

Julian Kunkel

# Data Storage



# Learning Objectives

- Sketch a typical I/O stack
- Develop a NetCDF data model for a given use case
- Compare the performance of different storage media
- Sketch application types and access patterns
- Justify the use for I/O benchmarks
- Describe an I/O performance optimization technique
- Describe a strategy for trustworthy benchmark result

# Outline

- 1 Introduction
- 2 NetCDF
- 3 Monitoring I/O
- 4 Benchmarking
- 5 Optimizations
- 6 Other
- 7 Outlook
- 8 Summary

# Reminder: High-Performance Computing (HPC)

## Definitions

- HPC: Field providing massive compute resources for a computational task
  - ▶ Task needs too much memory or time on a normal computer
  - ⇒ Enabler of complex scientific simulations, e.g., weather, astronomy
- Supercomputer: aggregates power of 10,000 compute devices
- Storage system: provides some kind of storage with some API
- File system: provides a hierarchical namespace and “file” interface
- Parallel I/O: multiple processes can access distributed data concurrently

# Supercomputers Host Costly Storage

- Compute performance growth by 20x each generation (~5 years). Real Values – 2018
- Storage throughput/capacity improves by just 6x.

Exascale Storage Systems – An Analytical Study of Expenses

	2004	2009	2015	2020	2025	Exascale (2020)
Performance	1.5 TF/s	150 TF/s	3 PF/s	60 PF/s	1.2 EF/s	1 EF/s
Nodes	24	264	2500	12,500	31,250	100k-1M
Node performance	62.5 GF/s	0.6 TF/s	1.2 TF/s	4.8 TF/s	38.4 TF/s	1-15 TF/s
System memory	1.5 TB	20 TB	170 TB	1.5 PB	12.8 PB	3.6-300 PB
Storage capacity	100 TB	5.6 PB	45 PB	270 PB	1.6 EB	0.15-18 EB
Storage throughput	5 GB/s	30 GB/s	400 GB/s	2.5 TB/s	15 TB/s	20-300 TB/s
Disk drives	4000	7200	8500	10000	12000	100k-1000k
Archive capacity	6 PB	53 PB	335 PB	1.3 EB	5.4 EB	7.2-600 EB
Archive throughput	1 GB/s	9.6 GB/s	21 GB/s	57 GB/s	128 GB/s	-
Power consumption	250 kW	1.6 MW	1.4 MW	1.4 MW	1.4 MW	20-70 MW
Investment	26 M€	30 M€	30 M€	30 M€	30 M€	\$200 M <sup>4</sup>

	Mistral
Characteristics	Value
Performance	3.1 PF/s
Nodes	2882
Node performance	1.0 TF/s
System memory	200 TB
Storage capacity	52 PB
Storage throughput	700 GB/s
Disk drives	10600
Archive capacity	500 PB
Archive throughput	18 GB/s
Compute costs	15.75 M EUR
Network costs	5.25 M EUR
Storage costs	7.5 M EUR
Archive costs	5 M EUR
Building costs	5 M EUR
Investment	38.5 M EUR
Compute power	1100 kW
Network power	50 kW
Storage power	250 kW
Archive power	25 kW
Power consumption	1.20 MW

# Application Data vs. File

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
- Vertical data access unpractical (e.g., to pick a slice of multiple files)

# 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
- Parallel file system: Lustre, GPFS, PVFS2
- File system: EXT4, XFS, NTFS
- Block device: utilizes storage media to export a block API
- Operating system: (orthogonal aspect)

Application

Middleware

MPI-IO / POSIX

Parallel File Systems

File Systems

Block device

Figure: Example I/O stack

# Storage Media

- Many technologies are available with different characteristics
- Block-accessible or byte-addressable (NVRAM)

	Memristor	PCM	STT-RAM	DRAM	Flash	HD
Chip area per bit (F <sup>2</sup> )	4	8–16	14–64	6–8	4–8	n/a
Energy per bit (pJ) <sup>2</sup>	0.1–3	2–100	0.1–1	2–4	10 <sup>1</sup> –10 <sup>4</sup>	10 <sup>6</sup> –10 <sup>7</sup>
Read time (ns)	<10	20–70	10–30	10–50	25,000	5–8x10 <sup>6</sup>
Write time (ns)	20–30	50–500	13–95	10–50	200,000	5–8x10 <sup>6</sup>
Retention	>10 years	<10 years	Weeks	<Second	~10 years	~10 years
Endurance (cycles)	~10 <sup>12</sup>	10 <sup>7</sup> –10 <sup>8</sup>	10 <sup>15</sup>	>10 <sup>17</sup>	10 <sup>3</sup> –10 <sup>6</sup>	10 <sup>15</sup> ?
3D capability	Yes	No	No	No	Yes	n/a

Figure: Source: ZDNet [100]



# Zoo of Interfaces

## 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 – a workflow may involve several

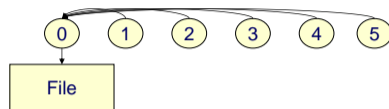
## Properties / qualities

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

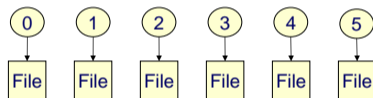
# Application I/O Types

## Serial, multi-file parallel and shared file parallel I/O

Serial I/O



Parallel Multi-file I/O



Parallel Shared-file I/O

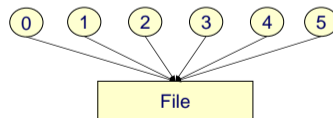
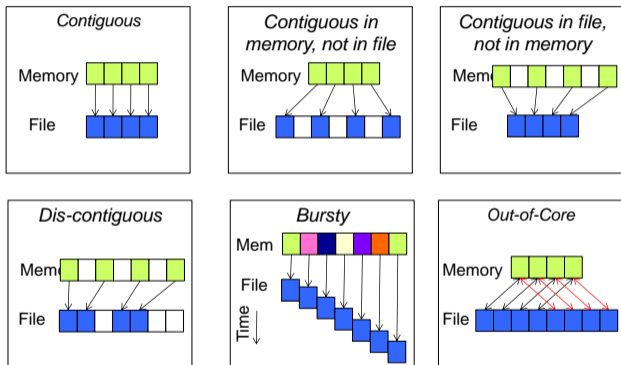


Figure: Source: Lonnie Crosby [101]

# Application I/O Access Patterns

## Access Patterns



# File Striping: Distributing Data Across Devices

## File Striping: Physical and Logical Views

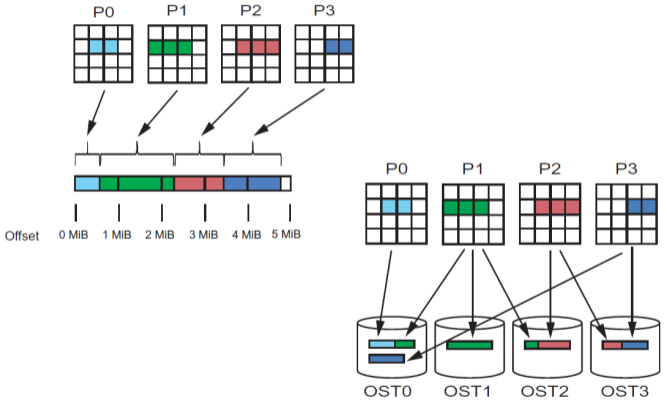


Figure: Source: Lonnie Crosby [101]

# Parallel I/O Efficiency

- I/O intense science requires good I/O performance
- DKRZ file systems offer about 700 GiB/s throughput
  - ▶ However, I/O operations are typically inefficient: Achieving 10% of peak is good
  - ▶ Unfortunately, prediction of performance is barely possible
- Influences on I/O performance
  - ▶ Application's access pattern and usage of storage interfaces
  - ▶ Communication and slow storage media
  - ▶ Concurrent activity – shared nature of I/O
  - ▶ Tenable optimizations deal with characteristics of storage media
  - ▶ Complex interactions of these factors
- The I/O hardware/software stack is very complex – even for experts
- Requires tools and methods for
  - ▶ diagnosing causes
  - ▶ predicting performance, identification of slow performance
  - ▶ prescribing tunables/settings

# Illustration of Performance Variability

- Measured at DKRZ (max. 700 GiB/s)
- Optimal performance:
  - ▶ Small configuration: 6 GiB/s per node
  - ▶ Large configurations: 1.25 GiB/s per node
- Best-case benchmark: optimal application I/O
  - ▶ Independent I/O with 10 MiB chunks of data
  - ▶ Real-world I/O is sparse and worse
- Configurations on user-side vary:
  - ▶ Number of nodes the benchmark is run
  - ▶ Processes per node
  - ▶ Read/Write accesses
  - ▶ Tunable: stripe size, stripe count
- Best setting depends on configuration!

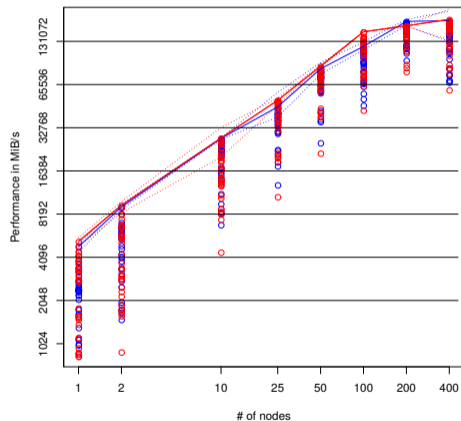


Figure: A point represents one configuration

# Outline

- 1 Introduction
- 2 NetCDF**
- 3 Monitoring I/O
- 4 Benchmarking
- 5 Optimizations
- 6 Other
- 7 Outlook
- 8 Summary

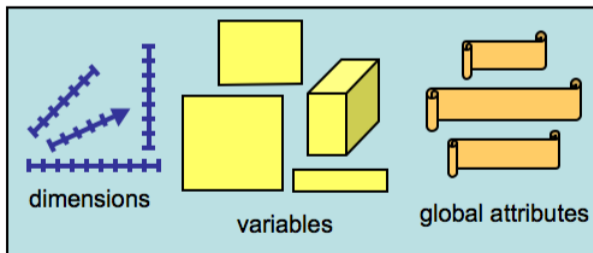
# NetCDF

- NetCDF is an example for a "high-level" I/O-API and ecosystem
- In a simple view, NetCDF is:
  - ▶ A data model
  - ▶ A file format
  - ▶ A set of APIs and libraries for various programming languages
- Together, the data model, file format, and APIs support
  - ▶ creation, access, and **sharing** of scientific data
- Allows to describe multidimensional data and include metadata which further characterizes the data
- APIs are available for most programming languages used in geo-sciences



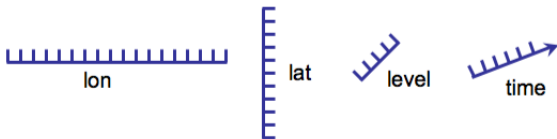
# The Classic NetCDF Model

- NetCDF files are containers for Dimensions, Variables, and Global Attributes.
- A NetCDF file (dataset) has a path name and possibly some dimensions, variables, global (file-level) attributes, and data values associated with the variables.



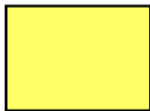
# The Classic NetCDF Model – Dimensions

- Dimensions are used to specify variable shapes, grids, and coordinate systems.
- A dimension has a name and a length.
- A dimension can be used to represent a real physical dimension
  - ▶ Example: time, latitude, longitude, or height
- A dimension can also be used to index other quantities
  - ▶ Example: station or model run number
- The same dimension can be used in multiple variables.

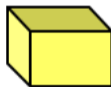


# The Classic NetCDF Model – Variables

- A variable holds a multidimensional array of values of the same type
- A variable has a name, type, shape (according to dimensions), attributes, and values
- In the classic data model, the type of a variable is the external type of its data as represented on disk, one of: char (text character), byte (8 bits), short (16 bits), int (32 bits), float (32 bits), double (64 bits)



sst



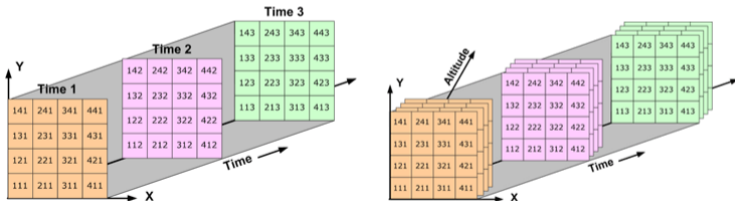
relative\_humidity



time

# The Classic NetCDF Model – Data

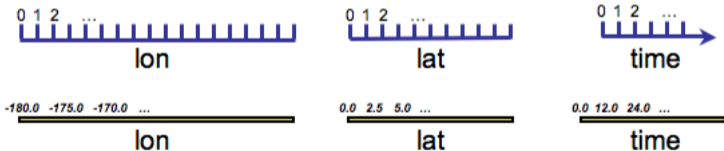
- The data in a NetCDF file is stored in the form of arrays. For example:
  - ▶ Temperature varying over time at a location is stored as a **one-dimensional array**
  - ▶ Temperature over an area for a given time is stored as a **two-dimensional array**
  - ▶ Three-dimensional (3D) data, like temperature over an area varying with time, or four-dimensional (4D) data, like temperature over an area varying with time and altitude, is stored as a **series of two-dimensional arrays**



Reference: <https://pro.arcgis.com/en/pro-app/help/data/multidimensional/fundamentals-of-netcdf-data-storage.htm>

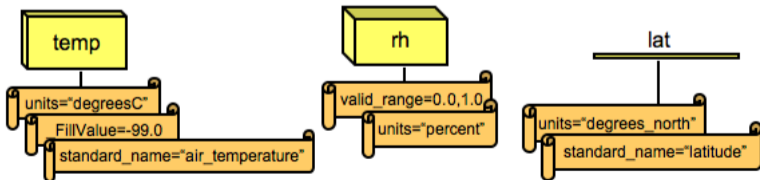
## The Classic NetCDF Model – Coordinate Variables

- A 1D variable with the same name as a dimension is a **coordinate variable**
- The coordinate variable is associated with a dimension of one or more data variables and typically defines a physical coordinate corresponding to that dimension
- Many programs that read NetCDF files recognize coordinate values they find



# The Classic NetCDF Model – Attributes

- Attributes hold metadata (data about data)
- An attribute contains information about properties of a variable or the whole dataset
- Attributes are scalars or 1-D arrays
- An attribute has a name, type, and values. Attributes are used to specify such properties as units, standard names (that identify types of quantity), special values, maximum and minimum valid values, scaling factors, offsets, ...



# Common Data form Language (CDL)

- Notation used to describe NetCDF object is called CDL (network Common Data form Language)
  - ▶ Provides a convenient way of describing NetCDF datasets
- Utilities allow producing CDL text files from binary NetCDF datasets and vice-versa
- File contains dimensions, variables, and attributes
- Components are used together to capture the meaning of data and relations among data fields

```

netcdf filename {
dimensions:
  lat = 3 ;
  lon = 4 ;
  time = UNLIMITED ; // (2 currently)

variables:
  float lat(lat) ;
    lat:long_name = "Latitude" ;
    lat:units = "degrees_north" ;
  float lon(lon) ;
    lon:long_name = "Longitude" ;
    lon:units = "degrees_east" ;
  int time(time) ;
    time:long_name = "Time" ;
    time:units = "days since 1895-01-01" ;
    time:calendar = "gregorian" ;
  float rainfall(time, lat, lon) ;
    rainfall:long_name = "Precipitation" ;
    rainfall:units = "mm yr-1" ;
    rainfall:missing_value = -9999.f ;

// global attributes:
  :title = "Historical Climate Scenarios" ;
  :Conventions = "CF-1.0" ;

data:
  lat = 48.75, 48.25, 47.75;
  lon = -124.25, -123.75, -123.25, -122.75;
  time = 364, 730;
  rainfall =
    761, 1265, 2184, 1812, 1405, 688, 366, 269, 328, 455, 524, 877,
    1019, 714, 865, 697, 927, 926, 1452, 626, 275, 221, 196, 223;
}

```

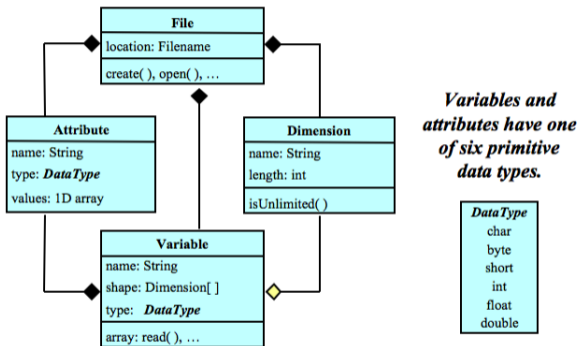
Coordinate variable

Variable attribute

Global attribute

# The Classic NetCDF Model – UML Diagram

- The classic NetCDF can be represented in an UML diagram



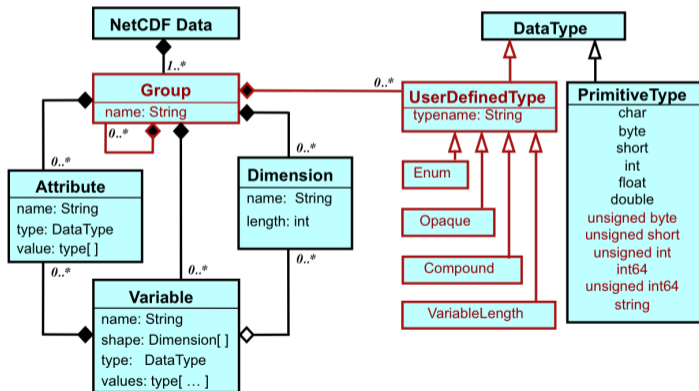
*A file has named variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One dimension may be of unlimited length.*

Figure: Source [102]: NetCDF UML Diagram



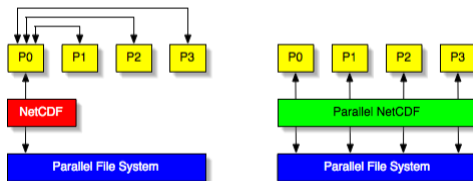
# NetCDF Data Models

- Classic: Simplest model – Dimensions, variables, attributes
- **Enhanced: More powerful model – Adds groups, types, nesting**



## Parallel I/O in NetCDF-4

- NetCDF-4 provides parallel file access to both classic and NetCDF-4/HDF5 files
- The parallel I/O to classic files is achieved through PNetCDF while parallel I/O to NetCDF-4 files is through HDF5 or ESDM, ZARR format support is coming
- NetCDF-4 exposes the parallel I/O features of HDF5
  - ▶ HDF5 provides easy-to-use parallel I/O feature



# Outline

- 1 Introduction
- 2 NetCDF
- 3 Monitoring I/O**
- 4 Benchmarking
- 5 Optimizations
- 6 Other
- 7 Outlook
- 8 Summary

# Understanding of I/O Behavior and Systems

## How can we understand system behavior?

### ■ Observation

- ▶ Measurement of runs on the system
- ▶ Can be many cases to run
- ▶ Slight bias since measurement perturbs behavior
- ▶ Benchmarking: applications geared to exhibit certain system behavior

### ■ Monitoring: system/tool-provided observation creation

### ■ Theory: Performance models

- ▶ Used to determine performance for a system/workload
- ▶ Behavioral models  
Build models based on ensemble of observations

### ■ System/application simulation

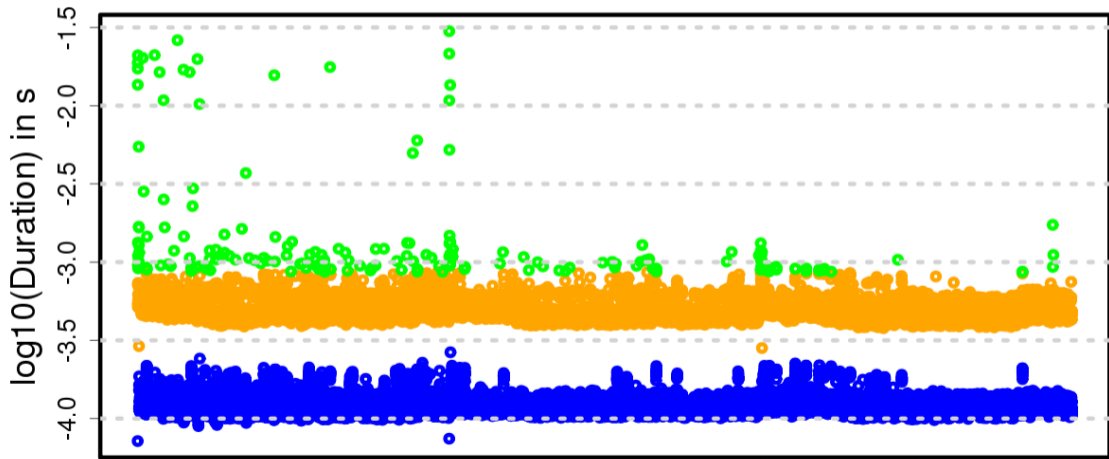
- ▶ Based on system and workload models

# Monitoring I/O

- To understand variability better, must analyze and understand behavior
- We need to capture I/O behavior, options
  - ▶ System-level, i.e., analyze OS-observable statistics such as bytes read
  - ▶ Application-level, record individual operations performance
- There are many interesting metrics that can be recorded
- Many tools exists that aid this analysis

# Performance Variability for Single Operations

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



# Understanding Performance Variability

## Issue

- Measuring 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

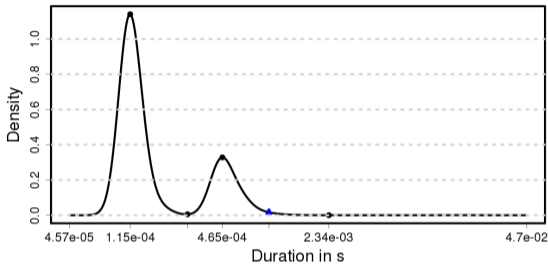
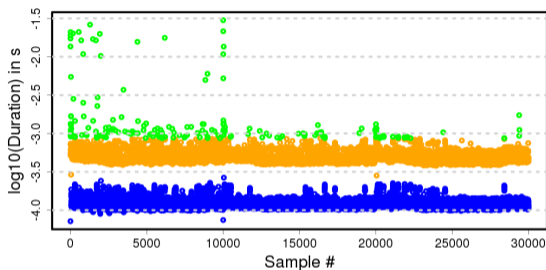


Figure: Duration for sequential reads with 256 KiB accesses (off0 mem layout)

## 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  $\Rightarrow$  outliers (unexpected slow)



# Write Operations

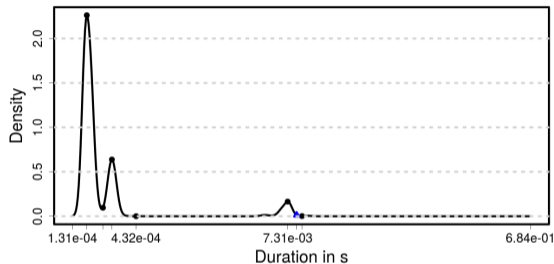
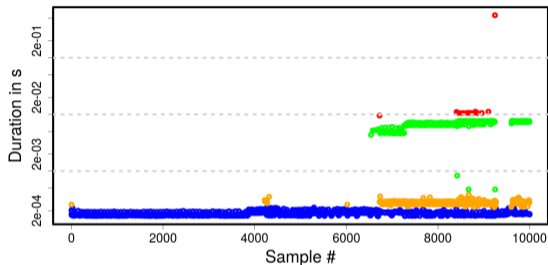


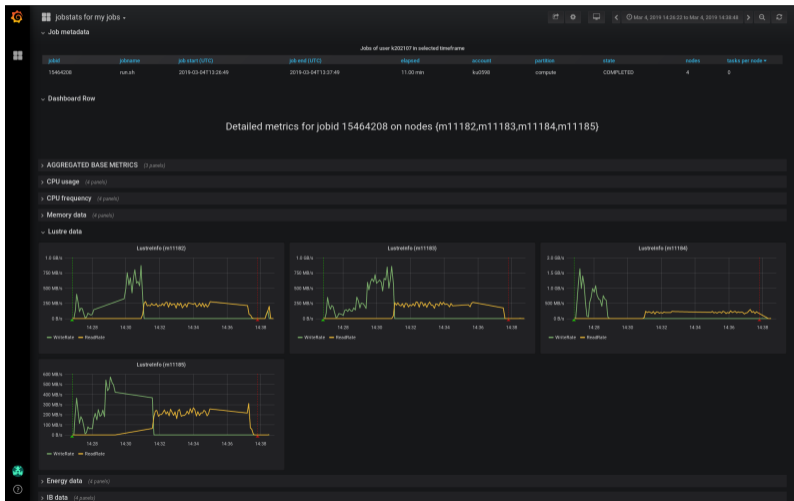
Figure: Results for one write run with sequential 256 KiB accesses (off0 mem layout).

## 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 !

# System-Wide Monitoring



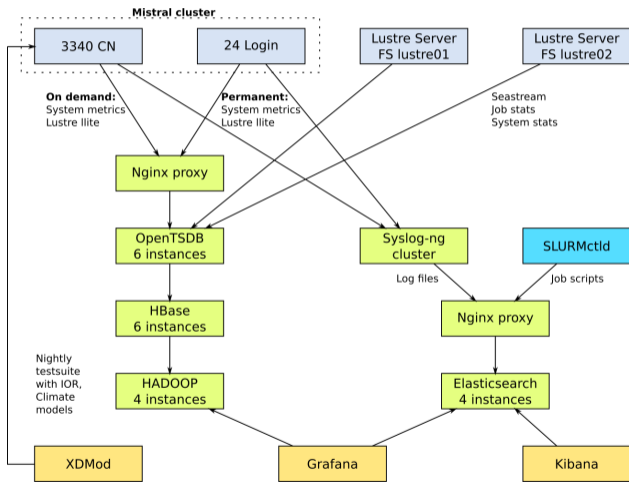
## ■ Grafana visualization

## ■ Read/write shown

## ■ Metrics supported

- ▶ md\_file\_create
- ▶ md\_file\_delete
- ▶ md\_read (only)
- ▶ md\_mod(ify)
- ▶ md\_other
- ▶ read\_bytes
- ▶ read\_calls
- ▶ write\_bytes
- ▶ write\_calls

# DKRZ Monitoring System



## Details

- Periodicity: 10s
- Record metrics
  - ▶ From /proc
  - ▶ 9 aggregated
- Jobs are linked to the data

## Mistral Supercomputer

- 3,340 Nodes
- 2 Lustre file systems
- 52 PByte capacity
- 100+ OSTs per fs

# Visualizing Job Behavior and Comparing different jobs

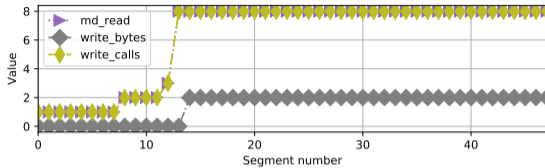
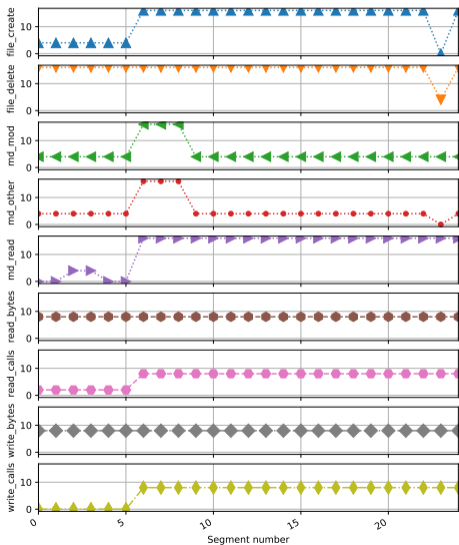


Figure: For this job, other metrics == 0

- Different jobs differ significantly
- We can compare jobs
- Metrics categorized based on categories
  - ▶ 0 = non-IO
  - ▶ 1 = intense
  - ▶ 4 = extreme
- Segments represent 10 min

# Outline

- 1 Introduction
- 2 NetCDF
- 3 Monitoring I/O
- 4 Benchmarking**
- 5 Optimizations
- 6 Other
- 7 Outlook
- 8 Summary

# How Can Benchmarks Help to Analyze I/O?

## ■ Benefits of benchmarks

- ▶ Can use simple/understandable sequence of operations
  - Ease comparison with theoretic values (that requires understandable metrics)
- ▶ May use a pattern like a realistic workloads
  - Provides performance estimates or bounds for workloads!
- ▶ Sometimes only possibility to understand hardware capabilities
  - Because the theoretic analysis may be infeasible

## ■ Benefits of benchmarks vs. applications

- ▶ Are easier to code/understand/setup/run than applications
- ▶ Come with less restrictive "license" limitations

## ■ Flexible testing (strategies)

- ▶ Single-shot: e.g., acceptance test
- ▶ Periodically: regression tests

# Benchmarks

- Benchmarks measure system behavior and implement (simple) well-known behavior
- Many I/O benchmarks exist covering various aspects
  - ▶ APIs used
  - ▶ Data access pattern
  - ▶ Memory access pattern
  - ▶ Parallelism and concurrency
- Let's talk about the IO-500 benchmark suite; it is
  - ▶ **Representative**: for optimized and naive workloads
  - ▶ **Inclusive**: cover various storage technology and non-POSIX APIs
  - ▶ **Trustworthy**: representative results and prevent cheating
  - ▶ **Cheap**: easy to run and short benchmarking time (in the order of minutes)
  - ▶ Favors a single metric to simplify the comparison across dimensions

# Goals of the IO-500 Benchmarking Effort

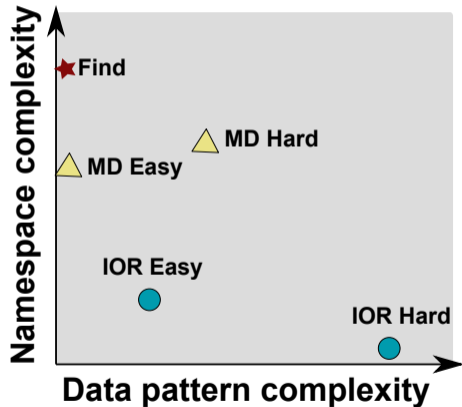
- Bound performance expectations for realistic workloads
- Track storage system characteristics behavior over the years
  - ▶ Foster understanding of storage performance development
  - ▶ Support to identify potent architectures for certain workloads
- Document and share best practices
  - ▶ Tuning of the system is encouraged
  - ▶ Submitters must submit detailed run parameters
- Support procurements, administrators and users

<https://io500.org>

**IO<sup>500</sup>**



# Covered Access Patterns



- IOR-easy: large seq on file(s)
- IOR-hard: small random shared file
- MD-easy: mdtest, per rank dir, empty files
- MD-hard: mdtest, shared dir, 3900 byte
- find: query and filter files based on name and creation time
- Executing concurrent patterns not covered (another dimension)

# Predictability and Latency Matters

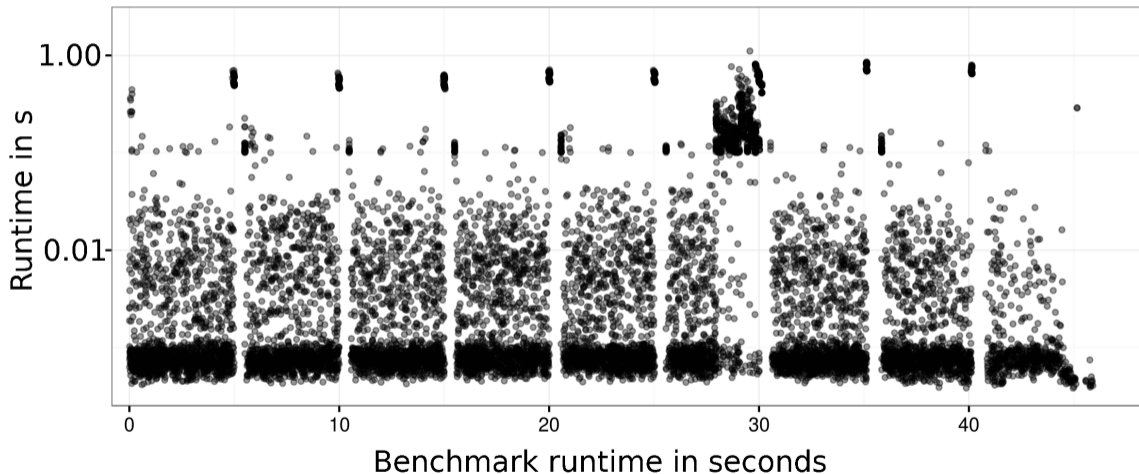
## Performance Predictability

- How long does an I/O / metadata operation take?
- Important to predict runtime
- Important for bulk-synchronous parallel applications
  - ▶ The slowest straggler defines the performance

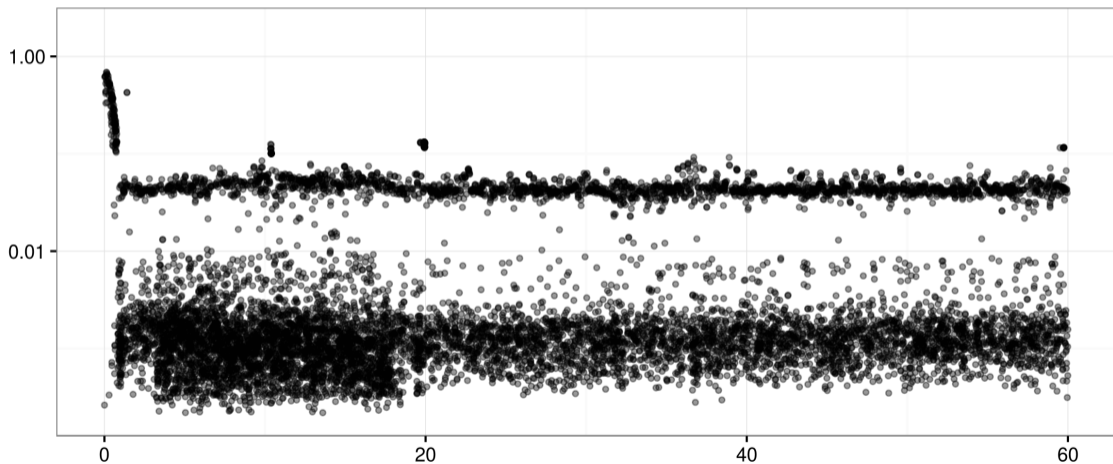
## Measurement

- In the following, we plot the timelines of metadata create operations
  - ▶ Sparse plot with randomly selected measurements
  - ▶ Every point above 0.1s is added
- All results obtained on 10 Nodes using MD-Workbench  
<https://github.com/JulianKunkel/md-workbench>
  - ▶ Options: 10 PPN, D=1, I=2000, P=10k, precreation phase

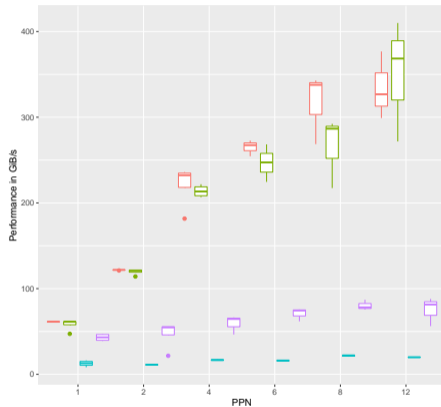
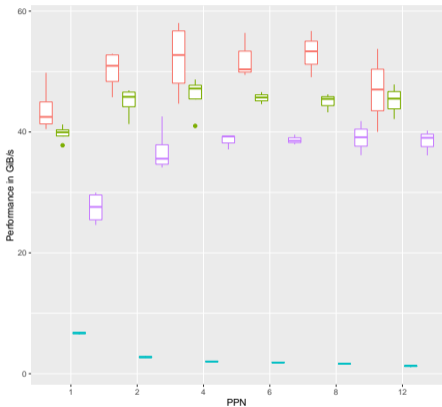
# Latencies: Lustre / Mistral at DKRZ



# Latencies: GPFS / Cooley at ALCF



# Performance of the NetCDF-Bench 100 Nodes@Mistral



Write

Read

■ Better performance than FPP but looks for users like a single file

config esdm-lustre-both esdm-lustre01 nc nc-fpp

config esdm-lustre-both esdm-lustre01 nc nc-fpp

# Importance of Choosing the Right Mean Value

- We must repeat a benchmark run to obtain trustworthy data
  - ▶ Reduce impact of random errors due to background activity
- How do we weight input when repeating a benchmark run?

## Tuning for improving the Geom-Mean value

Description	Input (11 values)	Geom	Arithmetic	Harmonic
Balanced system	10 ... 10 10 <b>10</b>	10	10	10
One slow bench	10 ... 10 10 <b>1</b>	8.1	9.2	5.5
Tuning worst 2x	10 ... 10 10 <b>2</b>	8.6	9.3	7.3
Tuning good 2x	10 ... 10 <b>20 1</b>	8.6	10.1	5.6
Tuning good 100x	10 ... 10 <b>100 1</b>	10	17.4	5.8

- Avoid arithmetic mean
- May use box-plots to visualize variability
- Geom mean honors tuning equally, insensitive to “outliers”
- Harmonic mean favors balanced systems (complex to scale results)

# Probing Approach

- Many sites run periodic regression tests, e.g., nightly
  - ▶ Helps to identify performance regressions with updates
- Instead, we run a non-invasive benchmark (a probe) with a high frequency
  - ▶ Mimic the user-visible client behavior
  - ▶ Measuring latency for metadata and data operations
- Generate and analyze generated statistics
- Derive a slowdown factor (file system load)

# Probing: Performance Measurement

## Preparation

- Data: Generate a large file (e.g.,  $> 4x$  main memory of the client)
- Metadata: Pre-create a large pool of small files (e.g., 100k+ files)

## Benchmarks

- Repeat the execution of the two patterns every second
- DD: Read/Write a random 1 MB block
- MD-Workbench: stat, read, delete, write a single file per iteration
  - ▶ Allows regression testing, i.e., retain the number of files
  - ▶ *J. Kunkel, G. Markomanolis. Understanding Metadata Latency with MDWorkbench.*

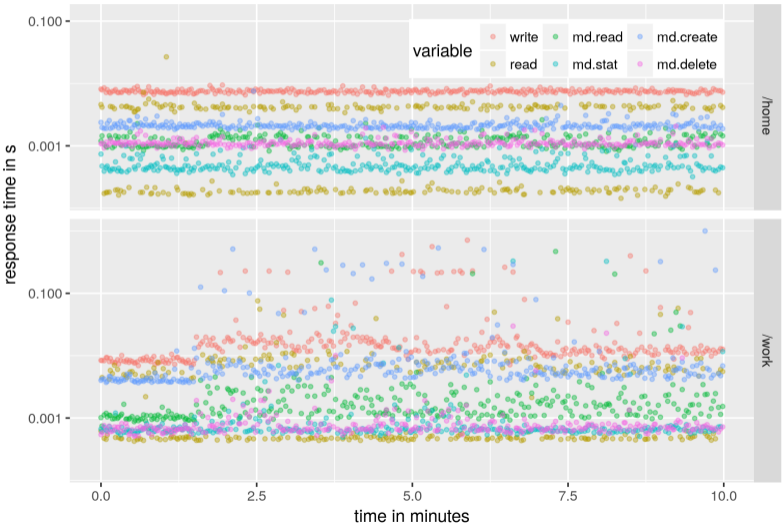
Executed as Bash script or an integrated tool: <https://github.com/joobog/io-probing>



# Test Systems

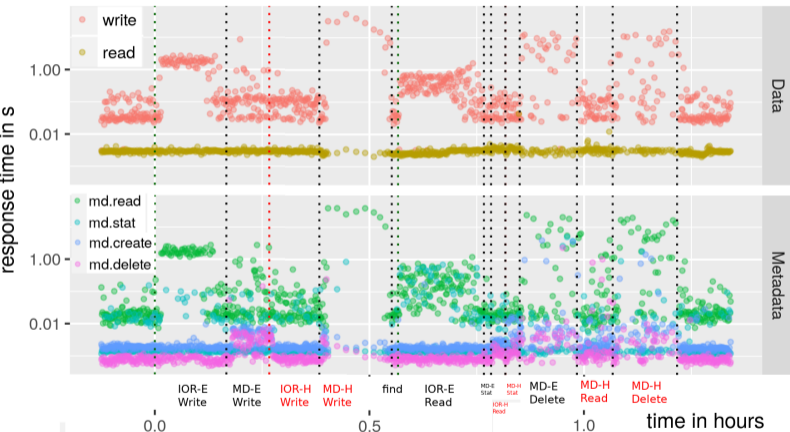
- JASMIN, the data analysis facility of the UK
  - ▶ Precreation: 200k files, 200 GB data file
  - ▶ 60 days of data
  - ▶ Script runs exclusively on a node
- Archer, the UK national supercomputer service
  - ▶ Precreation: 200k files, 200 GB data file
  - ▶ 30 days of data
  - ▶ Script runs on a shared interactive node
- Mistral, the HPC system at the German Climate Computing Center
  - ▶ Precreation: 100k files, 1.3 TB data file
  - ▶ 18 days of data
  - ▶ Tool runs on a shared interactive node

# Understanding the Timeseries



- Every probe (1s) for 10 min
- For two file systems
- Home is stable
- Work shows irregularities

# IO-500 Response Time on Archer



- Run on 100 nodes score 8.45
- The IO-500 various phases Data and metadata heavy
- First, all measurements

Figure: Response time (all measurements)

# Validating Slowdown on All Measurements

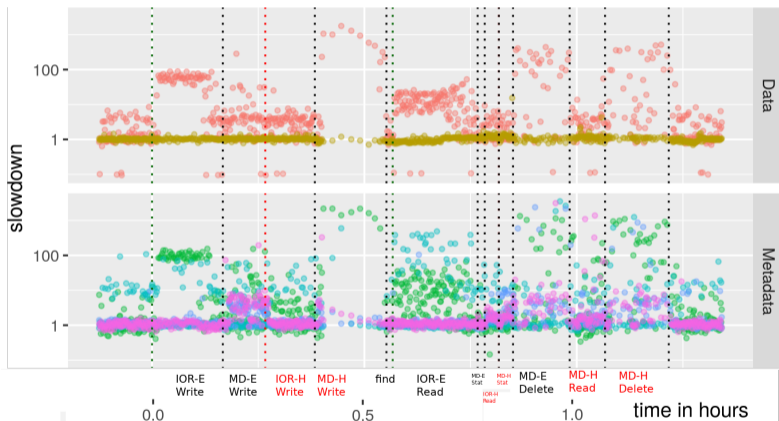
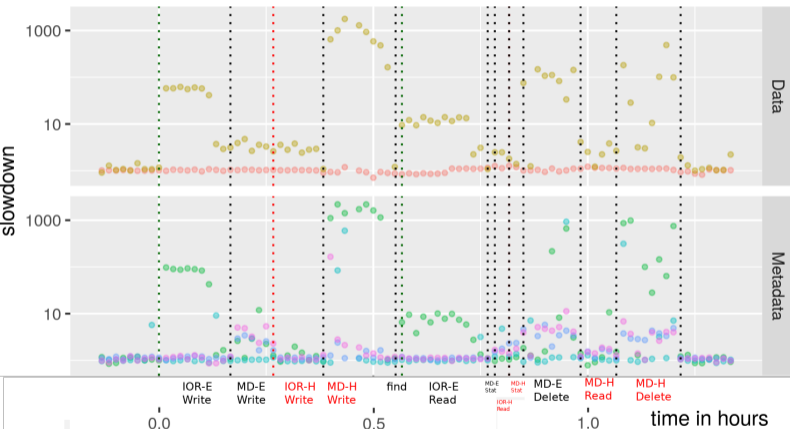


Figure: Slowdown (all measurements)

- Computed median slowdown  
Expected: median of 30 days
- Influence of phases is visible
- MDHard 1000x slowdown  
Influences data latency!  
10s of seconds latency
- IOREasy 100x slowdown
- IORHard not too much
- Data read is stable

# Validating Slowdown: Reduced Data



- Data reduction: 60s mean
- More robust, clearer to see

Figure: Slowdown (60s mean statistics)

# Timelines of 4h Statistics

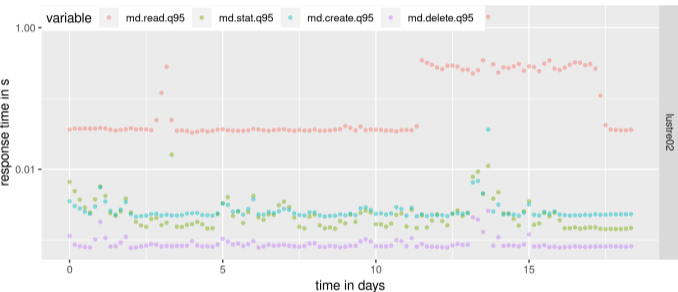


Figure: Mistral metadata timeline

- Use Q95, 5% ops are slower
- Change in behavior at day 12  
Reason: unknown

# Slowdown for 4h Statistics

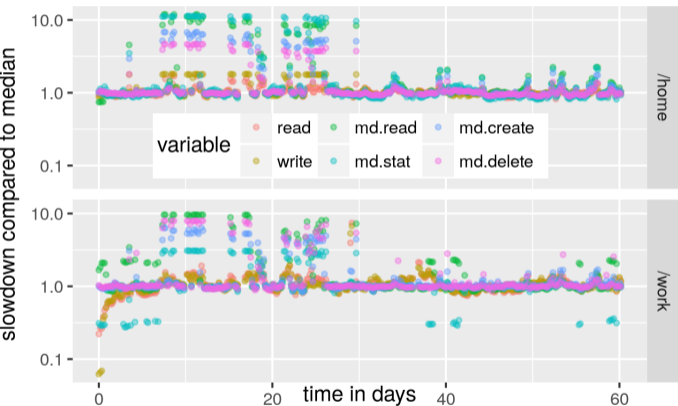


Figure: JASMIN, computed on 4 hour intervals

- Slowdown: Using the median
- Typically value is 1
- Sometimes a system is 10x slower
  - ▶ Due to user interactions
  - ▶ Concurrent application execution
- Values below 1, unusual (caching)
- Good to see long-term issues

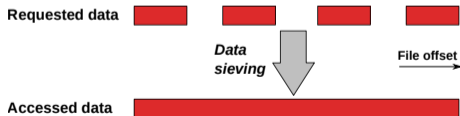
# Outline

- 1 Introduction
- 2 NetCDF
- 3 Monitoring I/O
- 4 Benchmarking
- 5 Optimizations**
- 6 Other
- 7 Outlook
- 8 Summary



# Optimizations

- There are too many tunables and optimizations for I/O
  - ▶ Read-ahead, write-behind, async I/O
  - ▶ Distribution of data across servers (e.g., Lustre stripe size)
  - ▶ We will investigate the complexity of one example...
- Performance benefit of I/O optimizations is non-trivial 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?

# Experiments

- 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% arithmetic mean performance
- Picking the best 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 Freq.	Worst Freq.	Arithmetic Mean			Harmonic Mean	
Servers	Stripe	Sieving			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
<b>2</b>	<b>128 K</b>	<b>Off</b>	87	76	<b>69.8%</b>	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
<b>2</b>	<b>128 K</b>	<b>On</b>	80	58	<b>68.7%</b>	239.6	62.6	<b>16.2%</b>	0.17
2	2 MiB	On	5	74	62.9%	258.0	107.3	14.9%	0.16

Table: 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

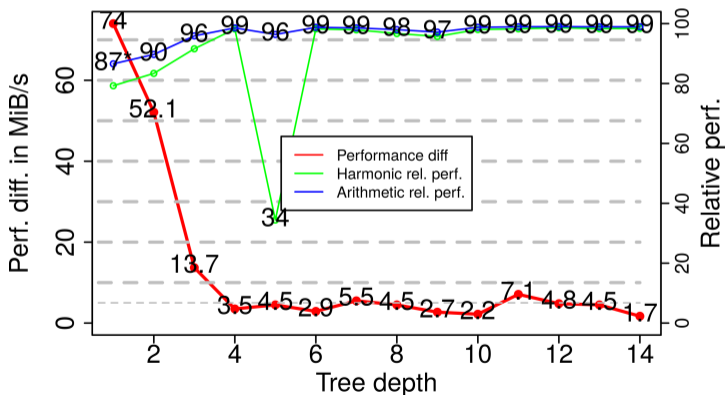


Figure: Perf. difference between learned and best choices, by maximum tree depth, for DKRZ's porting system

# 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

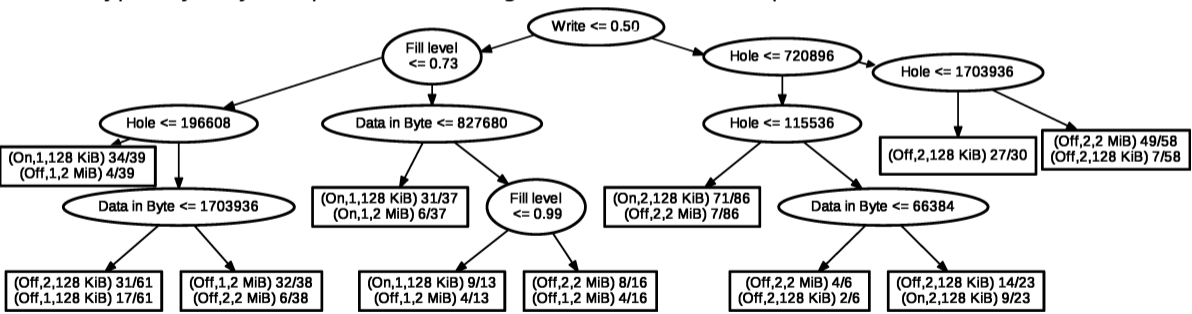


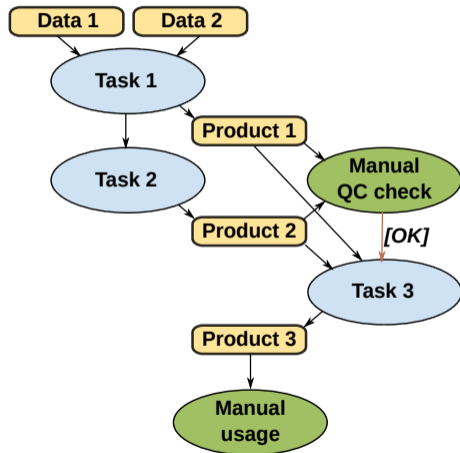
Figure: Decision tree with height 4. In the leaf nodes, the settings (Data sieving, server number, stripe size) and number of instances for the two most frequent best choices

# Outline

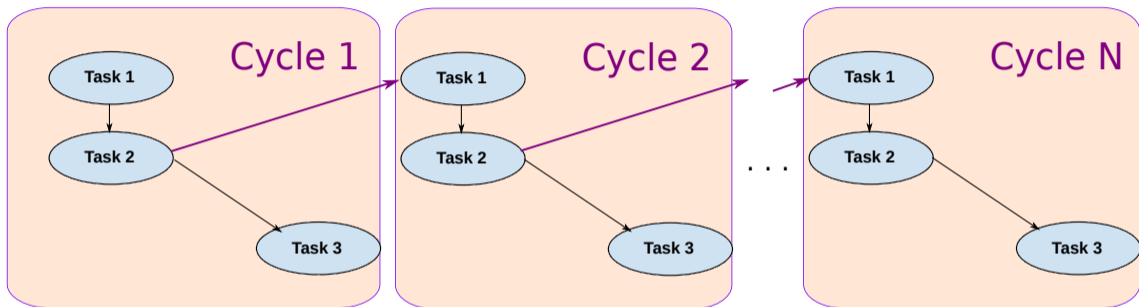
- 1 Introduction
- 2 NetCDF
- 3 Monitoring I/O
- 4 Benchmarking
- 5 Optimizations
- 6 Other**
- 7 Outlook
- 8 Summary

# Workflows

- Insight: What users are interested in
- Consider workflow from 0 to insight
  - ▶ Needs input
  - ▶ Produces output data
  - ▶ Uses tasks
    - Parallel applications
    - Big data tools
    - Manual analysis / quality control
  - ▶ May need month to complete
  - ▶ Manual tasks are unpredictable



# A (Science) Workflow Description



- Current practice (in climate/weather)
- Dependencies between tasks are described
- Assume a calculation that repeats for multiple cycles/iterations

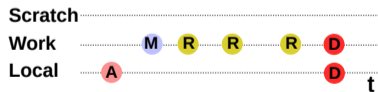


# Complexity of Data Placement Scheduling

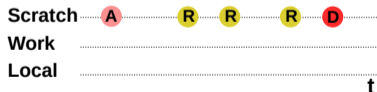
## Scenario

- Consider three file systems: local, scratch, and work
  - ▶ Local is a compute-node local storage system
- Data can be stored on any of these storage systems
- Users need to manually optimize data placement to hardware throughout life cycle
- Could the system do more knowing details about the workflow?

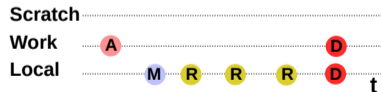
## Alternative life cycles for mapping a dataset (Selection)



Local and work file systems



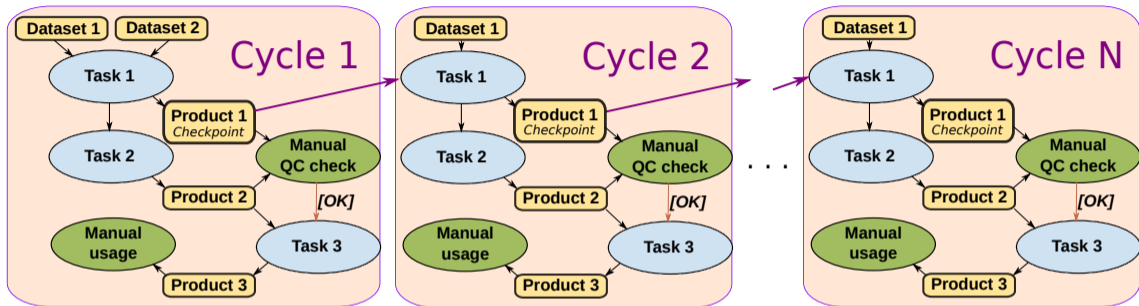
Scratch file system only



Local and work file systems

**A**llocation, **M**igration, **R**eading, and **D**eleting

# Possible Extended (Science) Workflow Description



## ■ Workflow description with IO characteristics

- ▶ Input required
  - ▶ Needed input
  - ▶ Generated output and its characteristics
  - ▶ Information Lifecycle (data life)
- ⇒ Explicit input/output definition (dependencies) instead of implicit

# Data-Reduction

## ■ Issues

- ▶ Storing data for a long time is expensive
- ▶ Performance is an issue

## ■ Data can be stored in various formats on storage media

## ■ Data-Reduction techniques aim to reduce storage requirements

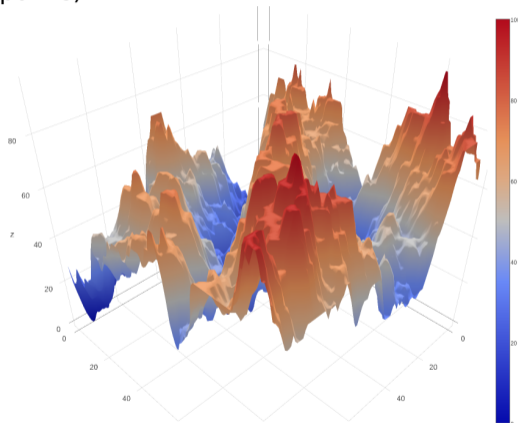
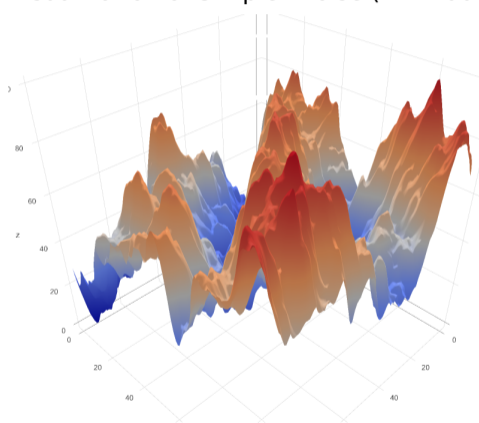
## ■ Strategies

- ▶ Avoiding output - challenge: need data for analysis!
- ▶ Re-computation - recreate data upon need using the same computing
- ▶ Lossless compression - compress data such that bit-identical data can be recreated
  - Examples: bzip, zip, WAV (audio)
- ▶ Lossy compression - (some, configurable) data loss upon recreation
  - Example: MP3, video files

## ■ Typically measured as compression ratio, e.g., 10:1 (means 10% capacity remains)

# Example Data

Visualization of Simplex noise (2D: 100x100 points)



Right picture compressed storing just 3 most significant bits (ratio 11.3:1)



# Outline

- 1 Introduction
- 2 NetCDF
- 3 Monitoring I/O
- 4 Benchmarking
- 5 Optimizations
- 6 Other
- 7 Outlook**
- 8 Summary

# Research Activities & Interest

## 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: Prescribing settings
- ▶ Management of (data-driven/big data) workflows

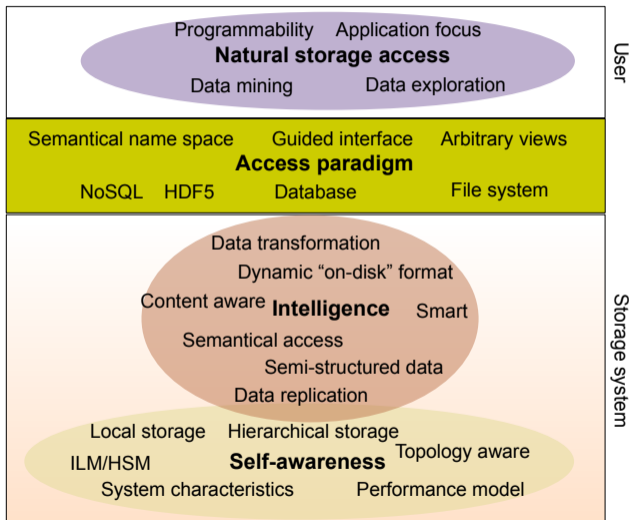
### ■ Data reduction: compression library, algorithms, methods

### ■ Interfaces: towards domain-specific solutions and novel interfaces

## Other research interests

- Application of big data analytics (e.g., for humanities, medicine)
- Cost-efficiency for data centers in general
- Scientific Software Engineering

# Personal Vision: Towards Intelligent Storage Systems and Interfaces



- Abstract data interfaces
- Enhanced data management
- Integrated compute/storage
- Flexible views on data
- Smart hardware/storage
  - ▶ Self-aware systems
  - ▶ AI optimized placement
  - ▶ Bring-your-own-behavior model
- Across sites and cloud



# Summary

- Achieving efficient I/O is challenging due to
  - ▶ complex systems
  - ▶ deep software stack
  - ▶ performance variability
  - ▶ optimizations
- Monitoring, performance analysis and benchmarking is needed
- There are many optimization strategies
- The NetCDF data model manages n-Dimensional data

# Bibliography

- 100 <http://www.zdnet.com/article/getting-flashy-apac-storage-market-shifts-as-cloud-demand-grows/>
- 101 <https://www.nics.utk.edu/sites/www.nics.tennessee.edu/files/pdf/Lonnie.pdf>
- 102 <https://www.unidata.ucar.edu/software/netcdf/workshops/most-recent/nc3model/NcClassicModel.html>