Data Reduction

Parallel Storage Systems

2023-07-03

Prof. Dr. Michael Kuhn
michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

- Why is it important to repeat measurements?
    1. Warm up the file system cache
    2. Randomize experiments for statistical purposes
    3. Eliminate systematic errors
    4. Eliminate random errors

- What does the queue depth for asynchronous operations refer to?
    1. Size of the operations
    2. Number of operations in flight
    3. Maximum size of an operation

- What is a context switch?
    1. The application switches between two open files
    2. The application switches between two I/O operations
    3. The operating system switches between two processes
    4. The operating system switches between two file systems

Performance Analysis

- Which is the fastest I/O setup in terms of potential throughput?
    1. One client communicating with one server
    2. One client communicating with ten servers
    3. Ten clients communicating with one server
    4. Ten clients communicating with ten servers

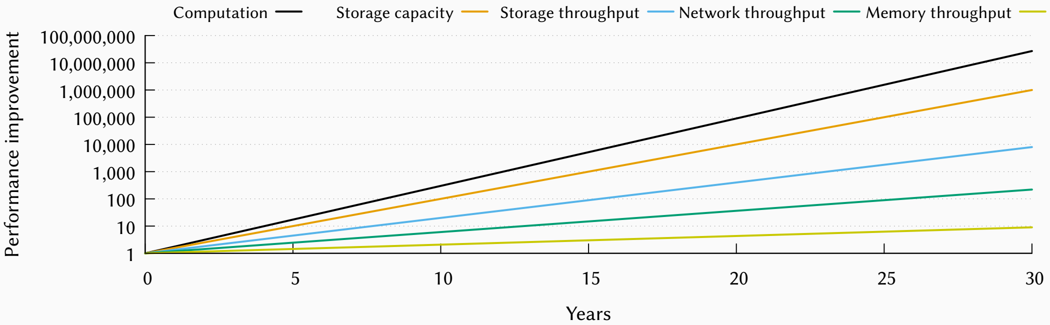
Outline

Data Reduction

- Review
- Motivation
- Recomputation
- Deduplication
- Compression
- Advanced Compression
- Summary

Performance Development

- Hardware improves exponentially, but at different rates
  - Storage capacity and throughput are lagging behind computation
  - Network and memory throughput are even further behind
- Transferring and storing data has become a very costly
  - Data can be produced even more rapidly
  - Often impossible to keep all of the data indefinitely
- Consequence: Higher investment costs for storage hardware
  - Leads to less money being available for computation
  - Alternatively, systems have to become more expensive overall
- Storage hardware can be a significant part of total cost of ownership (TCO)
  - Approximately 20 % of total costs at DKRZ, $\approx$ € 6,000,000 procurement costs



- Computation: 300× every ten years (based on TOP500)
- Storage capacity: 100× every 10 years
- Storage throughput: 20× every 10 years

Example: DKRZ [Kunkel et al., 2014]

	2009	2015	Factor
Performance	150 TF/s	3 PF/s	20x
Node Count	264	2,500	9.5x
Node Performance	0.6 TF/s	1.2 TF/s	2x
Main Memory	20 TB	170 TB	8.5x
Storage Capacity	5.6 PB	45 PB	8x
Storage Throughput	30 GB/s	400 GB/s	13.3x
HDD Count	7,200	8,500	1.2x
Archive Capacity	53 PB	335 PB	6.3x
Archive Throughput	9.6 GB/s	21 GB/s	2.2x
Energy Consumption	1.6 MW	1.4 MW	0.9x
Procurement Costs	30 M€	30 M€	1x

Example: DKRZ... [Kunkel et al., 2014]

	2020	2025	Exascale (2020)
Performance	60 PF/s	1.2 EF/s	1 EF/s
Node Count	12,500	31,250	100k–1M
Node Performance	4.8 TF/s	38.4 TF/s	1–15 TF/s
Main Memory	1.5 PB	12.8 PB	3.6–300 PB
Storage Capacity	270 PB	1.6 EB	0.15–18 EB
Storage Throughput	2.5 TB/s	15 TB/s	20–300 TB/s
HDD Count	10,000	12,000	100k–1M
Archive Capacity	1.3 EB	5.4 EB	7.2–600 EB
Archive Throughput	57 GB/s	128 GB/s	—
Energy Consumption	1.4 MW	1.4 MW	20–70 MW
Procurement Costs	30 M€	30 M€	200 M$

Data Reduction Techniques

- There are different concepts to reduce the amount data to store
  - We will take a closer look at three in particular

1. Recomputing results instead of storing them
   - Not all results are stored explicitly but recomputed on demand
2. Deduplication to reduce redundancies
   - Identical blocks of data are only stored once
3. Compression
   - Data can be compressed within the application, the middleware or the file system

Outline

Overview [Kunkel et al., 2014]

- Do not store all produced data
  - Data will be analyzed in-situ, that is, at runtime
- Requires a careful definition of the analyses
  - Post-mortem data analysis is impossible
  - A new analysis requires repeated computation
- Recomputation can be attractive
  - If the costs for keeping data are substantially higher than recomputation costs
- Cost of computation is often still higher than the cost for archiving the data
  - Computational power continues to improve faster than storage technology

Quiz

- How would you archive your application to be executed in five years?
    1. Keep the binaries and rerun it
    2. Keep the source code and recompile it
    3. Put it into a container/virtual machine

Problems: Binaries [Kunkel et al., 2014]

- Keep binaries of applications and all their dependencies
  - Containers and virtual machines have made this much easier
- Effectively impossible to execute the application on differing future architectures
  - x86-64 vs. POWER, big endian vs. little endian
  - Emulation usually has significant performance impacts
- Recomputation on the same supercomputer appears feasible
  - Keep all dependencies (versioned modules) and link statically

Problems: Source Code [Kunkel et al., 2014]

- All components can be compiled even on different hardware architectures
  - Most likely will require additional effort from developers
  - Different operating systems, compilers etc. could be incompatible
  - Might still require preserving all dependencies
- Changes to minute details could lead to differing results
  - Different processors, network technologies etc. could change results
  - Can be ignored in some cases as long as results are “statistically equal”

Summary

- Recomputation can be worth it given current performance developments
  - Computation is developing much faster than storage
- Reproducibility is relevant in general, not only for saving space
  - It should be possible to reproduce results independently
- Requires a careful definition of all experiments
  - Experiment cannot be adapted after the fact
- All input data has to be kept around for later executions

Outline

Overview [Kunkel et al., 2014]

- Data is split up into blocks (4–16 KB)
  - Different chunking methods can be used (static or content-defined)
- Each unique block of data is stored only once
  - A reference to the original block is created for each repeated occurrence
- Previous study for HPC data showed 20–30 % savings
  - Total amount of more than 1 PB
  - Full-file deduplication 5–10 %
- Deduplication also has its drawbacks
  - Deduplication tables have to be kept in main memory
    - Per 1 TB of data, approximately 5–20 GB for deduplication tables

Overhead [Kunkel et al., 2014]

- Deduplication tables store references between the hashes and data blocks
  - SHA256 hash function (256 bits = 32 bytes)
  - 8 KB file system blocks (using 8 byte offsets)
  - Additional data structure overhead of 8 bytes per hash
- Have to be kept in main memory for efficient online deduplication
  - Potential duplicates have to be looked up for each write operation
  - Fast storage devices such as SSDs are still orders of magnitude slower
    - NVRAM might be suitable in the future

$$1\,\mathrm{TB} \div 8\,\mathrm{KB} = 125,000,000$$

$$125,000,000 \cdot (32\,\mathrm{B} + 8\,\mathrm{B} + 8\,\mathrm{B}) = 6\,\mathrm{GB} \quad (0,6\,\%)$$

Analysis [Kunkel et al., 2014]

	2009	2015	2020	2025
Storage	5.6+**1.68** PB	45+**13.5** PB	270+**81** PB	1.6+**0.48** EB
RAM	20+**33.6** TB	170+**270** TB	1.5+**1.62** PB	12.8+**9.6** PB
Power	1.6+**0.24** MW	1.4+**0.20** MW	1.4+**0.14** MW	1.4+**0.09** MW
Costs	30+**2.52** M€	30+**2.38** M€	30+**1.62** M€	30+**1.13** M€

- Assumption: Optimistic savings of 30 %
- Deduplication is not suitable in an HPC context
  - Requires more additional RAM than available for computation (except for 2025)
  - Requires significantly more power (5–15 %)
  - Increases overall costs (3–8 %)

Analysis… [Kunkel et al., 2014]

2009	2015	2020	2025
4.3+**1.3** PB	34.6+**10.4** PB	207.7+**62.3** PB	1.2+**0.4** EB
20+**25.8** TB 1.54+**0.19** MW 28.27+**1.94** M€	170+**207.7** TB 1.34+**0.15** MW 28.27+**1.83** M€	1.5+**1.2** PB 1.34+**0.1** MW 28.27+**1.25** M€	12.8+**7.4** PB 1.34+**0.07** MW 28.27+**0.87** M€

- Assumption: Use deduplication to achieve same capacity
- Overhead is now more balanced
  - Still requires significantly more main memory
  - Power consumption is increased by up to 8 %
  - Overall costs drop starting 2020

Summary

- Larger blocks reduce overhead caused by deduplication tables
  - 8 KB → 0.6 %, 16 KB → 0.3 %, 32 KB → 0.15 %
  - Larger blocks also have a negative impact on deduplication rate
- Full-file deduplication can be an alternative
  - Storage throughput is not affected negatively
  - Files have to be written completely before hash can be computed
- Offline deduplication reduces runtime overhead
  - Relatively easy to implement using modern copy-on-write file systems
  - Especially useful for full-file deduplication
  - Influence on performance is not as dramatic
    - Tables do not have to be kept in main memory all the time

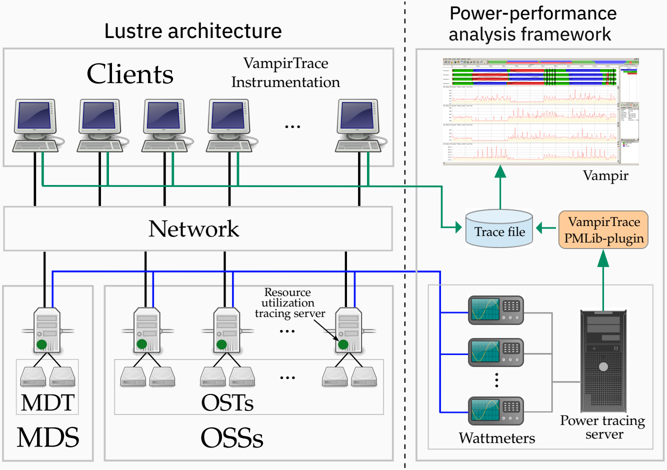
Outline

Overview [Chasapis et al., 2014]

- Goal: Capturing most important performance metrics of compression algorithms
    - Compression ratio, processor utilization, power consumption and runtime
- $\approx$ 500 GB of climate data (MPI-OM)
    - Preliminary tests with repeating and random data
    - Serial tests to determine base performance
    - Parallel tests for real-world applications

Tracing [Chasapis et al., 2014]

- Instrumented installation
  - VampirTrace for applications
  - pmserver for file system servers
  - Server to record power consumption
- Allows correlating client and server activities



- Which algorithm would you use?
  1. `none`
  2. `zle`
  3. `lzjb`
  4. `lz4`
  5. `gzip-1`
  6. `gzip-9`

Algorithm	Ratio	Utilization	Runtime
`none`	1.00	23.7	1.00
`zle`	1.13	23.8	1.04
`lzjb`	1.57	24.8	1.09
`lz4`	1.52	22.8	1.09
`gzip-1`	2.04	56.6	1.06
`gzip-9`	2.08	83.1	13.66

[Chasapis et al., 2014]

Algorithms [Chasapis et al., 2014]

- Compress climata data set
- Runtime is increased moderately
  - Except for higher gzip levels
- gzip increases utilization significantly
- lz4 (and gzip-1) are most interesting

Algorithm	Ratio	Utilization	Runtime
none	1.00	23.7	1.00
zle	1.13	23.8	1.04
lzjb	1.57	24.8	1.09
lz4	1.52	22.8	1.09
gzip-1	2.04	56.6	1.06
gzip-9	2.08	83.1	13.66

Algorithms… [Chasapis et al., 2014]

- Repeating data
  - Generated using yes
- `lz4` has low utilization
  - Even lower than no compression
- Both algorithms increase runtime

Algorithm	Ratio	Utilization	Runtime
none	1.00	23.7	1.00
`lz4`	126.96	15.8	1.28
`gzip-1`	126.96	23.3	1.24

Algorithms… [Chasapis et al., 2014]

- Random data
  - Generated using `frandom` module
- `gzip-1` increases utilization
  - Almost 3× of the others
- Almost no effect on runtime
  - Reminder: Serial test on one HDD

Algorithm	Ratio	Utilization	Runtime
`none`	1.00	23.5	1.00
`lz4`	1.00	24.1	0.97
`gzip-1`	1.00	66.1	1.03

Parallel Application [Chasapis et al., 2014]

- Modified IOR benchmark
  - More realistic write activity
- Application performance unaffected
  - Higher I/O throughput on servers
- Energy consumption lower for `lz4`
  - Lower runtime with almost no increase in power consumption
- `gzip-1` increases energy by only 1 %

Algorithm	Runtime	Power	Energy
none	1.00	1.00	1.00
lz4	0.92	1.01	0.93
gzip-1	0.92	1.10	1.01

Analysis [Kunkel et al., 2014]

	2009	2015	2020	2025
Storage	5.6+**2.8** PB	45+**22.5** PB	270+**135** PB	1.6+**0.8** EB
Power	1.6+**0.025** MW	1.4+**0.025** MW	1.4+**0.025** MW	1.4+**0.025** MW

- Assumption: Compression ratio of 1.5 for `lz4`
  - 10 % increase in power consumption (pessimistic)
- Runtime ratio of 1.0, that is, no change
  - Does not require additional processors for compression

Summary

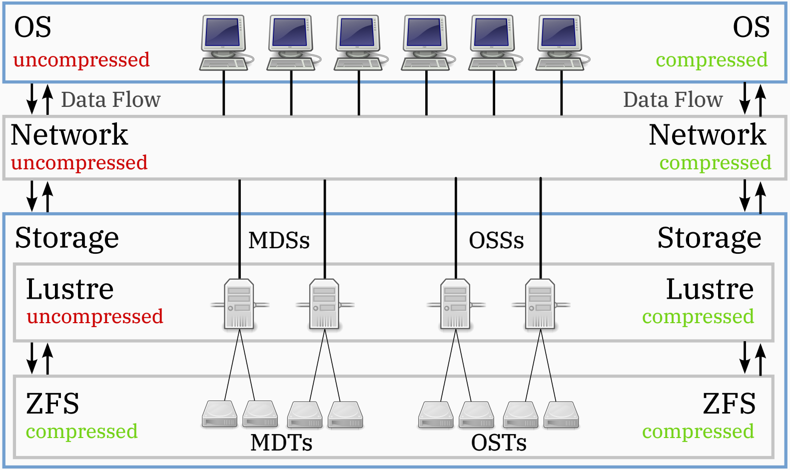
- Compression can increase storage capacity significantly
  - Suitable algorithms have negligible overhead
  - Often not necessary to buy additional hardware
- Low increase in power consumption
  - Overall, still worth it due to capacity increase
- Application-specific compression can increase ratios significantly
  - Applications can leverage lossy compression
  - Compression ratios of ≥ 10 are possible

Outline

Overview

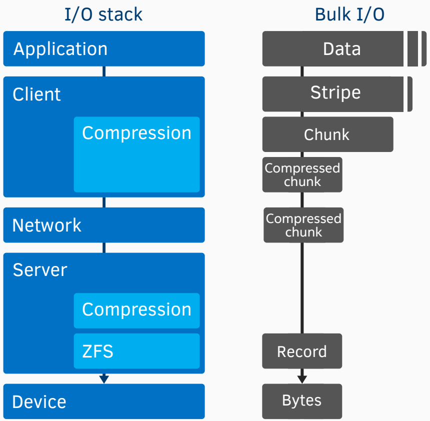
- Compression is already available in some file systems
  - ZFS and btrfs support transparent compression
  - Lustre can make use of ZFS as a local backend
- File systems currently use static approaches for compression
  - Typically one compression algorithm/setting per file system
  - Dynamic approaches can compress data more efficiently
- Application knowledge can improve compression results
  - Dynamic approaches also have to guess algorithms and settings
  - Compression hints can be used to influence decisions

- Left: Current status
- Right: Work in progress
  - Compress across full data path
  - Improve network throughput
  - Avoid redundant compression



Overview...

- Transparent compression in Lustre
    - No application changes necessary
- Additional benefits
    - Effective network throughput is increased
    - Recompression for archival is possible
- Significant cost savings are possible
    - Shrinking to 50 % is often feasible

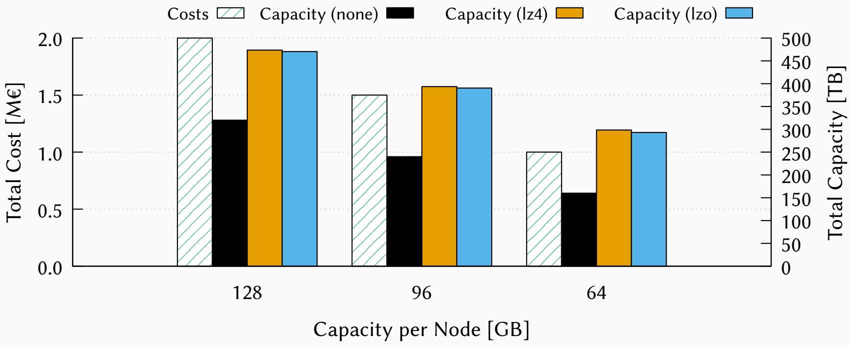


Overview… [Kuhn et al., 2016]

- lz4 and lz4fast are good overall
  - zstd is also interesting
  - All three can be tuned using parameters
- Multiple candidates for archival

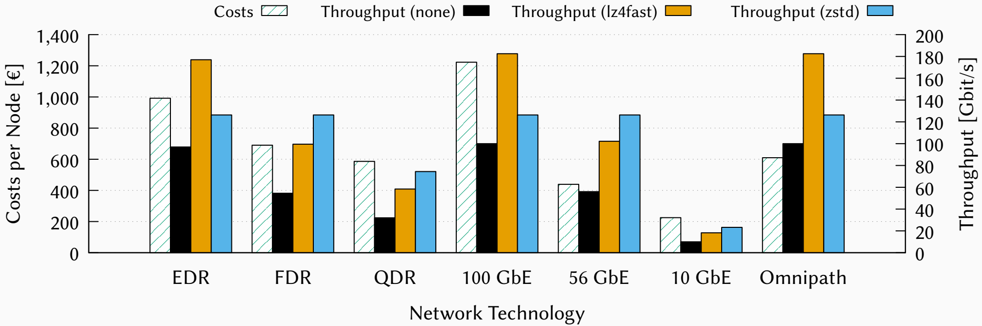
Alg.	Comp.	Decomp.	Ratio
lz4fast	2,945 MB/s	6,460 MB/s	1.825
lz4	1,796 MB/s	5,178 MB/s	1.923
lz4hc	258 MB/s	4,333 MB/s	2.000
lzo	380 MB/s	1,938 MB/s	1.887
xz	26 MB/s	97 MB/s	2.632
zlib	95 MB/s	610 MB/s	2.326
zstd	658 MB/s	2,019 MB/s	2.326

Main Memory [Kuhn et al., 2016]



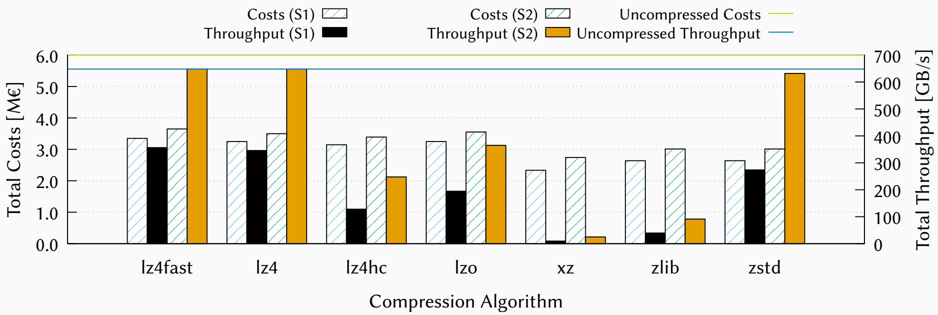
- Compress main memory transparently (using zram)
- Goal: Capacity of 128 GB per node
  - Not possible with 64 GB of main memory (compress 60 GB, leave 4 GB uncompressed)
  - lzo can slow down memory throughput tremendously (< 10 GB/s)

Network [Kuhn et al., 2016]

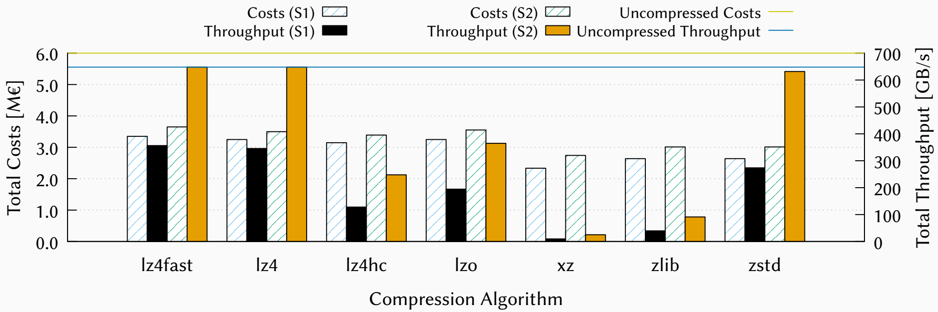


- zstd reduces throughput for networks with very high throughput (> 54 Gbit/s)
- FDR can be replaced with QDR when using lz4fast (cost reduction of 15 %)
  - lz4fast and zstd can increase throughput to 100 Gbit/s and 125 Gbit/s with FDR

Storage [Kuhn et al., 2016]



- S1: As many servers as necessary for 50 PB (lower costs/throughput)
- S2: 50 servers and as many HDDs as necessary for 50 PB (higher costs/throughput)



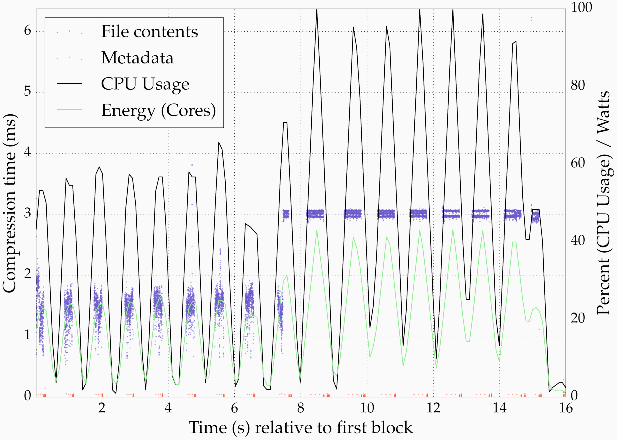
- lz4 and lz4fast do not impact performance negatively
  - Costs are reduced to € 3,500,000 (instead of € 6,000,000)
- zstd decreases throughput by 20 GB/s
  - Costs are reduced by 50 % to € 3,000,000

Adaptive Compression [Ehmke, 2015]

- Reminder: Compression is typically static
  - ZFS allows setting an algorithm per file system
- Adaptive compression supports multiple modes
  - Performance, archival, energy consumption etc.
- Uses different heuristics to determine compression algorithm
  - Heuristics are based on file type and cost functions
- All algorithms are tried for the cost function
  - Best algorithm is used for the following operations

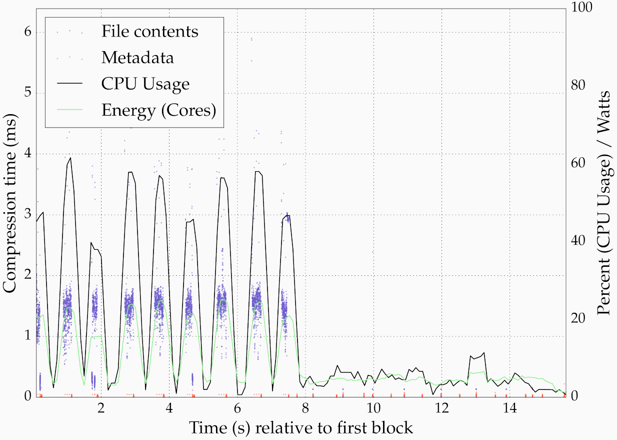
Adaptive Compression… [Ehmke, 2015]

- Compressing a mixed file
  - First part is compressible, second part is random
  - ZFS's `gzip-1` setting
- Random data increases utilization and power consumption

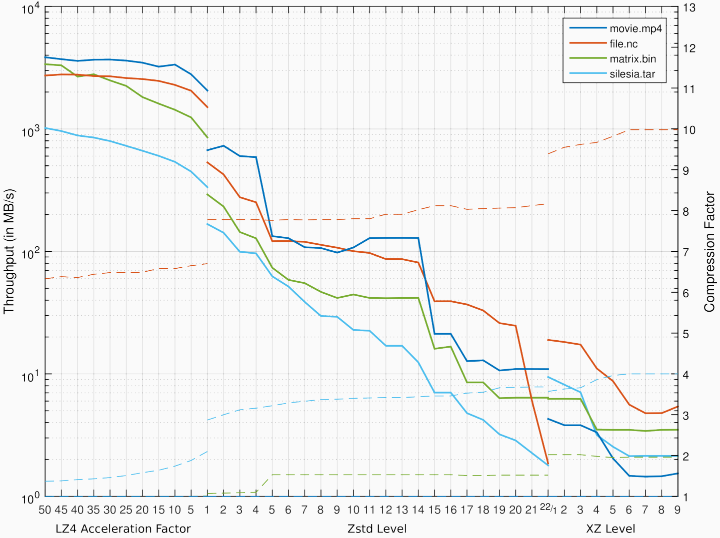


Adaptive Compression… [Ehmke, 2015]

- Compressing a mixed file
  - First part is compressible, second part is random
  - Adaptive archival mode
- Random data is effectively skipped

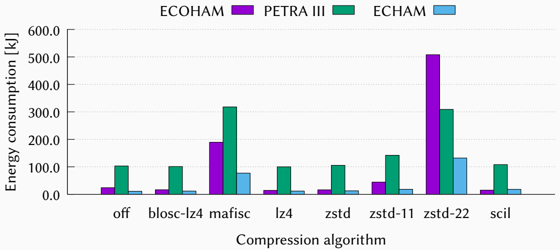


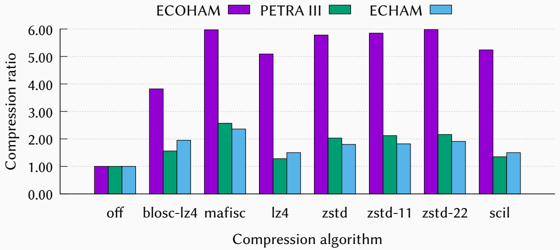
- Algorithms support levels
  - lz4 is very fast
  - zstd in middle range
  - xz suited for archival
- Combine algorithms
  - Allows adapting compression throughput



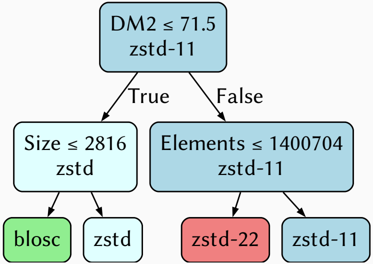
Adaptive Compression... [Kuhn et al., 2020]

- Selecting algorithms is complex
  - Performance depends on data
  - Currently a manual process
- Similar compression ratios with different energy consumption
  - See mafisc and zstd for ECOHAM
- Goal: Intelligent automatic selection
  - Less overhead for the developer
  - Avoid performance degradation





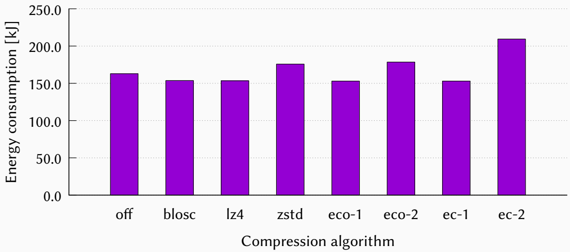
Adaptive Compression… [Kuhn et al., 2020]

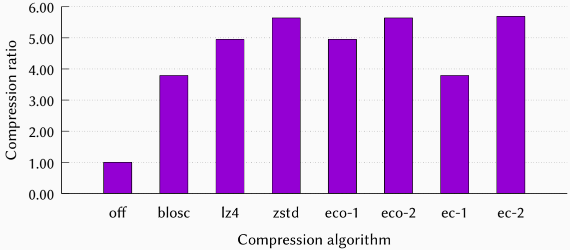


- Analysis of relevant properties
  - Matrix dimensions
  - Size of dimensions
  - Number of elements
  - Size of the data
  - Information about data type
- Decision component is trained with collected data
  - Collecting compression ratio, processor utilization, energy consumption etc.
- Decision component chooses algorithm and settings at runtime
  - Developer does not have to deal with compression anymore

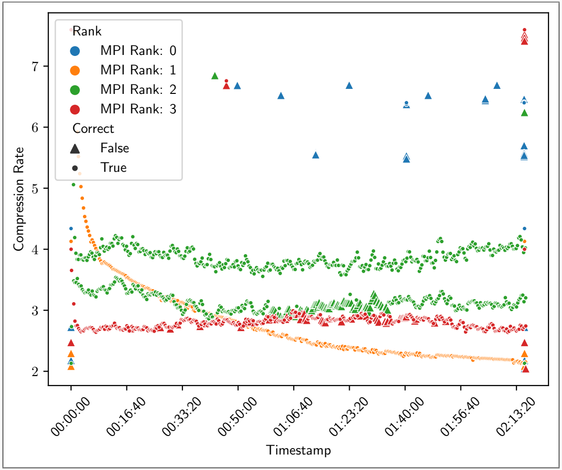
Adaptive Compression… [Kuhn et al., 2020]

- Tested using application ECOHAM
- Two use cases
    1. Training with known application (eco-1 and eco-2)
    2. Training with unknown application (ec-1 and ec-2)
- Automatic selection
    - Optimal result for known application
    - Slightly increased energy consumption/lower compression ratio for unknown application





- Neural network trained on data
    - Up until a certain timestep
    - Good results also for short durations
- Inferencing at application runtime
    - Can be integrated into an HDF5 filter
- Best choices vary within application
    - Ranks behave differently
    - Changes over time
- 14.5 GB reduced to 10.0 GB
    - Ideal compression only 0.14 % better



Outline

Summary

- Data reduction techniques can be very useful even in HPC contexts
  - Recomputation, deduplication and compression have different strengths
  - Performance and cost impact have to be analyzed carefully
  - Cost models and measurements can be combined to get a clear picture
- Compression can be leveraged relatively easily
  - Several algorithms offer high performance with little overhead
  - Data reduction should be performed in the most useful layer
- Computation and storage will likely continue developing at different rates
  - Storage capacity and throughput limitations will only get worse

References

[Chasapis et al., 2014] Chasapis, K., Dolz, M., Kuhn, M., and Ludwig, T. (2014). **Evaluating Power-Performace Benefits of Data Compression in HPC Storage Servers.** In Fries, S. and Dini, P., editors, *IARIA Conference*, pages 29–34. IARIA XPS Press.

[Ehmke, 2015] Ehmke, F. (2015). **Adaptive Compression for the Zettabyte File System.** Master's thesis, Universität Hamburg.

[Hirsch, 2017] Hirsch, J. (2017). **Dynamic decision-making for efficient compression in parallel distributed file systems.** Master's thesis, Universität Hamburg.

[Kuhn et al., 2016] Kuhn, M., Kunkel, J. M., and Ludwig, T. (2016). **Data Compression for Climate Data.** *Supercomput. Front. Innov.*, 3(1):75–94.

[Kuhn et al., 2020] Kuhn, M., Plehn, J., Alforov, Y., and Ludwig, T. (2020). **Improving Energy Efficiency of Scientific Data Compression with Decision Trees.** In *ENERGY 2020: The Tenth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 17–23. IARIA XPS Press.

[Kunkel et al., 2014] Kunkel, J. M., Kuhn, M., and Ludwig, T. (2014). **Exascale Storage Systems - An Analytical Study of Expenses.** *Supercomput. Front. Innov.*, 1(1):116–134.

[Plehn et al., 2022] Plehn, J., Fuchs, A., Kuhn, M., Lüttgau, J., and Ludwig, T. (2022). **Data-aware Compression for HPC Using Machine Learning.** In Kuhn, M., Duwe, K., Acquaviva, J., Chasapis, K., and Boukhobza, J., editors, *CHEOPS@EuroSys 2022: Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems, Rennes, France, 5 April 2022*, pages 8–15. ACM.