Department of Computer Science

The importance of AI for high-performance I/O

Limitless Storage
Limitless Possibilities

https://hps.vi4io.org

Julian M. Kunkel

BSSG Open source AI workshop

2019-04-05

Outline

University of Reading

High-Performance Computing (HPC)

Definitions

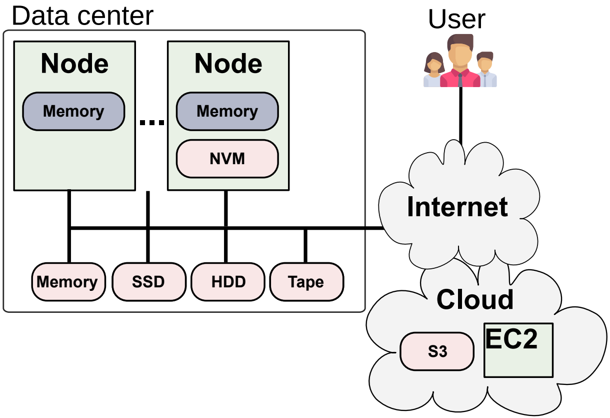
- HPC: Field providing massive compute resources for a computational task
  - ▶ Task needs too much memory or time for a normal computer
  - ⇒ Enabler of complex scientific simulations, e.g., weather, astronomy
- Supercomputer: aggregates power of many compute devices

Example: Summit (Oak Ridge National Laboratories)

- Compute: 4,608 nodes; 2.4 Million core
  - Peak 200 Petaflop/s (10^{15})
  - 2x IBM POWER9 22C 3.07GHz; 6x NVIDIA Volta V100 GPU
- 10 PB memory (DRAM + HBM + GPU)
- Network: 100G Infiniband
- Storage: 32 PB capacity; 1 TB/s throughput

Supercomputers & Data Centers

University of Reading



Credits: STFC

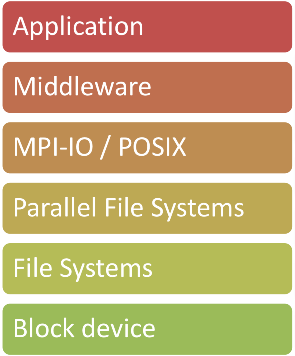
JASMIN Cluster at RAL / STFC

Used for data analysis of the Centre for Environmental Data Analysis (CEDA)

A View on The I/O Stack

University of Reading

- 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)



Example I/O stack

These layers provide plenty of optimization strategies and various tunables

Challenges

- The I/O hardware/software stack is very complex – even for experts
- Achieving high performance
- Understanding observed behavior (and performance)
- Tuning system settings and configurations
- Limited performance portability – manual tuning
- 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

Outline

Research Activities & Interest

High-performance storage for HPC

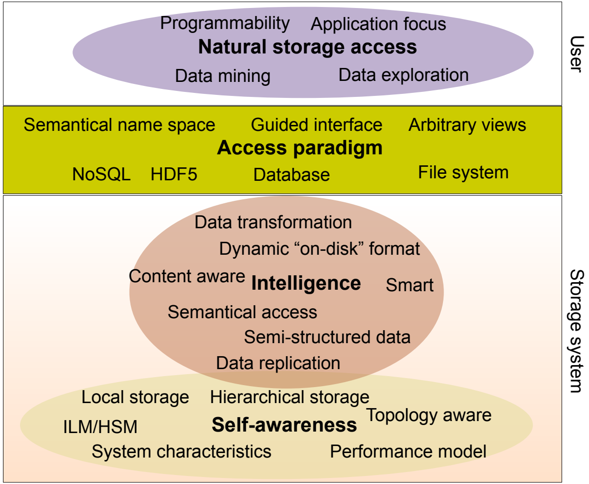
- Efficient I/O
  - Performance analysis methods, tools and benchmarks
  - Optimizing parallel file systems and middleware
  - Modeling of performance and costs
  - Tuning of I/O: Prescribing settings
  - Management of workflows
- Data reduction: compression library, algorithms, methods
- Interfaces: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



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

Outline

I/O Modeling and Diagnosing Causes with Statistics

Problem

Assessing observed time for I/O is difficult:

What is the cause for the slow/fast operation?

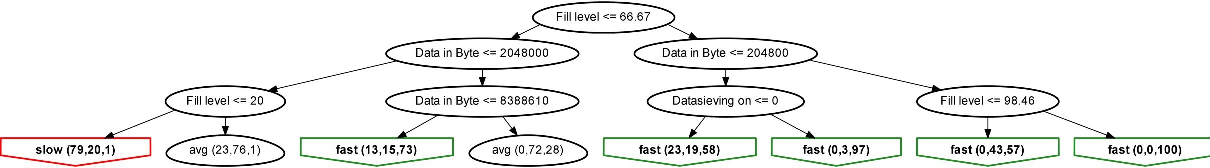
What best-case performance can we expect?

Goal

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

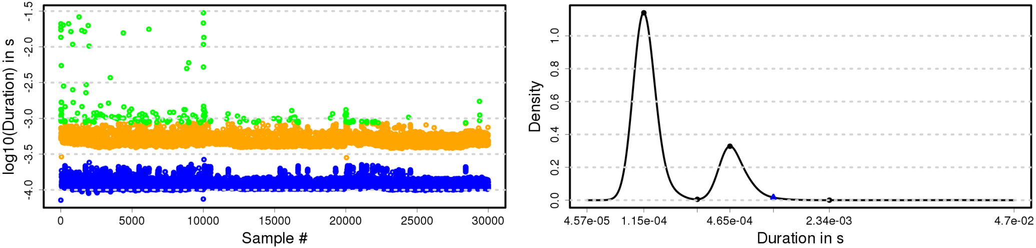
A Simple Performance Model for Extracting Knowledge

- Here: measured 408 different configurations applying two tunables
- Consider a performance prediction in three classes (fast, avg., slow)
- Rules extracted from decision trees (this is common sense for I/O experts)
  - Small fill levels and data sizes are slow
  - Large fill levels achieve good performance
- Surprising finding: 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.

Towards Predicting Cause of Effect

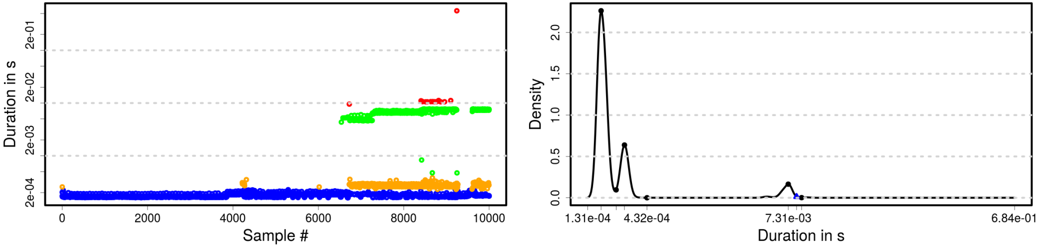


Duration for sequential reads with 256 KiB accesses (off0 mem layout)

Issues

- 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; they depend on internal state
- Consequence: Non-linear access performance

Write Operations



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

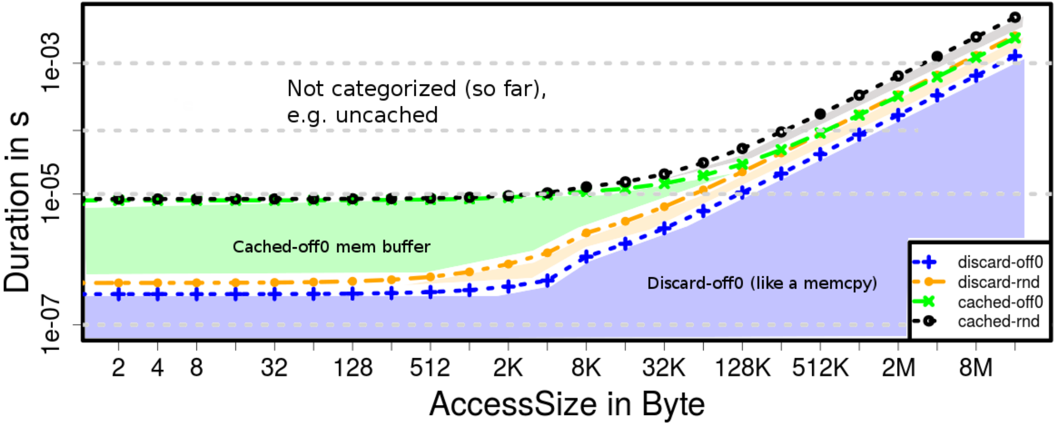
Known optimizations for write

- 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 but is opaque to users and applications

Towards Performance Models

Here are simple linear models for cached client side observations



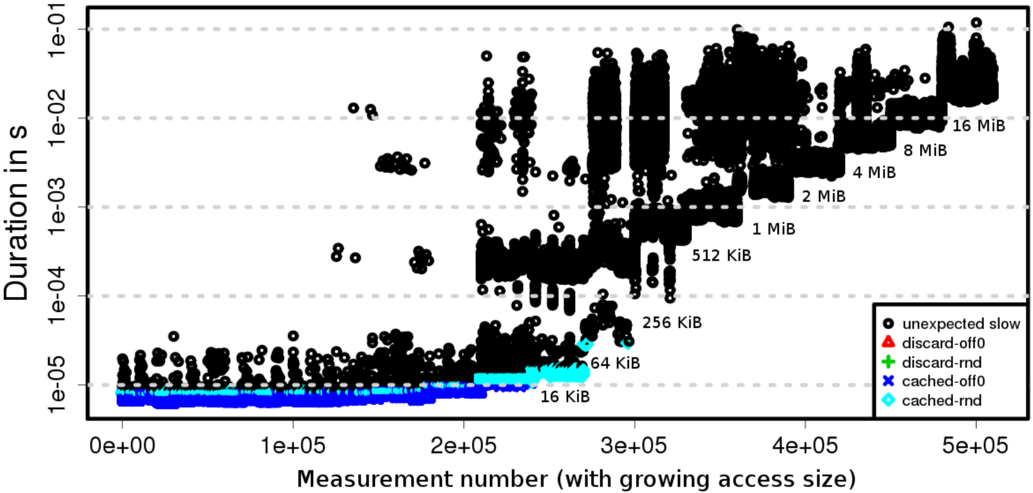
Read models predicting caching and memory location.

Couldn't machine learning or deep learning do better to identify the cause?

Using the Linear Model to Identify Anomalies

University of Reading

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

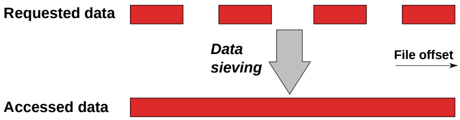


Outline

Prescriptive Analysis: Learning Best-Practises for DKRZ

University of Reading

- **Performance benefit of I/O optimizations is non-trival to predict**
- Example: Non-contiguous I/O supports data-sieving optimization
  - Transforms non-sequential I/O to large contiguous I/O
  - Tunable with 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?

Paper: *Predicting Performance of Non-contiguous I/O with Machine Learning. Kunkel, Julian; Zimmer, Michaela; Betke, Eugen. 2015, Lecture Notes in Computer Science*

System-Wide Defaults

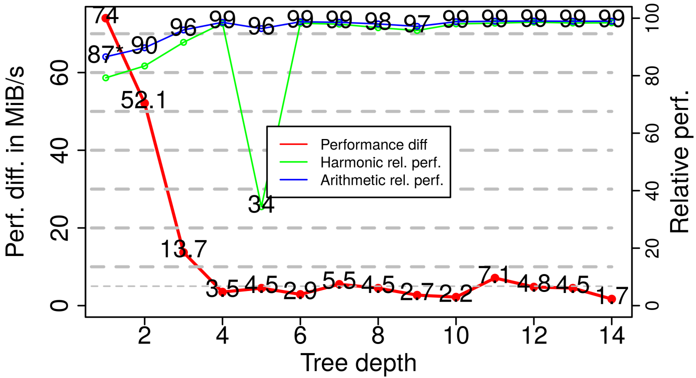
- Comparing a default choice with the best choice
- All default choices achieve 50-70% arithmethic mean performance
- Picking the best default for stripe count/size: 2 servers, 128 KiB
  - 70% arithmetic mean performance
  - 16% harmonic mean performance ⇒ some bad choices result in very slow performance

Default Choice			Best	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

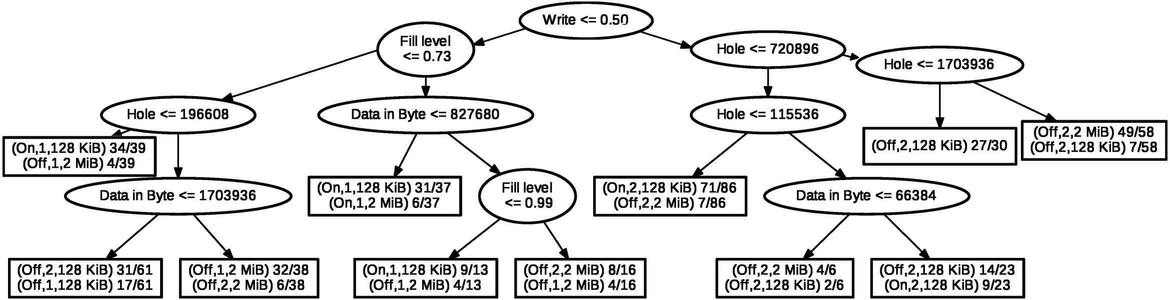


Perf. difference between learned and best choices, by maximum tree depth, for DKRZ's porting system

Extraction of Knowledge From a tree

University of Reading

- 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

This produced similar knowledge as known by experts from data center

Prescribing Compression Methods

- We are developing 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
- Still unresolved question:
  What metrics and algorithm to make best compression choice?

https://github.com/JulianKunkel/scil

Outline

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

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

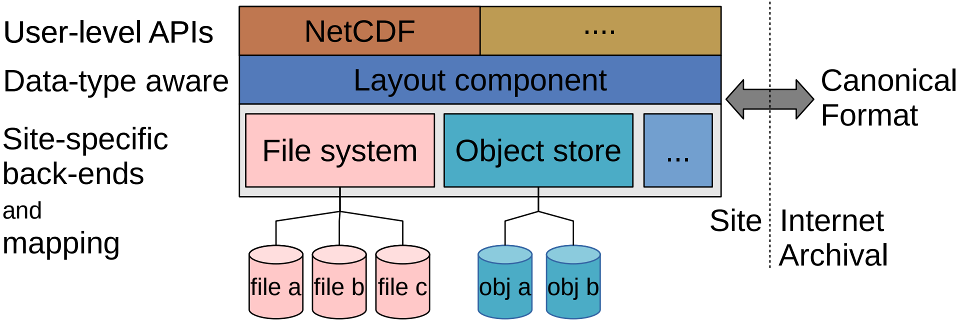
- Scalable data management practice
- The inhomogeneous storage stack
- Suboptimal performance & performance portability
- Data conversion/merging

Architecture

University of Reading

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



Earth-System Data Middleware

University of Reading

Design Goals of the Earth-System Data Middleware

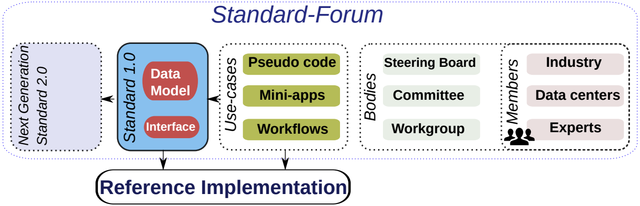
1. Relaxed access semantics, tailored to scientific data generation
2. **Site-specific (optimized) data layout schemes**
   - Based on site-configuration and performance models
   - Site-admin/project group defines mapping
   - Flexible mapping of data to multiple storage backends
   - Exploiting backends in the storage landscape
3. Enable a configurable namespace based on scientific metadata

We hope machine learning will make smarter choices for data layouting and system parameters

ESDM is just the Beginning: Next Generation Interfaces

University of Reading

Community Strategy via a Forum / Open Board



Towards a new storage/compute stack (for data-flow processing)

- Higher-level semantics
  - User metadata and workflows as first-class citizens
  - Liquid-Computing enabling smart-computing and storage
- **Smart hardware and software components with AI**
  - Self-aware instead of unconscious
  - Improving over time (self-learning, hardware upgrades)

Outline

Summary

University of Reading

- Parallel I/O is complex
  - System complexity and heterogeneity increases significantly
  - ⇒ Expected and measured performance is difficult to assess
  - Humans are unable to understand the behaviorial complexity of HPC systems
  - HPC users (scientists) and data centers need methods and tools
- I believe **AI and machine learning** are key to overcome complexity and aid us
  - 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

Visit!

- The Virtual Institute for I/O `https://vi4io.org`
- The IO-500 list `http://io-500.org` and `https://ngi.vi4io.org`

Appendix

Measured Data

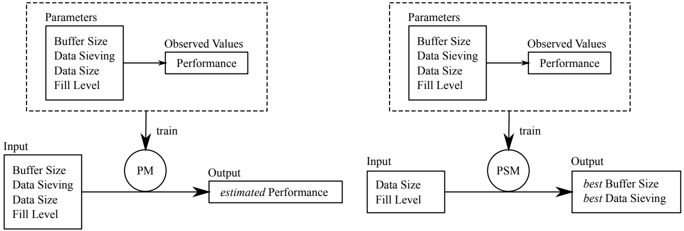
- Simple single threaded benchmark, vary access granularity and hole size
- Captured on DKRZ porting system for Mistral
- Vary Lustre stripe settings
  - 128 KiB or 2 MiB
  - 1 stripe or 2 stripes
- Vary data sieving
  - Off or On (4 MiB)
- Vary block and hole size (similar to before)
- 408 different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

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

University of Reading

- 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 arithmetic mean performance (54.7 MiB/s)
  - Achieves an arithmetic mean error of 28.5 MiB/s
- Linear models yield a mean error of $\geq$ 12.7 MiB/s

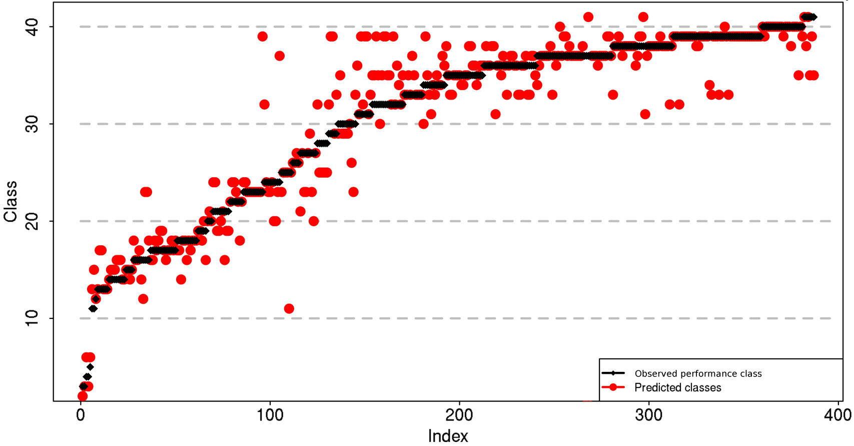
CART results

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

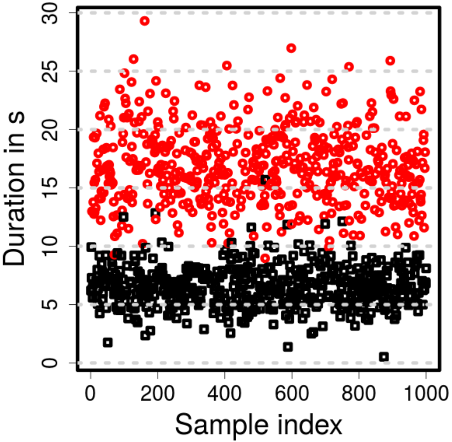


Performance classes and error for k=2, sorted by the observed performance class. Trained by 387 instances, validated on the other 387 instances.

Simulation of this Behavior

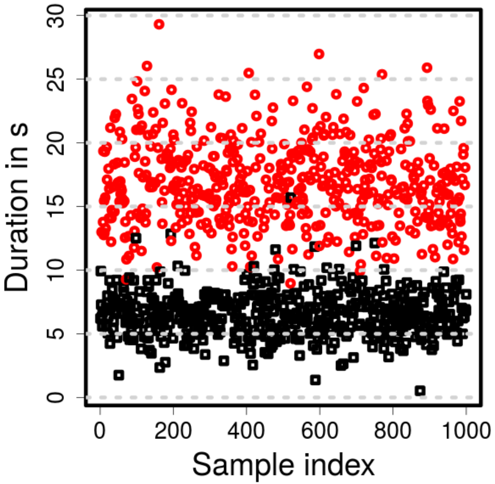
University of Reading

- 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)

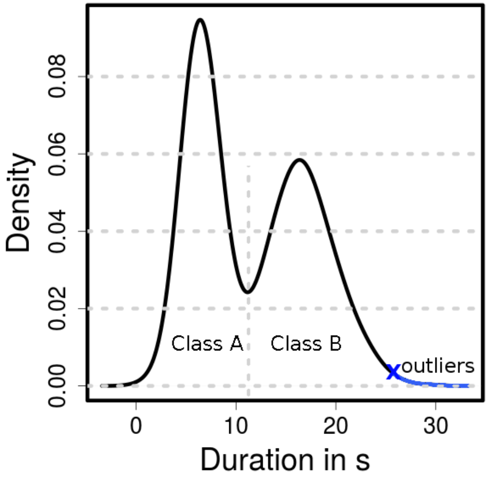


Resulting time for 1000 data points

Simulated Access Time and Resulting Density



(a) Timeline



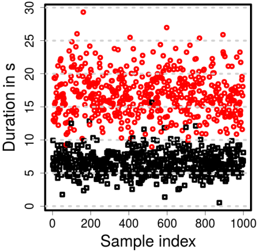
(b) Density reveals two classes

Approach

Assumptions

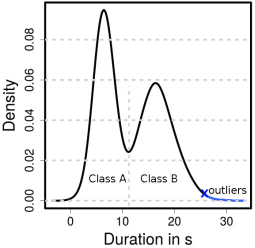
- Each "class" is caused another optimization/technology
  - Assign an observation to the likely class
  - This may lead to (tolerable) errors
- Behavior not visible on the density plot is irrelevant

$\Rightarrow$ 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

University of Reading

- 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 ...

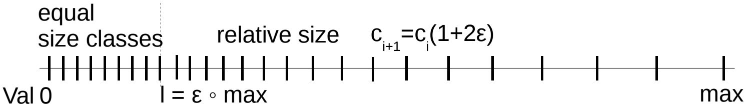
Transformation of the Problem

- 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

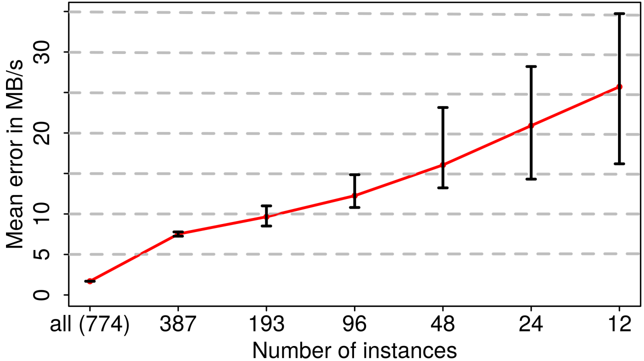
- Lustre 2.5
- 10 server nodes
- 1 Gb Ethernet
- 1 client node (max performance 110 MiB/s)

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)

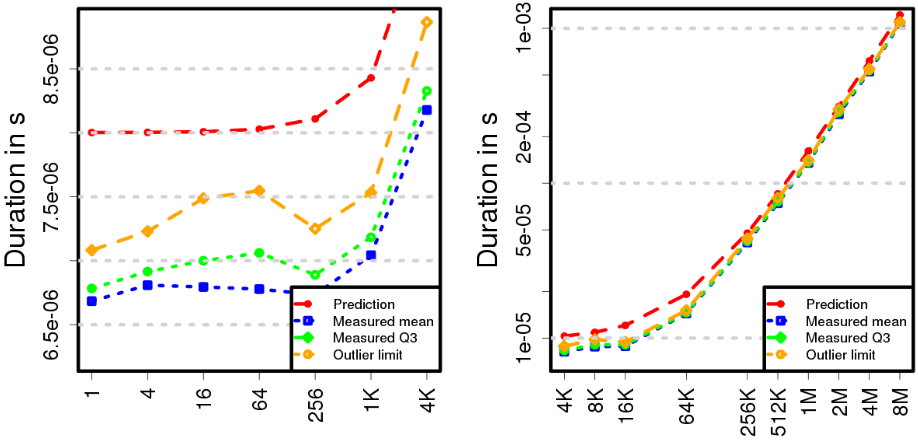
Investigating Training Set Size

- 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



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

Validation: Classify Different Patterns

Experiment state-mem-file	cached off0	cached rnd	discard off0	discard 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.