Improving Energy Efficiency of Scientific Data Compression with Decision Trees

Michael Kuhn, Julius Plehn and Yevhen Alforov
University of Hamburg
Hamburg, Germany
Email: michael.kuhn@ovgu.de,juplehn@me.com,alforov@gmx.de

Thomas Ludwig
German Climate Computing Center
Hamburg, Germany
Email: ludwig@dkrz.de

***Abstract*—Scientific applications, simulations and large-scale experiments generate an ever increasing deluge of data. Due to the storage hardware not being able to keep pace with the amount of computational power, data reduction techniques have to be employed. Care has to be taken such that data reduction does not impact energy efficiency as it is an important cost factor for supercomputer systems and infrastructures. Data reduction techniques are highly data-specific and, therefore, unsuitable or inappropriate compression strategies can utilize more resources and energy than necessary. To that end, we propose a novel methodology for achieving on-the-fly intelligent decision making for energy-efficient data compression using machine learning. We have integrated a decision component into the Scientific Compression Library (SCIL) and show that, with appropriate training, our approach allows SCIL to select the most effective compression algorithms for a given data set without users having to provide additional information. This enables achieving compression ratios on par with manually selecting the optimal compression algorithm.**

Keywords–Data Compression; Energy Efficiency; Decision Tree.

I. Introduction

Even though the rate of improvement for processors has slowed down, the gap between the computational power of processors and other hardware components, such as main memory and storage, is still widening [1]. To keep pace, additional investments are necessary for storage hardware. This also results in additional costs for energy because new hardware used in computing systems requires additional power. Therefore, especially in data-intensive fields, costs for storage and energy are increasing. For instance, for each PetaByte (PB) of disk-based storage space, the German Climate Computing Center (Deutsches Klimarechenzentrum, DKRZ) has to pay investment costs of roughly 100,000 € and annual electricity costs of 3,680 €. For its 54 PiB storage system, this amounts to almost 200,000 € per year for electricity alone (one PB of storage needs 3 kW of power and 1 kWh of energy costs 0.14 €).

Data reduction techniques, such as compression, transformations and deduplication are straight-forward solutions to minimize the energy consumption of storage systems by reducing the amount of storage hardware required to store the same amount of data. However, data reduction itself can consume significant amounts of energy, potentially negating its beneficial effects on energy efficiency. While the energy efficiency of supercomputers should be increased, the impact on runtime performance should be minimal. A number of approaches and mechanisms to reduce energy consumption in supercomputers have been suggested at the different levels of computing systems. However, the impact of data reduction on High-Performance Computing (HPC) systems' energy efficiency remains largely unexplored, even though more and more HPC applications produce enormous volumes of data and data reduction techniques are increasingly adopted.

Developers of scientific software have a great interest in data reduction. However, to make best use of it, the used methods and algorithms have to be appropriate for their data sets and must be tuned to achieve optimal results. Additionally, decreasing runtime performance should be avoided both for performance reasons and for its impact on energy consumption. For these users, choosing a suitable compression algorithm is a technical decision that is difficult to make since the choice depends on the data set in question as well as the software and hardware environments. Data reduction schemes and options are highly data-specific and, therefore, our ultimate goal is to automatize the decision making process on behalf of the users. Poor manual choices can lead to low compression ratios, decreased performance and increases in energy consumption. In this paper, we are focusing on scientific applications in the context of HPC, where defining data reduction strategies with high performance and energy efficiency suitable for the generated deluge of scientific data is a challenging task.

Based on the methodology first introduced in [2], we define and extend mechanisms for intelligently selecting algorithms from a variety of state of the art reduction techniques with an emphasis on their energy consumption. In addition to employing machine learning to pick the most suitable compression strategy for a data set, users are able to specify additional criteria.

The remaining of the paper is structured as follows: In Section II, we give an overview of a common HPC I/O stack as it is used in earth system science. We introduce our framework for scientific data compression through high-level I/O interfaces in Section III. In Section IV, we present data collected for several compressors used to train our decision component. A detailed evaluation is performed in Section V using a real-world ecosystem simulation. After a review of related work in Section VI, we conclude in Section VII with a summary of our findings and describe our future work in Section VIII.

II. HPC I/O and Data Reduction

Applications typically use high-level I/O libraries to access data, which in turn uses I/O middleware to communicate with an underlying file system (see Figure 1). Two popular and common high-level I/O interfaces in the scientific community to access data in both serial and parallel manner are the *Hierarchical Data Format (HDF5)* and the *Network Common Data Form (NetCDF)*. They allow HPC applications written in various programming languages to manipulate and store data in a self-describing and portable way by using multidimensional arrays. Self-describing data formats contain a description of the file layout and are thus easy to share and distribute.

 ISBN: 978-1-61208-788-7

While NetCDF provides a convenient programming interface, the actual data format and I/O routines are implemented as part of HDF5. HDF5, in turn, uses the I/O implementation of the Message Passing Interface (MPI). MPI employs the I/O operations of the underlying parallel file system using optimized backends for a wide range of different file systems. In the end, I/O operations are posted by the file system to the underlying I/O driver. If the application performs data writing, it uses the high-level I/O library, and the data is going through the stack down until it is placed in the driver layer. A data read works in the opposite direction.

There are two main levels of the data path where data reduction mechanisms can be deployed: They are *system (low)* and *application (high)* levels, which each have different benefits and drawbacks. Data reduction usage on higher levels of HPC I/O stack is typically advantageous. Unlike low layers, it is possible to access and exploit additional meta information provided by the high-level I/O libraries. Different HPC applications for, e.g., climate change and weather forecasting, are using a common I/O stack, making it easier to employ application-level data reduction for them. Thus, usage of data reduction at the application level can be fine-tuned by taking application requirements and metadata into account. Among others, techniques such as deduplication, compression and transforms can be used on the application level. Moreover, as long as these techniques perform lossless data reduction, they can be deployed in a way that is transparent for users.

III. Energy-Efficient Data Compression

Based on these observations, we have extended the Scientific Compression Library (SCIL) to support energy-efficient data reduction by using machine learning approaches. SCIL already provides a rich set of features. In general, SCIL is a meta-compressor that aims to exploit knowledge on the application level [3]. The library should ultimately pick a suitable chain of algorithms satisfying the user's requirements. This is currently done based on the capabilities of the algorithms but has been extended by a decision component that can use different criteria, such as an energy-aware selection of the algorithms.

The overall architecture of using SCIL in scientific applications is depicted in Figure 1. Our main goal is providing the most appropriate data reduction strategy for a given scientific data set on the basis of semantical information and performance of algorithms. SCIL currently supports a wide range of lossless and lossy compression algorithms. Any application using the HDF5 data model can use SCIL via its HDF5 filter.

A. Decision Component

Instead of relying solely on user-provided hints, we have extended SCIL with a decision component that takes into account information about the data's structure to provide improved data reduction capabilities. The decision component uses machine learning techniques to infer which compression algorithms and settings are best suited for a given data set. To provide enough information for the decision component to use, a separate training step is necessary. Currently, training is done separately from application runs by post-processing existing output data. The output data set is split up into its individual variables, which are then analyzed. For each variable, information about achievable compression ratios, processor utilization, energy consumption, etc. is collected. This is done

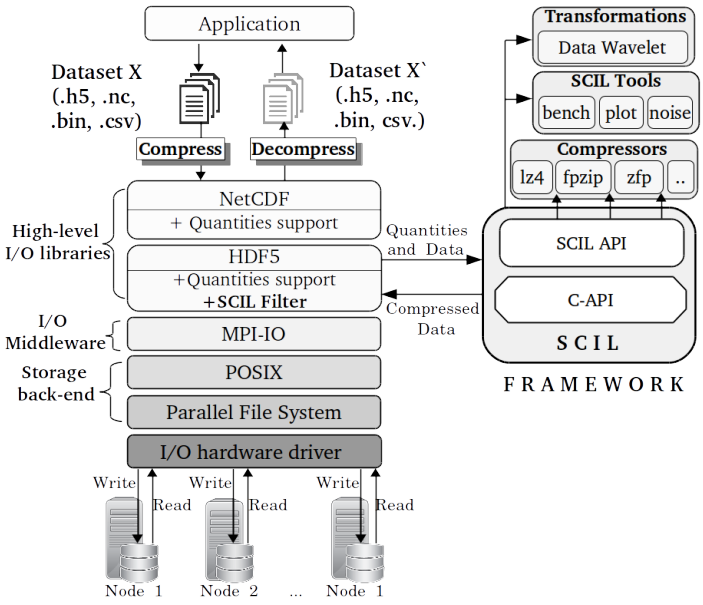

Figure 1. General architecture of Scientific Compression Library (SCIL) when integrated into the I/O workflow of scientific applications

for a wide range of data sets to provide the decision component with a sufficiently large pool of training data. The exact setup will be explained in more detail in Section V. Based on these measurements, it is possible to statically compute the appropriate compression algorithm for any given HDF5 variable. This information is then used to train the decision component, which currently makes use of decision trees but is planned to be extended with other techniques. In order to gain insights into the accuracy of those decision trees and to prevent overfitting, the data is split into a train and a test set. Those pre-processing steps are necessary in order to gather data that adds valuable information to the learning phase of the decision tree classifier.

The decision trees are created using the `DecisionTreeClassifier` component provided by scikit-learn, which produces a tree representation that is then parsed into a file that is usable in SCIL. Every time data is passed to SCIL, the decision component infers which compression algorithm and settings should be used before invoking its integrated compressors. The decision component's behavior can be separated into two distinct modes, as described below.

If the data's structure is known because it was part of the training set, the decision component can select the optimal compression algorithm and settings. This is currently done by comparing the data set's name but can be extended easily if necessary. This mode of operation is mainly useful for production runs of known applications. For instance, the decision component can be trained for a specific application and will be able to choose the best compressor for each subsequent run without the developers having to modify the application.

If the data's structure is unknown, the decision component will use machine learning techniques (currently, decision trees) to infer which compression algorithm and settings are best suited for the data set in question. The decision component will make use of information about the storage size, the number of elements, the data's dimensions, the data's type (float, double,

 ISBN: 978-1-61208-788-7

etc.) and other factors. Moreover, it is possible to influence the decision based on whether energy efficiency, compression ratio or performance should be prioritized.

IV. Training of the Decision Component

Our current approach requires collecting several metrics, such as compression ratio, processor utilization and energy consumption for each supported compressor. Since the data's structure can heavily influence a compressor's behavior, these metrics are collected per data set. This collected information can then be employed in selection of power-aware data reduction techniques for a given data set. In the next paragraphs, we will gather the distinct performance and energy consumption characteristics of a wide range of compression algorithms through the use of HDF5 filters. Algorithms like LZ4 [4] and Zstandard [5] are fast and provide high throughputs. However, their compression ratios can be lower compared to slower algorithms that consume more energy. To exclude the influence of SCIL and the computation present in the actual applications, the training step uses output data generated by several real-world applications and experiments. The output data is split up into individual data sets and compressed using HDF5's `h5repack` utility. Overall, the evaluation has been conducted by repacking the data 10 times to obtain averaged metrics.

Hardware and Software Environment: In order to investigate the performance of data compression at the application level, we used a cluster, which operates with the parallel distributed file system Lustre. Since performance is not a priority, only a single node outfitted with two 2.80 GHz quad-core Intel Xeon X5560 processors and 12 GB of RAM was used to collect the required metrics. Due to this, the maximum throughput was limited to roughly 110 MB/s. To capture each compressor's energy consumption, the ArduPower wattmeter was used [6]. It is designed to simultaneously measure the power consumption of different components (e.g., motherboard, CPU, GPU and disks) inside computing systems even at very large scale. ArduPower provides 16 channels to monitor the power consumption with a variable sampling rate of 480–5,880 Hz.

Metrics: The main metrics in which we were interested are the Compression Ratio (CR) to quantify the data reduction, runtime of each algorithm to see how slow or fast is it, and consumed energy. We define compression ratio as $CR = \frac{\text{original size}}{\text{compressed size}}$.

Data Sets: For the evaluation of data reduction techniques, we used three data sets from different scientific domains:

1) ECOHAM: 17 GB data set produced by the 3-dimensional ecosystem model for the North Sea ECOHAM [7] (from climate science)
2) PETRA III: 14 GB data set of tomography experiments from PETRA III's P06 beamline (PCO 4000 detector) [8] (from photon science)
3) ECHAM: 4 GB data set produced by the atmospheric model ECHAM [9] (from climate science)

Evaluated Techniques: To perform the reduction of data sets, different HDF5 compression filters have been leveraged. As part of our experimental evaluation, we have compared the following algorithms commonly used in HPC. While others, such as DEFLATE, LZMA or xz also offer high compression ratios, their performance is typically not sufficient for HPC workloads.

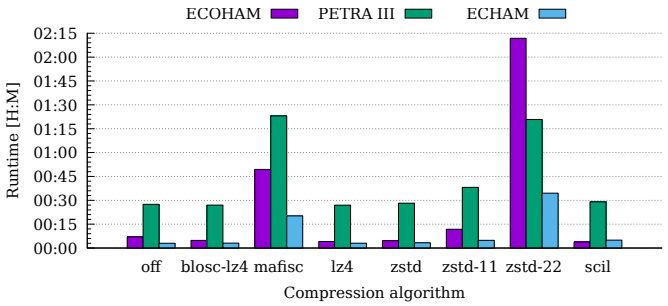

Figure 2. Runtime of evaluated compressors

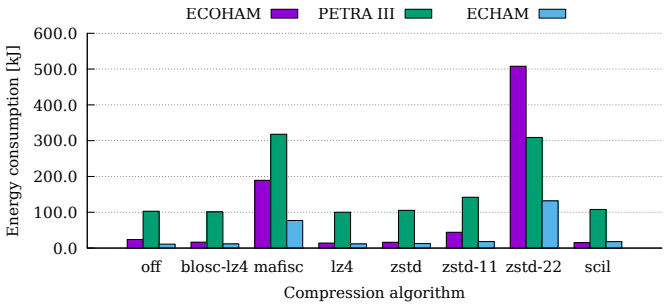

Figure 3. Average energy consumption depending on the HDF5 filter used for data compression

Note that the following results only include compression as decompression is usually much faster and thus negligible.

off: No filtering is applied. This represents the baseline. *blosc*: The Blosc meta-compressor using the LZ4 compressor. Additionally, Blosc's shuffle pre-conditioner was used. *mafisc*: The MAFISC compression algorithm that uses several pre-conditioners and the LZMA compressor. *lz4*: The LZ4 compression algorithm using its default acceleration factor. *zstd*: The Zstandard compression algorithm using its default aggression parameter. The *zstd-11* and *zstd-22* variants represent Zstandard with aggression parameters of 11 and 22, respectively. *scil*: SCIL's LZ4 compressor with some pre-conditioners.

Figure 2 shows that the runtimes vary wildly depending on the internal structure of the datasets. The consumed energy and obtained compression ratios for the ECOHAM, PETRA III and ECHAM data sets are plotted in Figures 3 and 4, respectively. It can be seen that different configurations achieve comparable

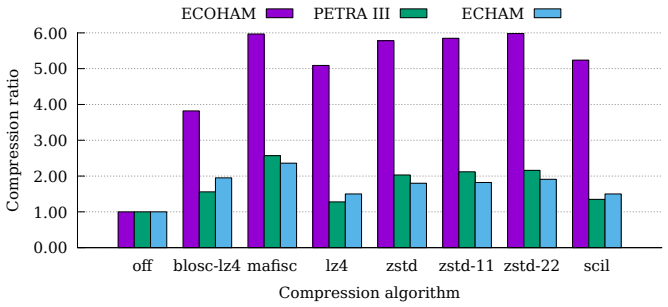

Figure 4. Average compression ratios depending on the HDF5 filter used for data compression

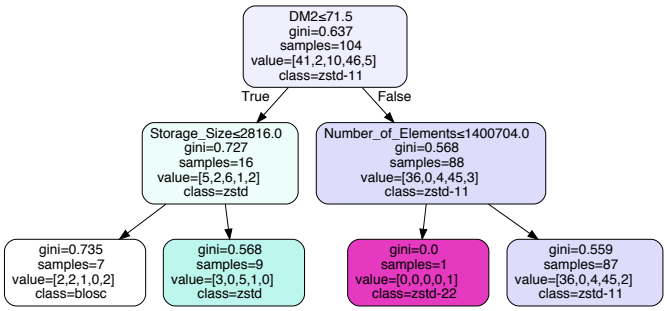

Figure 5. Decision component trained with ECOHAM data, optimized for maximal compression ratio per time

compression ratios with significantly different runtimes and energy consumptions (for instance, compare mafisc and zstd for the ECOHAM dataset). It is, therefore, necessary to select the compression algorithm intelligently to avoid wasting performance and energy. For a detailed analysis of the results for the ECOHAM and PETRA III datasets, please refer to [2].

V. EVALUATION

Compressors behave very differently depending on the data structure. Based on the results and data obtained in the previous section, we have trained the decision component using two different sets of training data and will run ECOHAM using our SCIL HDF5 plugin to evaluate our approach. All relevant code and data for this paper are available at [10].

ECOHAM data used as training data: This configuration represents the case that the application in question is known and has been run before on the system. The decision component has knowledge about this particular application's data and can take informed decisions regarding data reduction. However, only a random subset consisting of 75 % of ECOHAM's output data is used for training to make sure there is a degree of uncertainty left. This uncertainty could correspond to updated versions of the application or slightly changed output structure.

ECHAM data used as training data: This configuration represents the case that a new application is run on a system and the decision component has to use information gathered from other applications to try to compress the application's data as best as possible. In this case, the decision component will try to map decisions that make sense for other data sets to the current application's data.

Moreover, we will look at two different optimization targets for the decision component, which correspond to different use cases: First, we optimized for minimal energy consumption per compression ratio. This strategy allows shrinking the data with the least amount of energy possible, which is typically of importance to data center operators. Second, we optimized for maximal compression ratio per time. This strategy makes sure that performance is not degraded excessively, which is usually one of the main concerns of data center users. For instance, Figure 5 shows a configuration using ECOHAM training data with a decision tree that has been optimized for maximal compression ratio per time. As can be seen, a multitude of metrics are taken into account, including the array dimensions, the data set's storage size, as well as the number of elements. In addition to these metrics, the size of each dimension and

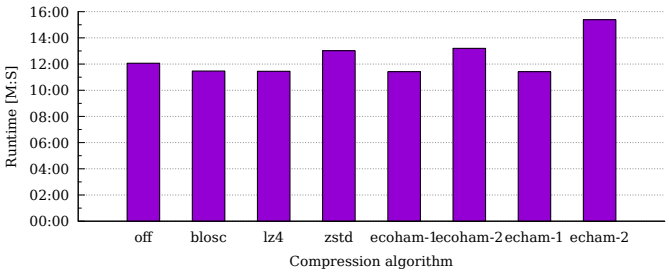

Figure 6. Runtime of evaluated compressors

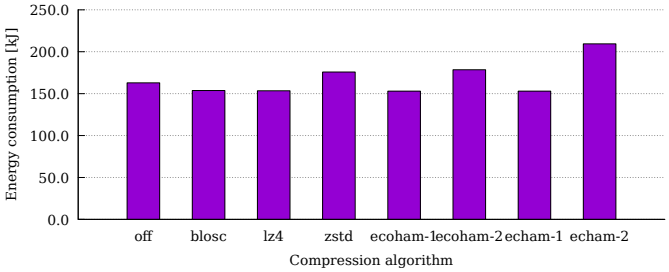

Figure 7. Energy consumption of evaluated compressors

information about data types can be used. In this specific case, the decision tree makes sure that a mix of the Blosc and Zstandard compressors are used. Moreover, Zstandard's compression level is adapted for maximum effect.

To compare our approach to static approaches, we have chosen to run ECOHAM with the most important compressors in static mode (that is, all data is compressed with the selected algorithm) and four of our decision trees. It is important to note that, although only Blosc, LZ4 and Zstandard are used as static approaches, the decision trees have access to all of SCIL's compressors, which also include different aggression parameters for Zstandard. Figures 6, 7 and 8 show the runtime, energy consumption and compression ratio of ECOHAM when run with the following configurations.

off: No HDF5 filter is used and, thus, no compression is performed. *blosc*: SCIL's HDF5 filter is configured to compress ECOHAM's data using Blosc. *lz4*: SCIL's HDF5 filter is configured to compress ECOHAM's data using LZ4. *zstd*: SCIL's HDF5 filter is configured to compress ECOHAM's data using Zstandard. *ecoham-1*: The decision component has been trained with ECOHAM data and is optimizing for minimal energy consumption per compression ratio. *ecoham-2*: The decision component has been trained with ECOHAM data and is optimizing for maximal compression ratio per time. *echam-1*: The decision component has been trained with ECHAM data and is optimizing for minimal energy consumption per compression ratio. *echam-2*: The decision component has been trained with ECHAM data and is optimizing for maximal compression ratio per time.

As can be seen in Figures 6 and 7, fast compression algorithms, such as LZ4 reduce runtime and energy consumption by causing less data being written to the file system and thus slightly speeding up the whole application. The same is true for the other light-weight algorithm Blosc and SCIL's default configuration. However, slower algorithms, such as Zstandard

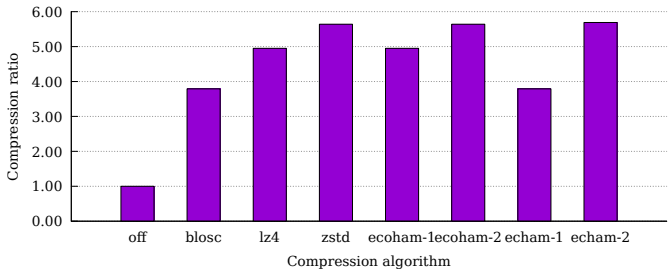

Figure 8. Compression ratio of evaluated compressors

have the opposite effect and cause increases in runtime and energy consumption. Most importantly, the decision component correctly uses energy-efficient algorithms when optimizing for minimal energy consumption per compression ratio, which can be seen in the *ecoham-1* and *echam-1* configurations. In the *ecoham-1* case, ECOHAM's data structure is known and this information can be used to effectively reduce the amount of data. In the *echam-1* case, however, the decision component only has knowledge about ECHAM's data structures but is still able to choose appropriate compressors. The *ecoham-2* and *echam-2* configurations both increase the runtime and energy consumption. This is expected since, in contrast to the previous cases, the optimization for maximal compression ratio per time puts more emphasis on data reduction instead of low energy consumption. Since the data structure is not known in the *echam-2* case, the decision component's choices are not as effective as in the *ecoham-1* case and increase runtime and energy consumption to a higher degree. Overall, the results show that the decision component can be used both for known as well as unknown applications. However, if the application's data structure are known, better decisions can be made.

This can also be seen in Figure 8, which illustrates the compression ratios achieved by all configurations. As expected, Blosc's compression ratio is the lowest (3.79), followed by LZ4 (4.95) and Zstandard (5.64). For *ecoham-1* and *ecoham-2*, the decision component has knowledge about ECOHAM's data structures and chooses the optimal algorithms for both optimization targets. When optimizing for low energy consumption (*ecoham-1*), the decision component favors LZ4 and achieves a compression ratio of 4.95. When a higher compression ratio is preferred (*ecoham-2*), Zstandard is chosen most of the time, leading to a compression ratio of 5.64. When the decision component has been training with ECHAM's data, the results are different: For *echam-1*, the decision component chooses Blosc most of the time, which achieves the goal of reaching a low energy consumption but boasts a lower compression ratio of 3.79. For *echam-2*, however, the decision component does end up using significantly more energy than *ecoham-2* but also provides a higher compression ratio of 5.69. This is due to the fact that the decision component also chooses Zstandard with higher aggression parameters.

In order to put the possible energy savings of this approach into perspective, the initially required energy consumption of the training has to be compared. For every variable existing in the data set, all available compression algorithms and selected compression levels are applied. In case of ECOHAM, 1,953 tests have been performed, resulting in 10,633 kJ being consumed. For ECHAM, 833 tests were necessary with a total energy consumption of 1,169 kJ. A comparison of the energy consumptions from Figure 7 shows that our approach can save roughly 10 kJ per run in contrast to using no compression. Even though these savings appear small, the presented runs were relatively short with 11–12 minutes. Production runs of these applications typically take several days or weeks and are repeated many times for comparison purposes. Therefore, significant cost savings are possible even though initial training costs appear high. Moreover, training costs can be decreased by eliminating algorithms and settings that prove to be inefficient.

VI. Related work

The results by Welton et al. [11] show that the achievable throughput is highly dependent on the chosen algorithm and data properties because slow algorithms or incompressible data can decrease throughput significantly. One way to compensate for this drawback is to implement these algorithms in hardware. Abdelfattah et al. [12] have implemented gzip on Field-Programmable Gate Arrays (FPGAs) using OpenCL, which offers higher throughput and a better performance-per-watt ratio. Intel's QuickAssist technology can also be used to lower the total cost of ownership by executing popular encryption and compression algorithms in hardware, as shown by Hu et al. [13]. However, all of these approaches still require developers to manually select a compression algorithm and settings. Inappropriate choices can lead to suboptimal performance and compression ratios. It is, therefore, important to foresee which reduction method will produce the best results. For example, Chen et al. [14] present a decision algorithm for MapReduce users to decide whether to use compression or not. They reported that compression provides up to 60 % energy savings for some jobs.

Machine learning techniques (especially Deep Learning) are being increasingly used to compress images and videos, as shown by Liu et al. [15]. They show that new approaches based on deep networks can produce comparable results to traditional coding approaches. Neural networks have also been used by Park et al. [16] to compress data gathered by Internet of Things devices in a lossy fashion. Rippel et al. [17] created a machine-learning-based approach for lossy image compression that outperforms traditional approaches, such as JPEG, JPEG 2000 and even WebP. A similar approach has been taken by Toderici et al. [18], who have created a compressor based on Recurrent Neural Network (RNNs) that also outperforms the traditional lossy JPEG compressor. The same is possible for lossless image compression, as shown by Mentzer et al. [19]: The proposed image compression system L3C outperforms PNG, WebP and JPEG2000. Machine learning can also be used for indirect space savings: Kraska et al. [20] have used machine learning to replace traditional data structures for b-trees, hash maps and bloom filters, which allowed reducing the amount of data needed by these data structures while simultaneously delivering competitive performance. However, in these cases, the actual compression is replaced using machine learning. This has the downside of not being able to perform lossless compression, which our approach can achieve.

VII. Conclusion

In this work, we have analyzed whether it is possible to automatically and intelligently pick compression algorithms for a given data set by making use of machine learning

 ISBN: 978-1-61208-788-7

techniques. Our results show that the amount of data which can be saved after using reduction techniques like compression heavily depends on the structure of data. Preconditioners, algorithms and settings might work well for one data set, but they might increase energy consumption for others. The preliminary results obtained during the training step have been taken into account when designing and implementing the decision unit for intelligent algorithms selection in SCIL. We have used fine-grained per-variable analyses to identify the optimal compression strategies for three different data sets and used this data to train the decision component for our real-world evaluation. We could demonstrate that the decision component is able to choose appropriate compressors for both known and unknown applications, which can be tuned for energy efficiency or compression ratio. Without providing additional information, the decision component was able to achieve satisfactory compression ratios without increases in energy consumption. Moreover, by changing the optimization strategy of the decision trees to allow slight increases in energy consumption, we could significantly boost compression ratios.

VIII. Future Work

Training currently has to be performed in a separate step. In the future, we envision training data collection to be more tightly integrated with production runs of applications. For instance, a specialized training mode of SCIL's HDF5 filter could be used to capture and analyze applications' output data during regular runs. In order not to influence application performance negatively, selected data samples could be sent to a training service that then takes care of analyzing it in more detail using a wide range of compression algorithms. For instance, a preloadable library, which intercepts calls to HDF5 and integrates filters into applications could offer this functionality in a transparent way. This would also allow us to extend our analysis to more compression algorithms and datasets. Since training is a manual process at the moment, we have focused on algorithms commonly used in HPC for now.

Additionally, the current interface used by HDF5 filters is too limiting to fully exploit all possibilities offered by our decision component. For instance, since HDF5 filters operate on opaque buffers, it is not easily possible to access single data points. However, this could be used gather further information about data variance, such as maximum and minimum values, which could be used to further tune compressor behavior. We will also experiment with chains of compressors. Applying multiple compressors in the correct order can lead to additional space savings. However, figuring out the order and suitable compressors is a combinatorial problem that is not easy to solve manually. Therefore, we want to extend the decision component to take this fact into account and predict chains of compressors instead of singles ones.

Acknowledgment

This work was supported in part by the BigStorage project, funded by the European Union under the Marie Skłodowska-Curie Actions (H2020-MSCA-ITN-2014-642963). We would also like to thank Anastasiia Novikova for her support with using the SCIL library and André Rothkirch from DESY for providing us with access to parts of their data.

References

[1] R. Chen, Z. Shao, and T. Li, "Bridging the I/O performance gap for big data workloads: A new NVDIMM-based approach," in 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016, pp. 9:1–9:12. [Online]. Available: https://doi.org/10.1109/MICRO.2016.7783712

[2] Y. Alforov, T. Ludwig, A. Novikova, M. Kuhn, and J. M. Kunkel, "Towards Green Scientific Data Compression Through High-Level I/O Interfaces," in 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018, Lyon, France, September 24-27, 2018, pp. 209–216. [Online]. Available: https://doi.org/10.1109/CAHPC.2018.8645921

[3] J. M. Kunkel, A. Novikova, E. Betke, and A. Schaare, "Toward Decoupling the Selection of Compression Algorithms from Quality Constraints," in High Performance Computing - ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers, pp. 3–14. [Online]. Available: https://doi.org/10.1007/978-3-319-67630-2_1

[4] Y. Collet, "LZ4," https://lz4.github.io/lz4/, 2020, retrieved: September, 2020.

[5] Facebook, "Zstandard," https://facebook.github.io/zstd/, 2020, retrieved: September, 2020.

[6] M. F. Dolz, M. R. Heidari, M. Kuhn, T. Ludwig, and G. Fabregat, "ArduPower: A low-cost wattmeter to improve energy efficiency of HPC applications," in Sixth International Green and Sustainable Computing Conference, IGSC 2015, Las Vegas, NV, USA, December 14-16, 2015, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IGCC.2015.7393692

[7] F. Große et al., "Looking beyond stratification: a model-based analysis of the biological drivers of oxygen depletion in the North Sea," Biogeosciences Discussions, 2015, pp. 2511–2535, retrieved: April, 2020. [Online]. Available: http://www.biogeosciences-discuss.net/12/12543/2015/bgd-12-12543-2015.pdf

[8] DESY, "PETRA III," http://petra3.desy.de/index_eng.html, 2015, retrieved: April, 2020.

[9] E. Roeckner et al., "The atmospheric general circulation model ECHAM 5," Max Planck Institute for Meteorology, 2003.

[10] M. Kuhn, J. Plehn, and Y. Alforov, "Supplementary Material for Improving Energy Efficiency of Scientific Data Compression with Decision Trees," https://github.com/wr-hamburg/energy2020-compression, 2020, retrieved: September, 2020.

[11] B. Welton et al., "Improving I/O Forwarding Throughput with Data Compression," in 2011 IEEE International Conference on Cluster Computing (CLUSTER), Austin, TX, USA, September 26-30, 2011, pp. 438–445. [Online]. Available: https://doi.org/10.1109/CLUSTER.2011.80

[12] M. S. Abdelfattah, A. Hagiescu, and D. Singh, "Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL," in Proceedings of the International Workshop on OpenCL, IWOCL 2013 & 2014, May 13-14, 2013, Georgia Tech, Atlanta, GA, USA / Bristol, UK, May 12-13, 2014, pp. 4:1–4:9. [Online]. Available: https://doi.org/10.1145/2664666.2664670

[13] X. Hu et al., "QTLS: high-performance TLS asynchronous offload framework with Intel QuickAssist technology," in Proceedings of the 24th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2019, Washington, DC, USA, February 16-20, 2019, pp. 158–172. [Online]. Available: https://doi.org/10.1145/3293883.3295705

[14] Y. Chen, A. Ganapathi, and R. H. Katz, "To compress or not to compress – compute vs. IO tradeoffs for MapReduce energy efficiency," in Proceedings of the 1st ACM SIGCOMM Workshop on Green Networking 2010, New Delhi, India, August 30, 2010, pp. 23–28. [Online]. Available: https://doi.org/10.1145/1851290.1851296

[15] D. Liu, Y. Li, J. Lin, H. Li, and F. Wu, "Deep Learning-Based Video Coding: A Review and A Case Study," CoRR, vol. abs/1904.12462, 2019. [Online]. Available: http://arxiv.org/abs/1904.12462

[16] J. Park, H. Park, and Y. Choi, "Data compression and prediction using machine learning for industrial IoT," in 2018 International Conference on Information Networking, ICOIN 2018, Chiang Mai, Thailand, January 10-12, 2018, pp. 818–820. [Online]. Available: https://doi.org/10.1109/ICOIN.2018.8343232

 ISBN: 978-1-61208-788-7

[17] O. Rippel and L. D. Bourdev, “Real-Time Adaptive Image Compression,” in Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, pp. 2922–2930, retrieved: April, 2020. [Online]. Available: http://proceedings.mlr.press/v70/rippel17a.html

[18] G. Toderici et al., “Full Resolution Image Compression with Recurrent Neural Networks,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, pp. 5435–5443. [Online]. Available: https://doi.org/10.1109/CVPR.2017.577

[19] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, “Practical Full Resolution Learned Lossless Image Compression,” CoRR, vol. abs/1811.12817, 2018, retrieved: April, 2020. [Online]. Available: http://arxiv.org/abs/1811.12817

[20] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The Case for Learned Index Structures,” in Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, pp. 489–504. [Online]. Available: https://doi.org/10.1145/3183713.3196909

 ISBN: 978-1-61208-788-7