**128 K**	**Off**	87	76	**69.8%**	209.5	92.7	8.8%	0.09
2	2 MiB	Off	81	14	72.1%	284.2	81.1	8.9%	0.09
1	128 K	On	79	37	64.1%	245.6	56.7	15.2%	0.16
1	2 MiB	On	11	75	59.4%	259.2	106.1	14.4%	0.15
2	**128 K**	**On**	80	58	**68.7%**	239.6	62.6	**16.2%**	0.17
2	2 MiB	On	5	74	62.9%	258.0	107.3	14.9%	0.16

Performance achieved with any default choice

Applying Machine Learning

- Building a classification tree with different depths
- Even small trees are much better than any default
- A tree of depth 4 is nearly optimal; avoids slow cases

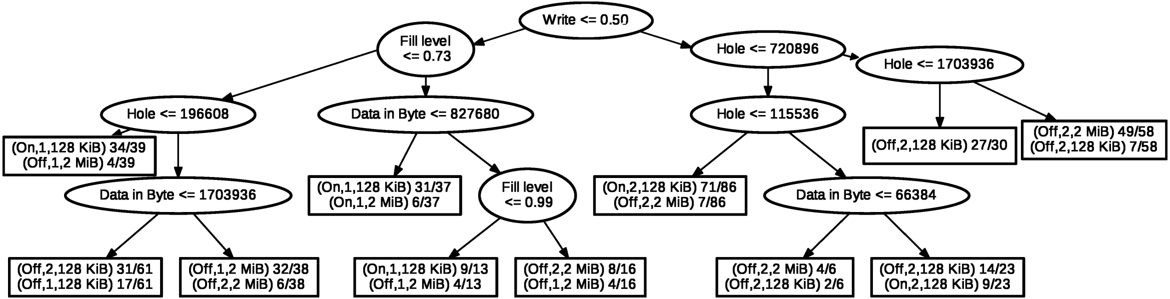


Perf. difference between learned and best choices, by maximum tree depth, for DKRZ's porting system

Decision Tree & Rules

Extraction of knowledge from a tree

- For writes: Always use two servers; For holes below 128 KiB ⇒ turn DS on, else off
- For reads: Holes below 200 KiB ⇒ turn DS on
- Typically only one parameter changes between most frequent best choices



Decision tree with height 4. In the leaf nodes, the settings (Data sieving, server number, stripe size) and number of instances for the two most frequent best choices

Outline

Compression Research: Involvement

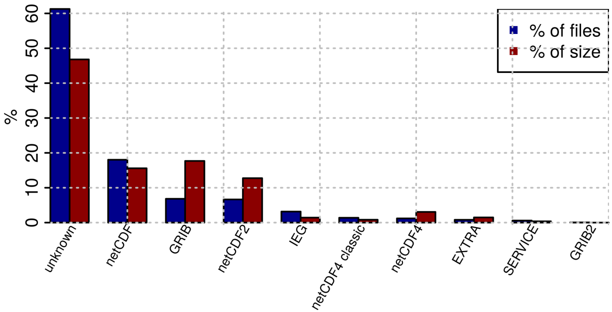
- Development of algorithms for lossless compression
  - MAFISC: suite of preconditioners for HDF5, aims to pack data optimally
    Reduced climate/weather data by additional 10-20%, simple filters are sufficient
- Cost-benefit analysis: e.g., for long-term storage MAFISC pays of
- Analysis of compression 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
- Development of the 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
- A **method for system-wide determination** of data characteristics
  - Method has been integrated into a script suite to scan data centers

Determining Characteristics for Data in a Data Center

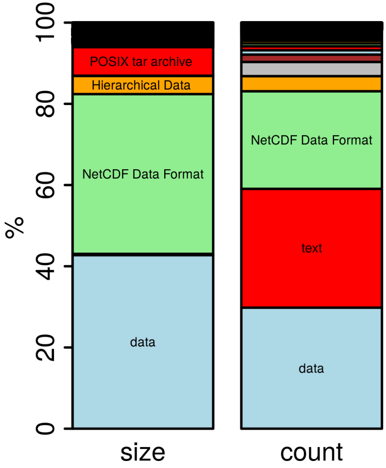
- Data characteristics:
  - Proportion of a given (scientific) file format
  - Compression characteristics (ratio, speeds)
  - Performance behavior when accessing file data (e.g. using alternative I/O)
- Understanding these characteristics is useful
  - Proportions of a file format to identify relevant formats
    - Starting point for optimization of format
  - Conducting what-if analysis on the scale of the data center
    - Estimate the influence storage compression has
    - Performance expectations when applying a new I/O strategy
- Existing studies use a manual selection of “data” for representing stored data
- Conducting analysis on representative data is non-trivial
  - What data makes up a representative data set?
  - How can we infer knowledge for all data based on the subset?
    - Based on file number/count (i.e., a typical file is like X)
    - Based on file size (i.e., 10% of storage capacity is like Y)

Example: Determine Scientific File Formats

- Notice the difference by file count and capacity



(a) Scientific file types



(b) File types according to file magic

Contribution

University of Reading

Goal

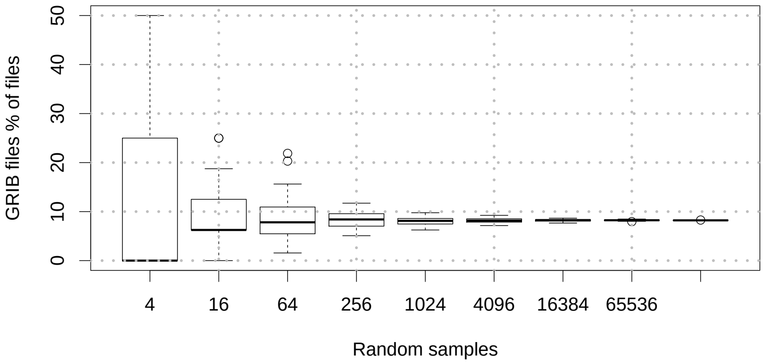
- Design a method based of statistical sampling to estimate file properties
- Conduct a simple study to investigate compression and file types

Approach

1. Scanning a large fraction of data on DKRZ file systems
   - Analyzing file types, compression ratio and speed
2. Investigating characteristics of the data set Filetype, compression ratio, ...
3. Statistical simulation of sampling approaches
   - We assume the population (full data set) is the scanned subset
4. Discuss the estimation error for several approaches

Investigating Robustness: Computing by File Count

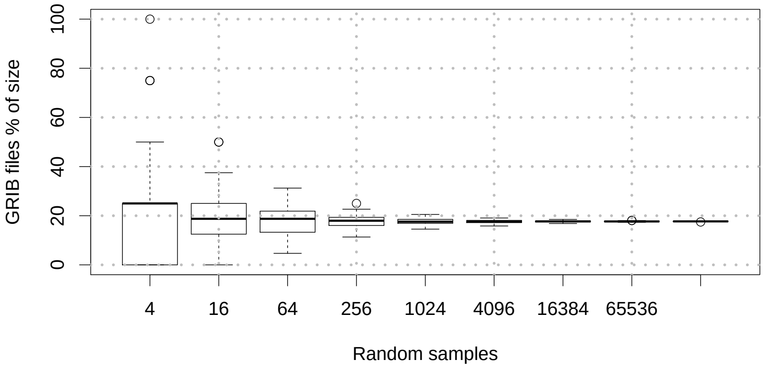
- Running the simulation 100 times to understand the variance of the estimate
- Clear convergence: thanks to Cochran's formula, the total file count is irrelevant



Box plot: Simulation of sampling by file count to compute compr.% by file count

Investigating Robustness: Computing by File Size

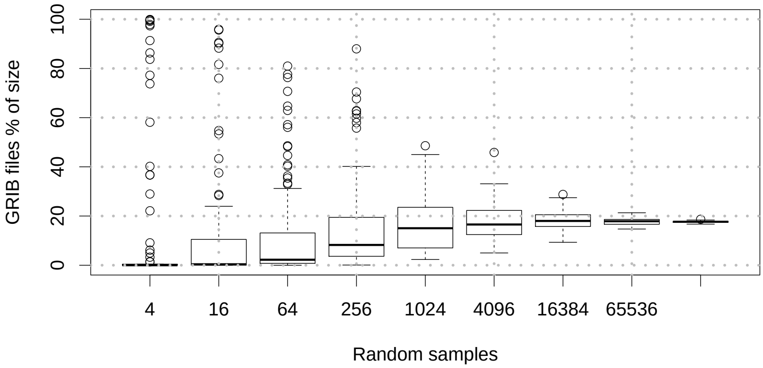
- Using the correct sampling by weighting probability with file size



Simulation of sampling to compute proportions of types by size

Investigating Robustness: Computing by File Size

- Using the WRONG sampling by just picking a simple random sample
- Almost no convergence behavior; you may pick a file with 99% file size at the end



Simulation of sampling to compute proportions of types by size

Selected Algorithms with Good Properties (out of 160+)

University of Reading

Algorithm	Ratio	Compr. MiB/s	Decom. MiB/s
csc33-5	0.485	3.4	16.7
lzlib17-9	0.491	1.4	17.0
xz522-9	0.493	2.1	20.8
lzma938-5	0.493	2.2	24.2
brotli052-11	0.510	0.2	110.6
lzma938-2	0.526	7.9	23.1
zstd100-22	0.526	2.2	294.3
xpack2016-06-02-9	0.548	12.3	282.9
brotli052-5	0.549	16.5	156.6
xpack2016-06-02-6	0.549	16.9	278.9
zstd100-11	0.549	13.8	394.0
zstd100-2	0.574	177.6	455.3
lz4hcr131-16	0.640	3.1	1522.2
lzsse22016-05-14-16	0.640	7.7	1341.6
lz4hcr131-12	0.640	9.4	1519.5
lz4hcr131-9	0.640	17.2	1511.5
lz4hcr131-4	0.649	30.0	1477.8
lz515	0.673	229.2	858.6
density0125beta-2	0.683	419.4	496.5
pithy2011-12-24-9	0.694	305.9	1131.4
lzo1x209-1	0.726	606.7	833.7
lz4r131	0.726	469.8	1893.1
lz4fastr131-3	0.741	646.1	2001.1
lz4fastr131-17	0.772	1132.	2263.1
blosclz2015-11-10-3	0.872	494.4	2612.6
blosclz2015-11-10-1	0.900	819.4	2496.9
memcpy	1.000	4449.	4602.0

WR data

Algorithm	Ratio	Compr. MiB/s	Decom. MiB/s
lzlib17-9	0.426	1.5	22.0
xz522-9	0.427	2.2	24.3
lzma938-5	0.431	2.9	29.1
lzham10-d26-1	0.445	1.4	113.3
csc33-3	0.445	6.5	23.3
brotli052-11	0.451	0.3	124.5
lzma938-0	0.473	13.0	28.2
zstd080-22	0.476	1.1	260.7
brotli052-5	0.489	18.4	165.6
zstd080-18	0.496	3.9	434.4
xpack2016-06-02-9	0.498	19.3	386.8
xpack2016-06-02-1	0.504	53.5	362.0
zstd080-5	0.511	69.4	560.8
brotli052-2	0.512	126.6	168.7
zstd080-2	0.518	220.9	594.0
zstd080-1	0.523	355.0	633.9
lzo1c209-999	0.566	13.5	939.5
lz5hc15-4	0.574	126.3	1410.1
lz515	0.576	326.9	1934.9
lz4hcr131-16	0.577	3.1	2720.6
lz4hcr131-12	0.577	12.4	2700.8
lz4hcr131-9	0.577	28.4	2670.3
lzo1b209-6	0.578	143.3	992.5
lz4r131	0.599	951.4	3037.4
lz4fastr131-3	0.603	1272.	3215.6
pithy2011-12-24-3	0.613	1787.	3535.2
lz4fastr131-17	0.614	1904.	3610.3

DKRZ data

Ongoing Activity: Earth-Science Data Middleware

- Part of the ESiWACE Center of Excellence in H2020
  - Centre of Excellence in Simulation of Weather and Climate in Europe
- ESiWACE2 follow up has been funded!

ESDM provides a transitional approach towards a vision for I/O addressing

- Scalable data management practice
- The inhomogeneous storage stack
- Suboptimal performance & performance portability
- Data conversion/merging

Earth-System Data Middleware

University of Reading

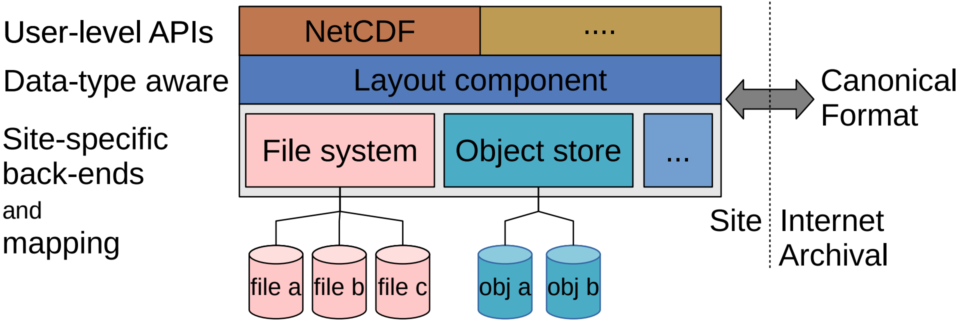
Design Goals of the Earth-System Data Middleware

1. Relaxed access semantics, tailored to scientific data generation
   - Avoid false sharing (of data blocks) in the write-path
   - Understand application data structures and scientific metadata
   - Reduce penalties of **shared** file access
2. Site-specific (optimized) data layout schemes
   - Based on site-configuration and performance model
   - Site-admin/project group defines mapping
   - Flexible mapping of data to multiple storage backends
   - Exploiting backends in the storage landscape
3. Ease of use and deployment particularly configuration
4. Enable a configurable namespace based on scientific metadata

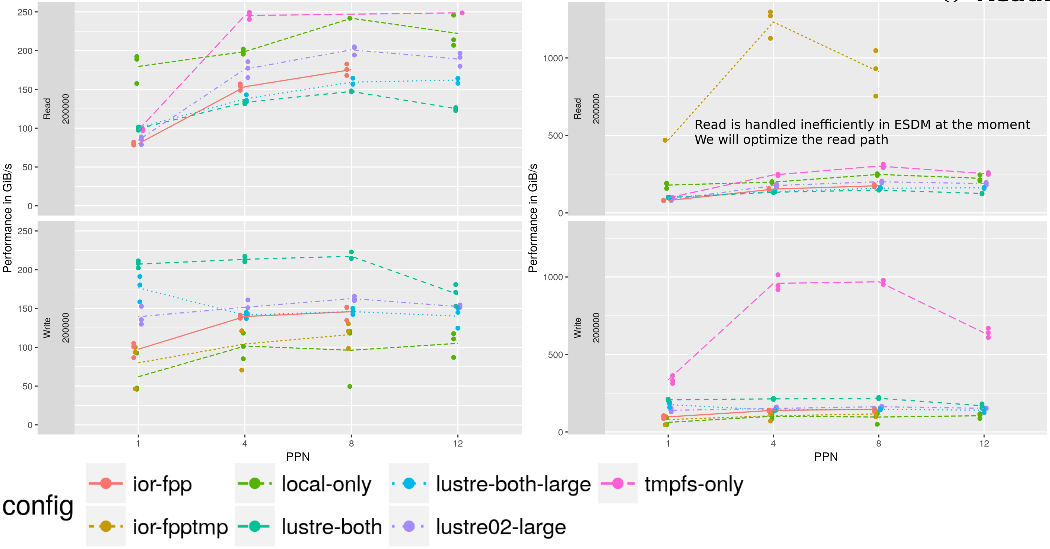
Architecture

Key Concepts

- Middleware utilizes layout component to make placement decisions
- Applications work through existing API (currently: NetCDF library)
- Data is then written/read efficiently; potential for optimization inside library



Measured Performance



ESDM is just the Beginning: Next Generation Interfaces

University of Reading

Towards a new I/O stack considering:

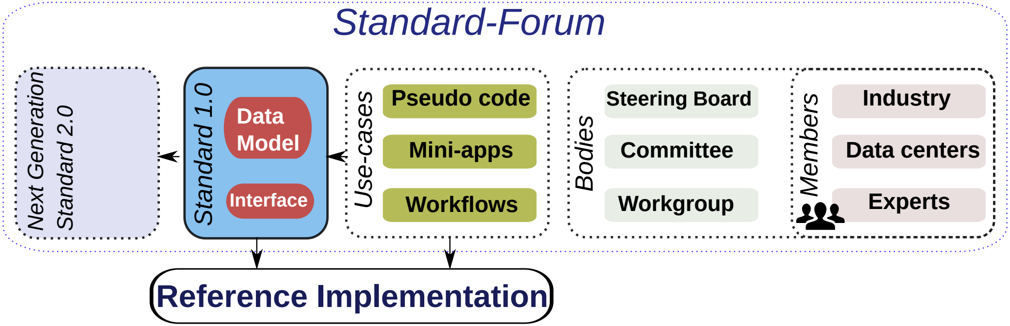
- User metadata and workflows as first-class citizens
- Smart hardware and software components
- Liquid-Computing: Smart-placement of computing
  - Utilizing arbitrary compute and storage technology!
- Self-aware instead of unconscious
- Improving over time (self-learning, hardware upgrades)

Why do we need a new domain-independent API?

- Other domains have similar issues
- It is a hard problem approached by countless approaches
- Harness RD&E effort across domains

Ongoing Community Strategy

- Model targets High-Performance Computing and data-intensive compute
- Goal: Establishing a Forum (similarly to MPI)
- Open board: encourage community collaboration
- *Joint whitepaper will be released before ISC-HPC*



Outline

University of Reading

Perspectives at Cologne

University of Reading

Continuation of ongoing research tracks

- Parallel I/O $\Rightarrow$ efficient I/O
  - Understanding behavior, costs and options
  - Co-design of future I/O interface
  - Smarter processing with **Liquid computing**
  - Data reduction techniques
  - Performance portability
  - **Better systems for data analytics!**
- Big data applications, e.g., humanities
- Domain specific languages for performance portability
- Develop I/O methods for earth-science and beyond

Summary

- Parallel I/O is complex
  - System complexity and heterogeneity increases significantly
  - $\Rightarrow$ Expected and measured performance is difficult to assess
  - HPC users (scientists) and data centers need methods and tools
- Tools, statistics and machine learning help with key aspects:
  - Diagnosing causes and identify anomalies
  - Predicting performance
  - Prescribing best practices
- I work towards intelligent systems to increase insight and ease the burden for users
  - Novel interfaces are needed to unleash the full potential of system resources

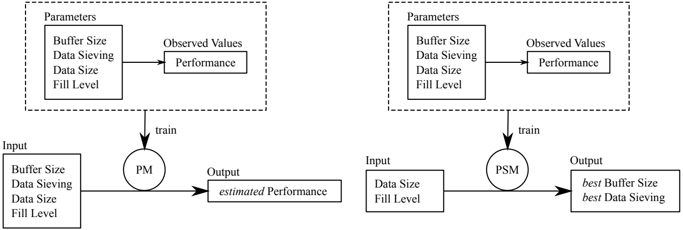
Appendix

Predicting Non-Contiguous I/O Performance

Goal: Predict storage performance based on several parameters and tunables

Alternative models

- Predict performance based on parameters
- Predict best (data sieving) settings



(a) Performance Model (b) Parameter Setting Model

PM provides a perf. estimate, whereas PSM provides the “tunable” variable parameters to achieve it

Validation on Data of the WR Cluster

- Apply k-fold cross-validation
  - Split data into training set and validation set
  - Train model with all (k-1) folds and evaluate it on 1 fold
  - Repeat the process until all folds have been predicted
- A baseline model is the arithmethic mean performance (54.7 MiB/s)
  - Achieves an arithmethic mean error of 28.5 MiB/s
- Linear models yield a mean error of $\geq$ 12.7 MiB/s

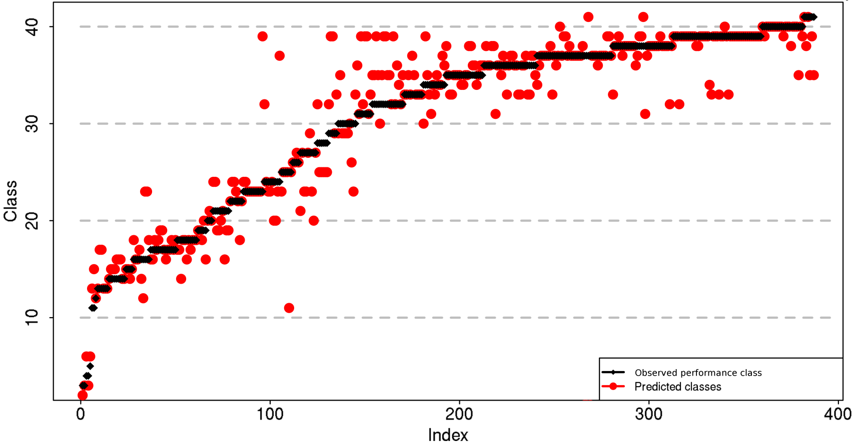
CART results

k	Performance errors in MB/s			Class errors		
	min	mean	max	min	mean	max
2	6.74	6.80	6.87	1.46	1.59	1.72
4	5.19	6.25	6.92	0.94	1.34	1.72
8	4.67	5.66	6.77	0.87	1.19	1.62

Prediction errors for training sets under k-fold cross-validation. Min & max refer to the folds' mean error. Values for k=3..7 lie in between

Comparing Prediction with Observation

University of Reading

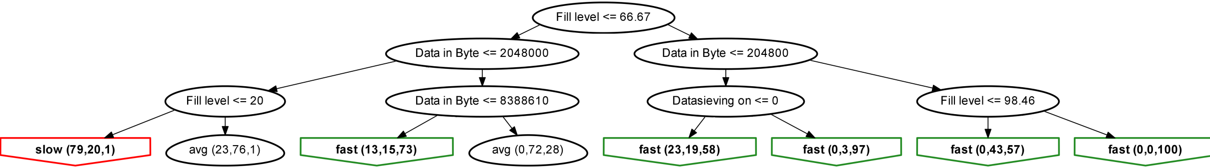


Performance classes and error for k=2, sorted by the observed performance class. Trained by 387 instances, validated on the other 387 instances.

Extracting Knowledge

University of Reading

- Rules can be easily extracted from decision trees
- Consider a performance prediction in three classes
- Rules (this is common sense for I/O experts)
  - Small fill levels and data sizes are slow
  - Large fill levels achieve good performance
- Surprising anomaly: smaller fill level, large access sizes are slower than medium

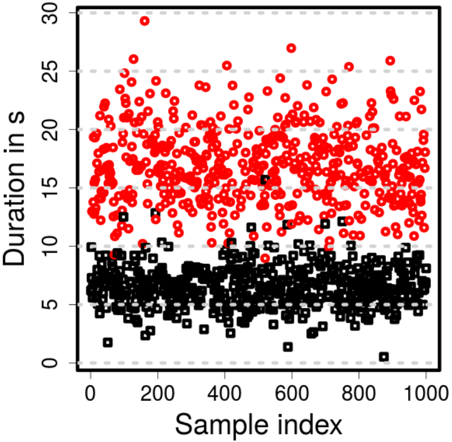


First three levels of the CART classifier rules for three classes slow, avg, fast ($[0, 25]$, $(25, 75]$, > 75 MB/s). The dominant label is assigned to the leaf nodes – the probability for each class is provided in brackets.

Simulation of this Behavior

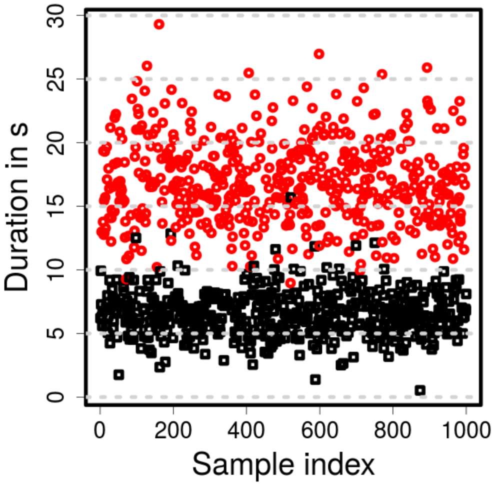
University of Reading

- Assume we have two components
  - Component A is faster than B
  - Either A or B transfer data
  - Cache miss of A leads to transfer for B
- Overlaying 3 stochastic processes:
  - Two gamma distributions with scale=1
  - Normal distribution (little impact)

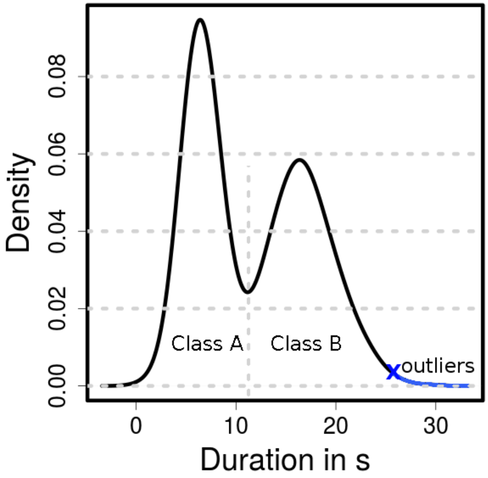


Resulting time for 1000 data points

Simulated Access Time and Resulting Density



(a) Timeline



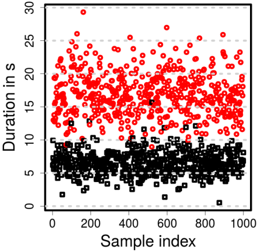
(b) Density reveals two classes

Approach

Assumptions

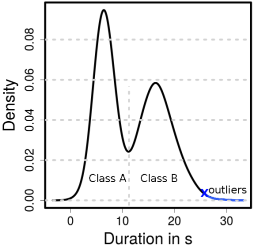
- Each "class" is caused another optimization/technology
  - Assign an observation to the likely class
  - This may lead to (tolerable) errors
- Behavior not visible on the density plot is irrelevant

⇒ The strategy identifies relevant "performance factors"



Concept

1. Repeatedly measure time for I/O with a given size
2. Construct the density graph and identify clusters
3. A class is caused by (at least) one performance factor
4. Build a model to assign the cluster across "sizes"
5. Optional: Identify the root cause for the cluster
6. Assign appropriate names, e.g., "client-side cached"



Approach: Models

- Apply a family of linear models predicting time; $lm(size) = c + f(size)$
  - Assume time correlates to the operation's size
  - Each model represents a condition C (cached, in L1, ...)
  - $t_C(size) = lm(size) + lm'(size) + ...$ and check $min(|t_C - \hat{t}|)$
- Assume the conditions for the closest combination are the cause
- Ignore the fact of large I/O requests with mixed conditions
  - i.e., some time of the operation may be caused by C and some by C'

Example models

- $t(size) = m$: Data is discarded on the client or overwritten in memory
- $t(size) = m + c(size)$: Data is completely cached on the client ...

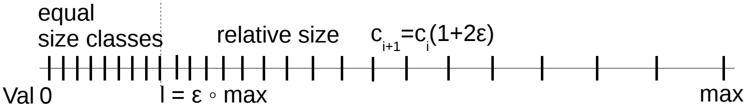
Transformation of the Problem

- Aim to apply alternative methods from machine learning
- Many require classification problems instead of regression

⇒ Performance values need to be mapped into classes

Mapping

- Create 10 classes with the same length up to 5% of max. performance
- Then increase performance range covered by 10% each



Evaluation Data

We analyzed the validity of the approach on two systems

System 1: WR cluster

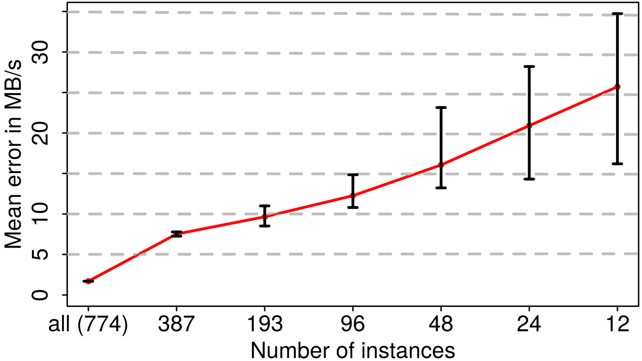
- Lustre 2.5
- 10 server nodes
- 1 Gb Ethernet
- 1 client node (max performance 110 MiB/s)

System 2: DKRZ porting system

- Lustre 2.5 provided by Seagate ClusterStor 9000
- 2 servers
- FDR-Infiniband
- 1 client node (max performance 800 MiB/s)

Investigating Training Set Size

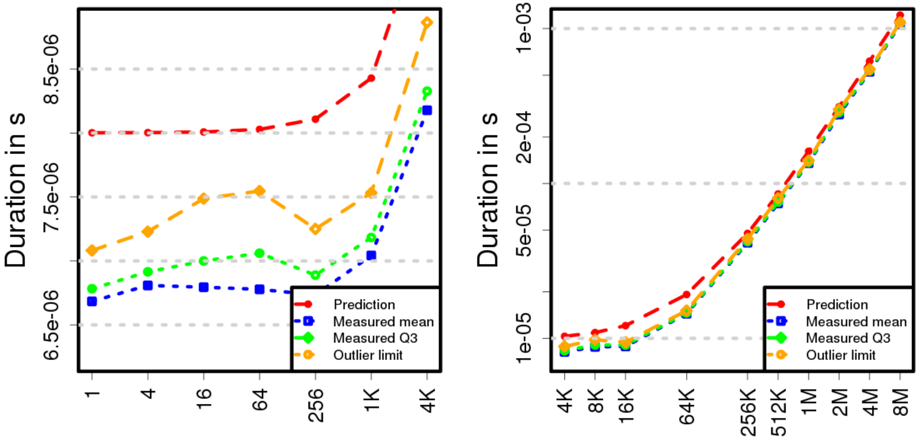
- Inverse k-fold validation: learn from 1 fold and test on (k-1)
- With $\geq$ 96 instances better than the linear model



Mean prediction error of PM by training set size under inverse *k*-fold cross-validation. Class prediction errors show similar behavior

Model for Reading Cached Data

University of Reading



Model accuracy for reading cached data (off0 locality in memory and file). Other figures look similar

Validation: Classify Different Patterns

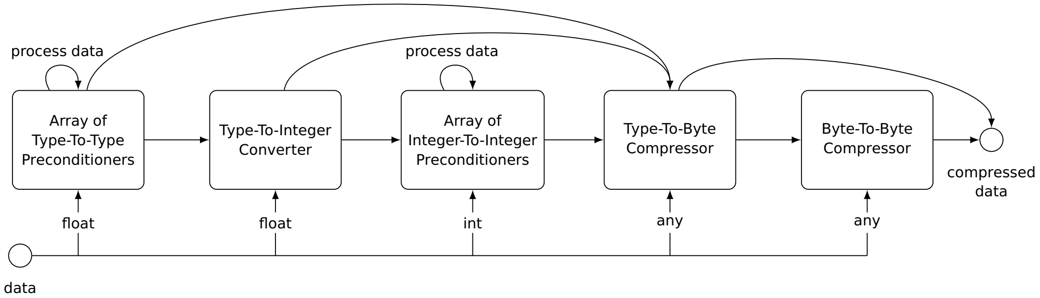
University of Reading

Experiment state-mem-file	cached off0	 rnd	discard off0	 rnd	uncached
D-reverse-off0 R	46	54	0.3	0.03	0.004
C-off0-off0 R	0	34	60	6.1	0.29
C-seq-off0 R	0	0	52	47	0.31
C-seq-reverse R	0	0	42	4.3	54
C-seq-rnd8 R	0	0	30	44	26
C-seq-rnd R	0	0	26	5.6	68
C-seq-seq R	0	0	48	9.5	42
C-seq-stride8,8 R	0	0	28	8.8	63
C-off0-rnd R	0	2e-04	18	1.9	80
U-off0-rnd R	0	0	0.01	0.15	100
U-seq-seq R	0	0	57	6.1	37
C-off0-rnd W	0	0	0	0.003	100
C-off0-seq W W	0	0	40	17	42
C-seq-seq W	0	0	40	12	48
C-off0-reverse W	0	0	71	14	15

Model predictions classes in % of data points for selected memory & file locations – access size is varied.

Architecture of SCIL

- Contains tools to
  - Create random patterns, compress/decompress, add noise, plot
- HDF5 and NetCDF4 integration
- Library offers
  - Automatic algorithm selection (under development)
  - Flexible compression chain:



Analyzing Performance of Lossy Compression using SCIL

University of Reading

Two new (point-wise) algorithms are provided with SCIL

Data

- Single precision (23 bits of significand, 8 bits of exponent, 1 sign bit)
- Synthetic, generated by SCIL's pattern lib.
  - e.g., Random, Steps, Sinus, Simplex
- Data of the variables created by ECHAM
  - The climate model creates up to 123 vars

Experiments

- Single thread, 10 repeats
- Lossless (memcopy and lz4)
- Lossy compression with significant bits (zfp, sigbits, sigbits+lz4)
- Lossy compression with absolute tolerance (zfp, sz, abstol, abstol+lz4)
  - Tolerance: 10%, 2%, 1%, 0.2%, 0.1% of the data maximum value

Comparing Algorithms for the Scientific Files

University of Reading

		Throughput [MiB/s]	
Algorithm	Ratio	Compression	Decompression
sigbits	0.448	462	615
sigbits,lz4	0.228	227	479
zfp-precision	0.299	155	252

Preserving 9 precision bits (instead of 23 from float) ≤ 0.56

		Throughput [MiB/s]	
Algorithm	Ratio	Compression	Decompression
abstol	0.19	260	456
abstol,lz4	0.062	196	400
sz	0.078	81	169
zfp-abstol	0.239	185	301

For absolute tolerance with 1% of max value < 0.22

The harmonic mean has been used

Representative Selection

- Analyzing large quantities of data is time consuming and costly
  - Scanning petabytes of data in $>$ 100 millions of files
  - With 50 PB of data and 5 GiB/s read, 115 node days are needed
  - Scanning DKRZ data with a few compression algorithms cost 4000 €
  - $\Rightarrow$ Working on a representative data set reduces costs

Efficient Sampling Strategies

Sampling to Compute by File Count

1. Enumerate all files
2. Create a simple random sample
   - Select a random number of files to analyze without replacement
   - For proportional variables, the number of files can be computed with Cochran's formula

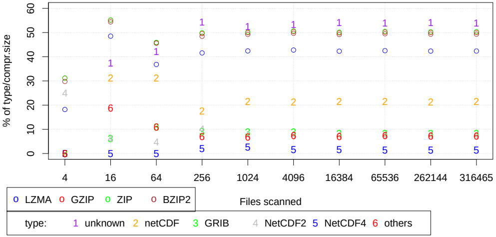
Sampling to Compute by File Size

1. Enumerate all files AND determine their file size
2. Pick a random sample based on the probability $\frac{filesize}{totalsize}$ with replacement
   - Large files are more likely to be chosen (even multiple times)
3. Create a list of unique file names and analyze them
   - Either scan full file (once) or measure feature on a random file section (chunk)
4. Compute the arithmetic mean for the variables
   - If a file has been picked multiple times in Step 2., its value is used multiple times

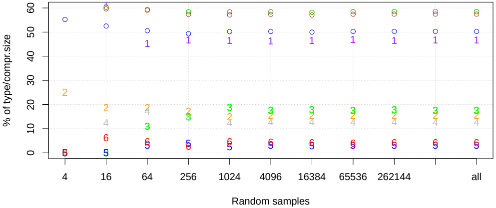
Demonstration of the Strategies

- Apply the approach with an increasing number of samples
  - Compare true value with the estimated value

Running one simulation for increasing sample counts



(a) Compute mean by count



(b) Compute mean by size

Evaluating various metrics (proportions) for an increasing number of samples

- This suggests that the results converge quickly but how trustworthy is one run?

Outline

Challenges in the Domain of climate/weather

- High data volume and velocity
- Data management practice does not scale
  - e.g., hierarchical namespaces does not reflect use cases
  - Scientists spend quite some time to define the namespace
- Suboptimal performance (& performance portability) of data formats
  - Tuning for NetCDF, HDF5 and GRIB necessary
  - Scientists worry about interoperability
- Data conversion is often needed
  - Between formats such as NetCDF and GRIB
  - To combine data from multiple experiments, time steps, ...
- External data services to share data in the community
  - (Scientific) metadata is provided by databases

Benefits

- Expose/access the same data via different APIs
- Independent and lock-free writes from parallel applications
- No fixed storage layout[1]
- Less performance tuning from users needed
- Exploit characteristics of different storage technology
- Multiple layouts of one data structure optimize access patterns
- Flexible namespace (similar to MP3 library)