Data-Aware Compression for HPC using Machine Learning

Julius Plehn
Universität Hamburg
Hamburg, Germany
julius.plehn@uni-hamburg.de

Anna Fuchs
Universität Hamburg
Hamburg, Germany
anna.fuchs@informatik.uni-hamburg.de

Michael Kuhn
Otto von Guericke University Magdeburg
Magdeburg, Germany
michael.kuhn@ovgu.de

Jakob Lüttgau
University of Tennessee Knoxville
Knoxville, USA
jluettga@utk.edu

Thomas Ludwig
Deutsches Klimarechenzentrum GmbH
Hamburg, Germany
ludwig@dkrz.de

Abstract

While compression can provide significant storage and cost savings, its use within HPC applications is often only of secondary concern. This is in part due to the inflexibility of existing approaches where a single compression algorithm has to be used throughout the whole application but also because insights into the behaviour of the algorithms within the context of individual applications are missing.

There are several different compression algorithms available, with each one also having a unique set of options. These options have a direct influence on the achieved performance and compression results. Furthermore, the algorithms and options to use for a given dataset are highly dependent on the characteristics of said dataset.

This paper explores how machine learning can help with identifying fitting compression algorithms with corresponding options based on actual data structure encountered during I/O. In order to do so, a data collection and training pipeline is introduced. Inferencing is performed during regular application runs and shows promising results. Moreover, it provides valuable insights into the benefits of using certain compression algorithms and options for specific data. Further investigations into more advanced machine learning techniques and a deeper integration into existing I/O paths will provide additional benefits.

CHEOPS '22, April 5, 2022, RENNES, France
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9209-9/22/04.
https://doi.org/10.1145/3503646.3524294

CCS Concepts: **• Information systems → Storage architectures**; **• Theory of computation → Data compression**; **• Computing methodologies → Supervised learning by classification**.

Keywords: compression, machine learning, file systems, HDF5

ACM Reference Format:
Julius Plehn, Anna Fuchs, Michael Kuhn, Jakob Lüttgau, and Thomas Ludwig. 2022. Data-Aware Compression for HPC using Machine Learning. In *Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '22), April 5, 2022, RENNES, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3503646.3524294

1 Introduction

Due to the significantly different increases in CPU and storage performance, using compression by default has become tenable. However, most file systems and I/O libraries either do not support compression at all or do not enable it by default, because the compression algorithm's performance and overhead depends on the data being compressed. In addition to that, most file systems and libraries that support compression, only allow choosing one specific algorithm that is used for all data.

Previous studies using decision trees have shown that machine learning approaches can help automatize the algorithm selection effectively [8]. However, these studies have been limited to using metadata contained in self-describing data formats to make such decisions. While this already allows automatizing the decision making process and takes the burden of selecting an appropriate algorithm from the user, it is still limited. We are therefore presenting an extended approach that also takes the actual data into account. Specifically, the contributions of this paper are:

1. We intercept of calls of commonly used I/O APIs (HDF5, NetCDF etc.) and specific applications (e.g., ICON) to analyze the data being written.

2. We use machine learning to train models that identify the optimal compression algorithm to use for new data according to several metrics (e.g., CR, compression speed, decompression speed etc.).

2 Background

In this section, we introduce background on the I/O stack, common practices in HPC storage systems and their built-in compression features and justify the need for adaptive compression.

Storage Stack. Before I/O operations issued by HPC applications become persistent, they typically traverse multiple layers, like illustrated on the left part of Figure 1. In particular, scientific applications often make use of self-describing data formats and libraries such as NetCDF and HDF5. While they used to be competing standards, HDF5 is used internally by NetCDF-4. Both applications as well as the libraries implementing the self-describing data formats internally use distributed coordination and communication standards such as the Message Passing Interface (MPI) for handling parallel I/O before it is passed on to OS-level APIs.

Compression. Compression can be categorized into lossy and lossless algorithms. Lossy compression requires the data to be approximable and is very application-specific, while lossless compression, which we focus on in this paper, can be applied automatically and transparently to the user and application. The two basic metrics when applying compression are speed (or throughput) and ratio (CR), which we define as $CR = \frac{original}{compressed}$. There are a couple of popular compression algorithms, which vary in speed (compressing and decompressing) and ratio and therefore can have different areas of application. Several studies [8, 12, 13] have shown that LZMA, ZLIB, ZSTD, XZ and BZIP2 belong to the group with the highest CRs on different sample data; LZ4 and snappy seem to be the fastest. We have chosen three tunable and well-supported algorithms for our study: ZLIB and ZSTD representing higher CRs and LZ4 with high compression (HC) mode expecting it to be faster with acceptable CR. ZLIB uses the DEFLATE algorithm, which is a combination of an LZ77-derivate and Huffman coding and offers 9 levels (the higher, the better it will compress). Its window is limited to 32 KiB for historical reasons and, therefore, its memory usage is low (about 0.03 % for larger data sets) [4]. ZSTD is another LZ77-based algorithm, additionally using fast Finite State Entropy and Huffman coding. While aiming for real time compression, it offers 7+22 levels. The negative seven levels accelerate the compressor at the cost of the ratio; the positive ones behave the other way around. Its window is not limited by design and the memory consumption may rise significantly especially for levels above 20 [15]. LZ4 is written by one of the authors of ZSTD and is known as one of the fastest compressors in the last years; it is also based on LZ77. Its fast mode has unlimited levels with about 3 % of speed increase per level and data-dependent CR reduction [1]. The HC mode offers 9 levels and reaches ratios comparable to the other compressors we have chosen. Both modes have a window size of 64 KiB and use the same extremely fast decompressor, which clearly outperforms all the others.

While the I/O stack used in HPC allows for various points to compress at, using no compression is still predominant. The first option to choose would be the different hardware components: client machines, which perform the computation and are equipped with powerful CPUs or GPUs and large amounts of main memory; and storage servers, which are commonly much weaker, but have no additional tasks to perform. Hardware-assisted compression, becomes more popular and is able to gain higher speeds with same compression algorithms, but requires special chips or cards and may have stricter requirements on the memory layout [12]. Within the client, compression can be applied in software in the application, middleware or third-party libraries, and the file system client. Application-specific compression is hardly ever implemented due to the huge metadata management efforts and troubles to match the underlying system's data management to be efficient. Using additional features or libraries is the common way for compressing data before it is sent to the storage server. NetCDF allows compression for collective I/O only for newest versions (netcdf-c ≥ 4.7.4 and hdf5 ≥ 1.10.3 [9]). HDF5 supports compression for parallel, collective writes and chunked datasets [5]. The libraries support, among others, ZLIB, LZ4 and ZSTD, while the usage of additional filters and external plugins offers other compression methods. Chunking the data is required for using compression here and can lead to many writes to multiple storage targets for each rank, which can result in higher I/O latency [14]. Therefore, client-side file system compression might be the best and easiest way to utilize client hardware for compression, but this feature does not exist yet.

File Systems. The only two file systems ever featured by Top100 supercomputing systems are Lustre and IBM's Spectrum Scale [11]. IBM's flagship uses a few algorithms for hot and cold data for server-side compression [10].

Lustre does not support any compression except for server-side when using the ZFS backend, which is used less than ldiskfs without compression support[3]. In both systems, data is transferred in full size over the network.

Adaptive Compression. However, even if compression is used at all, the algorithm is used mostly statically and independent of the actual occurring data. Due to the missing universal perfect algorithm, one needs adaptive mechanisms to avoid introducing too much overhead. The two basic compression metrics mentioned above (speed and CR), can be measured very easily, but do not provide enough insights to choose the optimal algorithm, since the application's, user's or system's needs are often more complex. Therefore, derived or combined metrics allow for better mapping of suitable algorithm characteristics to the requirements. These can be very abstract like best energy efficiency or overall performance, or very specific, like best ratio at a specific minimum throughput or fastest algorithm at a minimum ratio. More metrics (memory, CPU usage etc.), their combinations and the distinction between write and read, where the decompression speed is significant, are possible.

In this paper, we use the CR per time as a derived metric for proof of concept purposes. In our studies we have seen clear trends for ZSTD when recognizing only CR or the clear winner LZ4 when looking only at decompression speed (independently of initial CR). However, the factor how much slower ZSTD would be for the continuously higher CR depends on the data. Chunk sizes affect the CR because of available entropy and read performance, since the whole chunk must be decompressed for small reads. Smaller chunks improve small random I/O, but decrease the CR and may hurt large I/O. With very high decompression speeds, the usage of larger chunks offer higher CR potential and at the end, depending on the data, lead to better overall performance than best-CR algorithms on smaller chunks. All those reasons do not allow us to predict the optimal algorithm statically.

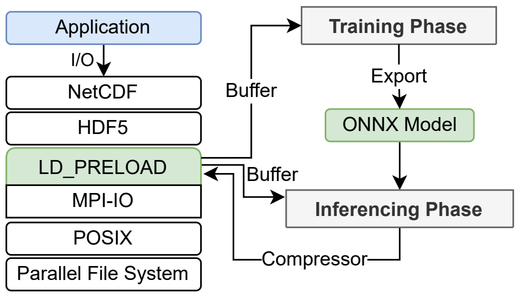
Use Case. Adaptive compression as a method should generally consider all relevant metrics, which include system metrics like CPU, memory and network load. In our project about client-side compression in Lustre [2], we can take care of them as much as possible and determine the limits for speeds and ratios to, e.g., saturate the network or avoid application interrupts, when sharing the CPU between compression and computation. Looking inside the data blocks during runtime and analyzing them for best suitable algorithm would slow down the file system. However, neglecting to take the data content into account will lead to inaccurate results. One would experience the same problems for any runtime environment with concurring resources, so that pre-analyzing the data content is the most efficient way.

Our approach aims for interfaces which can set the compression algorithms for separate chunks or blocks, which is currently no possible in middleware solutions, but will be possible within the file system.

3 Architecture

In the following we introduce our architecture which helps to automate three of the core challenges associated with implementing adaptive data-dependent choice of compression algorithms:

1. *Sampling Phase*: Allowing for automated sampling of input-target pairs for training without requiring changes to applications.
2. *Training Phase*: Leveraging state of the art machine learning frameworks such as PyTorch for training and hyper-parameter search.



(a) HPC I/O Stack

(b) Sampling, training, and inferencing architecture for optimization of compressor selection.

Figure 1. Architecture overview showing a typical I/O path and points of instrumentation in relation to the training phase as well the inferencing phase. The training phase exports a machine learning model which is then used in an inferencing phase to predict a compressor based on data that is written by an application. A more detailed breakdown of the training phase is illustrated in Figure 2.

3. *Inferencing Phase*: Integration of in-band machine learning based decision components efficiently without the overhead of python runtime components by using the portable ONNX format and runtimes.

All related materials and source code is available online[1].

Sampling Phase. We leverage the fact that most application I/O is performed through common interfaces such as MPI-IO. By intercepting MPI-IO calls, we are able to cover applications using MPI-IO directly but also those using HDF5 or NetCDF's parallel I/O.

In Figure 1 a top view onto the architecture is given. In order to intercept I/O easily from various applications the MPI-IO layer is well-suited. As can be seen, our approach is based on preloading a custom library (using the `LD_PRELOAD` environment variable) and intercepting the MPI-IO calls. For each I/O operation, we log certain metadata (size, datatype etc.) and compress the data buffer using a wide range of compression algorithms and settings. All metadata and resulting metrics, such as compression speed and CR, are then stored in an HDF5 file for later analysis. This information is used as part of our training phase.

Training Phase. For training, the data itself is converted into an image representation and fed to a neural network together with the measured metrics. We use PyTorch to train the model. The complete training phase is shown in more detail in Figure 2.

Inferencing Phase. Once the training phase is complete, we export the model using ONNX, which allows the model to be used from our preloadable C library. The inferencing

[1]https://github.com/wr-hamburg/eurosys2022-cheops-compression

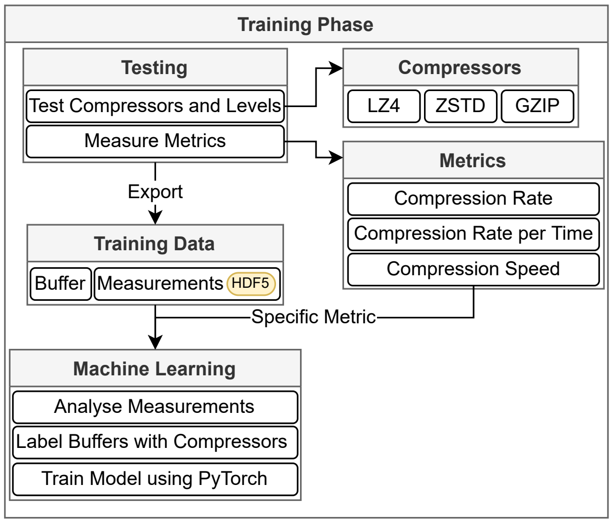


Figure 2. Training phase which shows the involved steps of gathering training data for specific metrics and using those measurements to train a model using PyTorch.

phase takes place during regular execution of an application and determines the most appropriate compression algorithm and settings to use. Our library can tune the predictions based on currently three metrics: the compression rate, the compression rate per time and the compression speed. On the one hand, optimizing only for the compression rate will always choose the best-compressing and thus most likely the most expensive compressor. On the other hand, using only the compression speed will most likely select compressors with a low compression rate. Including both the compression rate and time will make sure that reasonable compression rates are achieved while not slowing down I/O excessively.

4 Evaluation

The architecture introduced above provides two valuable data sources, which are discussed and evaluated in this section. A fundamental aspect of the architecture is the ability to collect compression statistics according to the introduced metrics. This provides meaningful insights into the variance of fitting compression algorithms and associated compression levels. While those aspects are of importance in order to train a neural network and therefore implement the ability to predict compression algorithms for unseen data, the behavior of said network and the performance during inferencing is evaluated, as well.

All measurements were done using the ICON modelling framework [2], which is a project between the German Weather Service and the Max Planck Institute for Meteorology. ICON is used for weather prediction as well as climate modelling. The application itself is written in Fortran. However NetCDF is used for I/O, which itself relies on HDF5, which then uses MPI-IO for actual reading and writing the data. As shown in Figure 1, the shared library is therefore capable of intercepting I/O related calls without any changes to the application itself. Another aspect, which makes ICON ideal for evaluation, is that the amount of data that is written is configurable by choosing appropriate options for the simulation environment. This is done by either selecting certain variables of interest or by limiting or extending the time frame which should be simulated. For evaluation purposes, metrics have been measured during a four month simulation period. The machine learning model is then evaluated on additional eight months, which results in an observed period of twelve months.

[2]https://code.mpimet.mpg.de/projects/iconpublic

4.1 Metrics

For each chunk of I/O that has been intercepted, all compressors and specific levels were tested. Therefore, when searching for the best fit for a certain metric, one specific compressor per chunk of I/O is selected for analysis. These perfect fits also serve as a label for the associated chunk when training a machine learning model further on.

This is shown in Figure 3, where we test thirteen compressor combinations in total. For the four month period used within ICON for data collection purposes, 7,548 MPI-IO calls were traced.

Compression Ratio. For visualization purposes, the range of shown compression ratios in Figure 3a is limited to below two, which lowers the number of shown measurements to 5,156. Those excluded measurements achieved exceptional ratios because of uniformity of the included data. In about 45 % of all cases, ZSTD-22 is the superior compressor, followed by ZLIB-6. A compressor with a lower compression level is sometimes selected, because depending on the data a more aggressive compression level might not result in a higher CR. In those cases there is not reason to prefer a higher level, as this might result in a longer runtime.

Compression Ratio per Time. When using this metric, the overwhelming majority of chunks should be compressed using ZSTD-1. Therefore, when using ICON and this metric is desired, the influence of other compressors is negligible. ZSTD-1 provides the optimal balance between a high compression rate and speed.

Compression Speed. This metric shows similarities to the previous one in that for most cases ZSTD-1 is ideal. It is also noteworthy that, for certain chunks, LZ4 achieves significant decompression speeds which are unmatched by any other compressor.

Decompression Speed. In over 45 % of the cases, LZ4-1 is capable of achieving the greatest decompression speeds. However, a variety of different levels, e.g., level 3, 9 and 12 are useful in certain situations, as well. While low compression

```
(flatten): Flatten(start_dim=1, end_dim=-1)
(linear_relu_stack): Sequential(
  (0): Linear(in_features=4096,
              out_features=512, bias=True)
  (1): ReLU()
  (2): Linear(in_features=512,
              out_features=512, bias=True)
  (3): ReLU()
  (4): Linear(in_features=512,
              out_features=num_classes, bias=True)
)
```

Listing 1. Architecture of the neural network used in our evalation. Despite the simplicity of the network and significant bias in our training data, it proves effective when considering decision quality in respect to realized compression ratio and compression and decompression speeds. Smaller models are desireable as they limit memory and runtime overhead.

levels are predominant and beneficial for high decompression speeds in most cases, there are also situations where higher levels like LZ4-12 are helpful. This might be due to efficiencies that can be exploited when the data is packed more densely into the memory and less data has to be processed in general.

Altogether it can be said that in most cases not a single compression algorithm serves as an ideal choice for each metric. As shown earlier the influence of the data itself should not be underestimated.

4.2 Machine Learning and Inferencing

The metrics shown previously are now used to train an application-specific neural network. For now the network is intentionally built using rather basic layers in order to explore the limitation of the current approach. This is shown below. The data is passed through a sequential stack which consists of several linear transformations. Those are connected to ReLU activation functions. Finally, the number of outputs is dynamically set to the number of possible classes for this specific metric.

Compression Ratio. Figure 4 shows the losses when training the compression rate metric. This model shows no sign of overfitting and the observable losses are leveling out. The final model accuracy is at 79.3 %. The confusion matrix shown in Figure 5 resembles the metrics seen in Figure 3a. In most cases, ZSTD-22 is predicted which matches the true compressor. However, in several cases ZLIB-9, ZSTD-10 or LZ4-12 should be used and ZSTD-22 is wrongfully predicted. In other cases, ZLIB-6 can be confidently predicted, followed by ZSTD-3. This is also due to the imbalance of usable data when collecting the metrics, which has to be improved further.

In Figure 6 the selection of CRs clearly shows that, for this specific application, a relationship between the used MPI rank and the compression rate exists. Over time, the ratios change differently which is especially noticeable when comparing rank 1 with ranks 2 and 3. This observation is also relevant when discussing how I/O changes during the run and how well a trained model might generalize over time and when using different application configurations.

During this evaluation run, ICON writes 14.5 GiB of data. Using the predicted compressors, this is reduced to 10.0 GiB. It is noteworthy that when in all occasions the ideal compressor would have been used, the total size would only reduce by an additional 0.14 %. This shows that when the CR is of concern, the most important decision is to use a compressor at all. In case of ICON, a sufficient choice might be ZSTD-22.

Decompression Speed. In Figure 7, the losses encountered during the individual epochs can be seen. Interestingly, the loss when using decompression speed as a target metric performs worse than CR (compare Figure 4). As seen previously in Figure 3d and now in the confusion matrix in Figure 8, ZLIB is not recommended when decompression speed is of interest. For the most part, LZ4-1 is rightfully predicted, followed by LZ4-12. Sometimes, the model predicts LZ4-12 instead of LZ4-9,because LZ4-9 is underrepresented in comparison with LZ4-12. The lack of correct predictions of ZSTD variants is less significant but is based on the same fundamental issue.

5 Related Work

This work is at the intersection of HPC storage systems, data reduction techniques and the application of machine learning techniques to automated decision making for performance optimization.

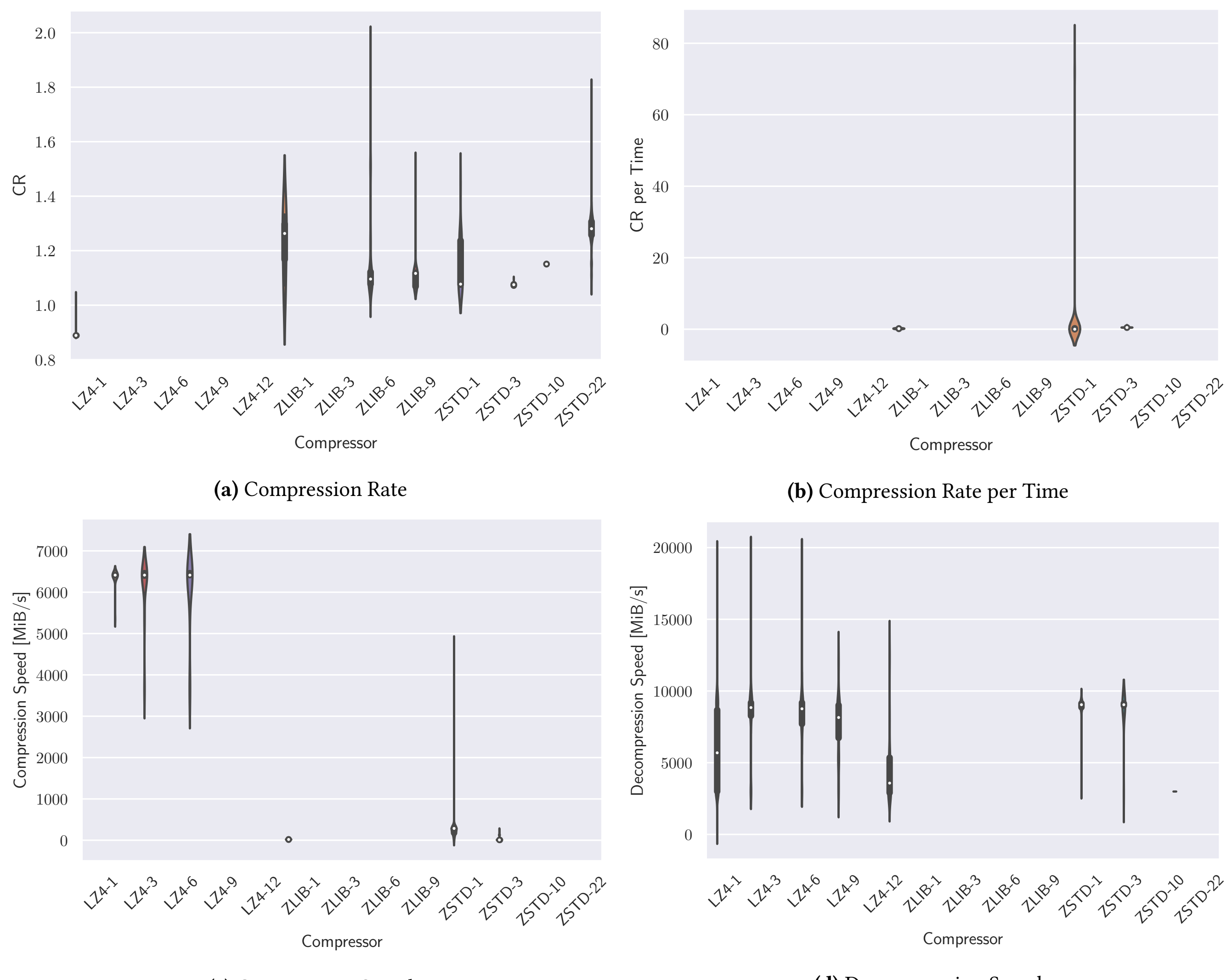
A recent survey analyzed opportunities for machine learning supported decision making [6]. The survey dedicates a sizable section to the discussion of approaches that produce performance estimates but no work considering their application to compression are discussed.

A machine learning approach that also considers a sampling phase to gather training data is evaluated by [17] but with a focus on lossy compression.

The authors of [8] have applied ML methods for adaptive compression based on metadata. They could boost the CRs for known and unknown data and achieve satisfactory CRs without increases in energy consumption.

In [16], the authors introduce an adaptive dynamic adjustment feedback mechanism for lossy compression of climate model data. They could achieve a higher CR under the same overall error by taking the local structural data characteristics into account. That shows the relevance of the data structure for optimal compression.

The compression algorithms perform very similar on different hardware architectures, like shown in [12]. Therefore,



(a) Compression Rate

(b) Compression Rate per Time

(c) Compression Speed

(d) Decompression Speed

Figure 3. Measurements showing compressors that were ideal for a specific metric when applied to intercepted writes.

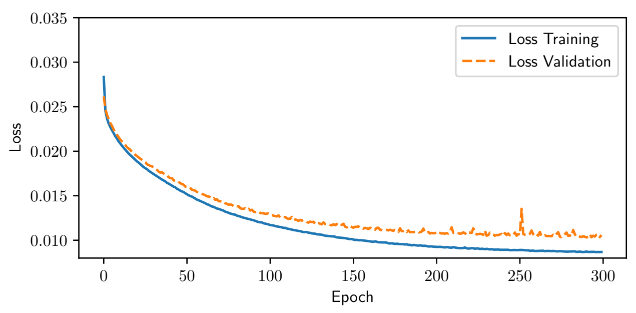


Figure 4. Plot of training and validation loss for compression ratio metric

heterogeneous model training should not affect the predictions validity, and the results can be applied across different systems.

6 Conclusion and Future Work

In our work we have applied ML techniques to dynamically choose the optimal compression algorithm for a specific data block. The block's size and content are intercepted from the application and represent the real world scenario. We show that the concept of identifying fitting compression algorithms and associated levels by leveraging common machine learning practices is promising. Optimal algorithms for complex metrics and specific needs on certain data are impossible to choose statically. It is therefore required to analyze data dynamically which helps building a trainable dataset for machine learning purposes. Furthermore, this gives valuable insights into how different compressors behave in general and in specific applications.

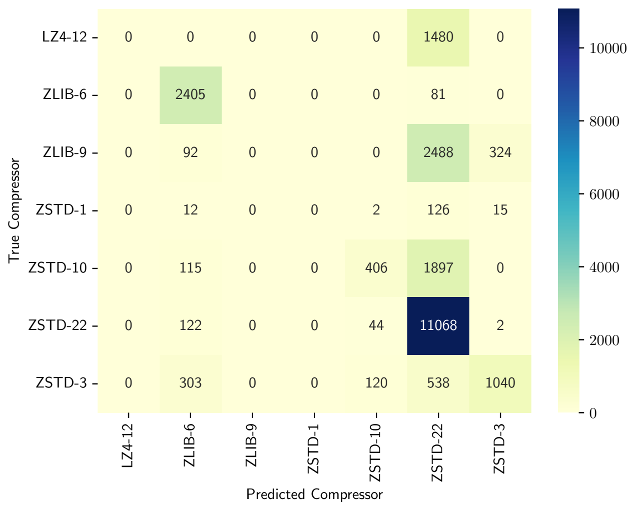


Figure 5. Confusion matrix after inferencing when using the compression ratio metric

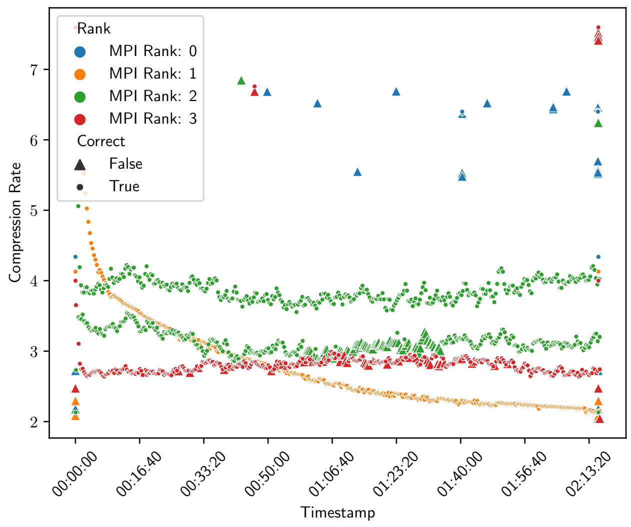


Figure 6. Plot of correct classified compressors over time and MPI rank

So far, we have trained for a limited number of well-known general purpose compression algorithms. More edgy algorithms like xz (very high CR, very low speed) or lz4-fast (highest speed with low CR) would complete the picture. Depending on the scientific domain, there exist more specialized algorithms with smaller comparative basis. Like in [7], some compressors for molecular sequence data can gain much higher CRs compared to ZSTD, but are much slower in decompression at the same time. Moreover, the algorithms have different needs for working memory amounts, up to unbounded growth. Image compression or lossy compression in general are worthwhile to analyze and may bring additional

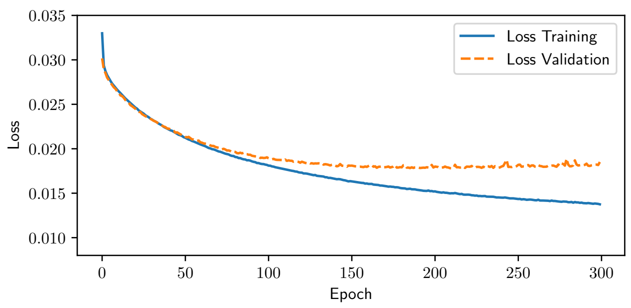


Figure 7. Plot of training and validation loss for decompression speed metric

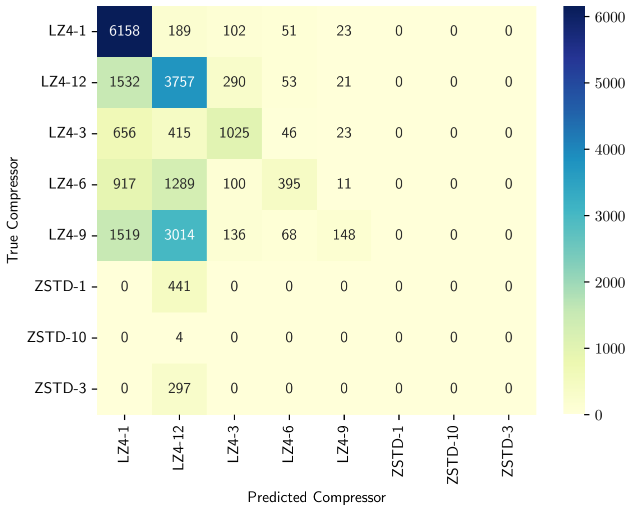


Figure 8. Confusion matrix after inferencing when using the decompression metric

challenges related to error bound metrics. These additional criteria and the variety of specific algorithms make it even harder to statically choose a universal compressor for all datasets and underlines the importance of our research.

Data analysis presented here ideally involves only fix costs for training runs. The produced application footprints and trained networks can be then directly reused for the same version of the application. It would be a great benefit to share these in an easily accessible way to avoid training of the same data for same application in a different institution. In the next step, it could be analyzed whether the gained knowledge for specific data structures is applicable to other applications without additional training.

Acknowledgments

This work used resources of the Deutsches Klimarechenzentrum (DKRZ) granted by its Scientific Steering Committee (WLA) under project ID ku0598.

References

[1] Cyan4973. 2020. LZ4 - Extremely fast compression. https://github.com/lz4/lz4

[2] Anna Fuchs. 2019. Enhanced Adaptive Compression in Lustre. https://wiki.lustre.org/Enhanced_Adaptive_Compression_in_Lustre

[3] Jean-loup Gailly Greg Roelofs and Mark Adler. 2022. OpenSFS Survey March 2021 Results. https://wiki.opensfs.org/images/2/20/OpenSFS_Survey_Results_March_2021.pdf

[4] Jean-loup Gailly Greg Roelofs and Mark Adler. 2022. zlib Technical Details. https://zlib.net/zlib_tech.html

[5] The HDF Group. 2018. Release of HDF5-1.10.2 - Newsletter #160. https://www.hdfgroup.org/2018/03/release-of-hdf5-1-10-2-newsletter-160/

[6] Flora Karniavoura and Kostas Magoutis. 2019. Decision-Making Approaches for Performance QoS in Distributed Storage Systems: A Survey. *IEEE Trans. Parallel Distributed Syst.* 30, 8 (2019), 1906–1919. https://doi.org/10.1109/TPDS.2019.2893940

[7] Kirill Kryukov, Mahoko Takahashi Ueda, So Nakagawa, and Tadashi Imanishi. 2019. Sequence Compression Benchmark (SCB) database — a comprehensive evaluation of reference-free compressors for FASTA-formatted sequences. *bioRxiv* (2019). https://doi.org/10.1101/642553 arXiv:https://www.biorxiv.org/content/early/2019/12/27/642553.full.pdf

[8] Michael Kuhn, Julius Plehn, Yevhen Alforov, and Thomas Ludwig. 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* (Lisbon, Portugal). IARIA XPS Press, 17–23. https://www.thinkmind.org/index.php?view=article&articleid=energy_2020_1_40_30038

[9] na. 2019. netCDF4 Version 1.6.0. https://unidata.github.io/netcdf4-python/

[10] na. na. IBM Spectrum Scale File compression. https://www.ibm.com/docs/en/spectrum-scale/5.1.0?topic=systems-file-compression

[11] Uli Plechschmidt. 2020. It's lonely at the top: Lustre continues to dominate top 100 fastest supercomputers. https://community.hpe.com/t5/Advantage-EX/It-s-lonely-at-the-top-Lustre-continues-to-dominate-top-100/ba-p/7109668

[12] Laura Promberger, Rainer Schwemmer, and Holger Fröning. 2021. Characterization of data compression across CPU platforms and accelerators. *Concurrency and Computation: Practice and Experience* n/a, n/a (2021), e6465. https://doi.org/10.1002/cpe.6465 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6465

[13] Shadura, Oksana, Bockelman, Brian Paul, Canal, Philippe, Piparo, Danilo, and Zhang, Zhe. 2020. ROOT I/O compression improvements for HEP analysis. *EPJ Web Conf.* 245 (2020), 02017. https://doi.org/10.1051/epjconf/202024502017

[14] Houjun Tang, Suren Byna, N. Anders Petersson, and David McCallen. 2021. Tuning Parallel Data Compression and I/O for Large-scale Earthquake Simulation. In *2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, December 15-18, 2021*. IEEE, 2992–2997. https://doi.org/10.1109/BigData52589.2021.9671876

[15] Chip Turner Yann Collet. 2016. Smaller and faster data compression with Zstandard. https://engineering.fb.com/2016/08/31/core-data/smaller-and-faster-data-compression-with-zstandard/

[16] Zhaoyuan Yu, Zhengfang Zhang, Dongshuang Li, Wen Luo, Yuan Liu, Uzair Bhatti, and Linwang Yuan. 2020. Adaptive lossy compression of climate model data based on hierarchical tensor with Adaptive-HGFDR (v1.0). (06 2020). https://doi.org/10.5194/gmd-2020-124

[17] Kai Zhao, Sheng Di, Xin Liang, Sihuan Li, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2020. Significantly Improving Lossy Compression for HPC Datasets with Second-Order Prediction and Parameter Optimization. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 89–100. https://doi.org/10.1145/3369583.3392688