Deliverable D2: Report

Coupled Storage System for Efficient Management of Self-Describing Data Formats (CoSEMoS)

Kira Duwe
kira.duwe@ovgu.de

Michael Kuhn
michael.kuhn@ovgu.de

September 6, 2022

Contents

1. Introduction and Motivation

Work on the CoSEMoS project started in mid-October of 2019 at University of Hamburg (UHH). Since then, CoSEMoS was moved from UHH to Otto von Guericke University Magdeburg (OVGU) in August 2020, where the work continues. The CoSEMoS project is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 417705296. More information can be found at `https://cosemos.de`.

This report reflects on the progress made in the second year of the CoSEMoS project. It will give a description for core features such as the global metadata management and the components for hierarchical storage management (HSM) in Section 2. Results published at SYSTOR 2021 are presented as well in Section 3. Lastly, the workshop outcome is discussed in Section 4.

The following paragraph is based on [Duwe and Kuhn, 2021a].

In times of continuously growing data sizes, performing insightful analysis is increasingly difficult [Duwe et al., 2020]. I/O libraries such as NetCDF and ADIOS2 offer options to manage additional metadata to make the data retrieval more efficient [Lofstead et al., 2008]. However, queries on this metadata are difficult as it is currently stored inside the corresponding self-describing data formats. By replacing the file system underneath with the storage framework JULEA, we can use dedicated backends for key-value and object stores, as well as databases [Kuhn, 2017]. Splitting the self-describing file into file metadata and file data enables novel and highly efficient data management techniques without creating redundancy. We have kept our approach transparent to the application layer by implementing custom ADIOS2 engines [Duwe and Kuhn, 2021a] and HDF5 VOL plugins [Kuhn and Duwe, 2020]. Moreover, we designed a data analysis interface (DAI) that allows speeding up metadata queries by a factor of up to 60,000 in comparison to the ADIOS2 API and data formats.

2. Design and Implementation

Parts of this section have been published in [Duwe and Kuhn, 2021a].

In the following, we explain the progress compared to the state documented in the first report. First, the storage tiering and HSM support is discussed. Afterwards, semantics are touched on shortly. Lastly, insights into the first DAI prototype are given.

2.1. Storage Tiering and HSM

In our last report, we discussed the reasoning behind changing our approach to the metadata backend selection and the storage tiering, that is, T2.3 and T2.4. An explanation of these tasks can be found in the previous report.

In the following, we give a short reminder. Neither the Global Metadata Manager (GMM) nor the Storage Tier Selector (STS) is realized as a separate layer or tool. The metadata management is implemented inside the I/O libraries instead. Both the ADIOS2 engines and the HDF5 VOL plugins can pass the metadata to a key-value or a database backend. Based on these interfaces, we can now track the applications' access patterns through logging the function calls inside of JULEA. Additionally, the option to visualize HDF5 access patterns has been developed to

inform the decision about the database schema design [Lobedank, 2021]. An enhanced version will be merged into JULEA in the future. Currently, more work is under way to visualize the actual on-disk data layout of HDF5 files. Thereby, we will gain more information about the structural composition and also new insights how to efficiently place data in an HSM system.

Figure 1 illustrates the current HSM support. The separation of file metadata and data allows us to make use of JULEA's versatile configuration features. This allows the database to be stored on faster hardware than the object store without the requirement of large amounts of fast hardware. That is possible because the database does not hold the data and, therefore, has a small size compared to the size of the data or the original file. As the different backends can be moved independently already, the implementation of a tiering mechanism to exploit HSM potential was a logical next step. It is discussed in the following. Also, the possibility to create multiple instances of a backend is realized.

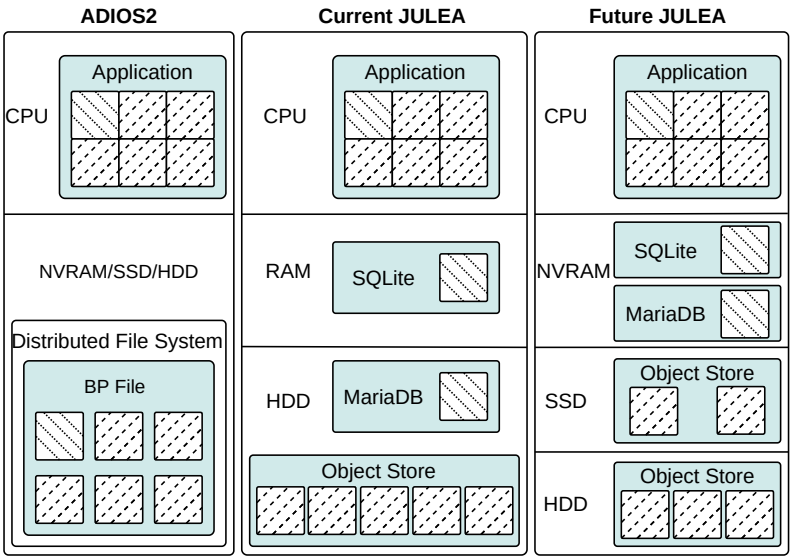

Figure 1: Different storage components on corresponding hardware layers: ADIOS2 writes its BP files into a parallel file system where they are stored as a byte stream across the storage devices the file system is configured to use. It is not easily possible to put different parts of the BP file onto separate storage devices (left). JULEA's design makes it possible to easily separate data and metadata, and handle it differently. In the current setup, the SQLite database is held in main memory while MariaDB and the object store reside on HDDs (middle). In the future, we will extend JULEA's support for storage hierarchies to be able to make full use of new technologies such as NVRAM. This will also allow us to put parts of the object store onto SSDs for fast access, while the majority can be kept on HDDs (right).

Based on this current design, we decided to combine the global metadata management and the tier selection for the HSM. Thus, we are building only one component, handling the metadata and data movements, which is called *Movement Handler (MH)* for now.

Adding HSM Support to JULEA In the future, ideally clients will be able to specify requirements for metadata, such as read-update-write ratios, access patterns (including the types of queries required), latencies, and throughputs (T1.1). Additionally, the MH should take into account the I/O semantics when selecting the metadata backend (T1.2); for instance, depending

on the atomicity and consistency requirements, specific database solutions are more appropriate than others. The MH will decide which backend type to use for a given set of metadata and its associated requirements. For this decision, it will include both information about the performance characteristics of the used database software and the underlying hardware. Automating this process would be very helpful but it has not yet been realized. However, the JULEA benchmarks have been extended to include further metrics about the latency. Besides minimum and maximum, the average, and the standard deviation are provided now, as well as the 1st, 10th, 90th and 99th percentiles. This allows getting a better overview of the performance behavior of JULEA's different backends.

Moreover, the MH will also take the backend type and configuration into account for its decisions. That will allow for mixing multiple appropriate technologies. While JULEA can provide all data storage functionality itself, it is also possible to integrate existing infrastructure. For instance, while node-local storage might use a simple object store backend on top of an SSD, the global storage could be provided using the POSIX backend on top of an existing parallel file system such as Lustre. As a first step in the direction of the MH, we introduced a policy module. It is implemented in the form of the `JBackendStack` in JULEA.

Integration into Current Object Backend In the following, the updated configuration is shown in Listings 1 and 2. Now multiple instances of a backend can be used. In addition, capacity, latency, and bandwidth can be configured for each tier. These parameters are optional and can be omitted to avoid incompatibility. The chosen encoding is a suffix representation to keep the configuration files human-readable.

```
[object]
backend=posix      |  backend=posix;posix
component=server   |  component=server
path=/tmp/object   |  path=/tmp/object;/mnt/fast
                   |  capacity=100MB;500MB
                   |  latency=50us;20ms
                   |  bandwidth=1GB;200MB
```

Listing 1: Current JULEA configuration for object backend (left) and definition of multiple object backends in new version (right)

Furthermore, a new configuration section extends the existing object backend configuration to include policies. The name of this section is `object.hsm-policy` to highlight that this policy is a HSM policy for the object backend as shown in Listing 2.

```
[object.hsm-policy]
kv_backend=leveldb
kv_path=/tmp/object-hsm-kv
policy=lru
args=200;5
```

Listing 2: New configuration group for policy

Additionally, a basic prototype for HSM policies has been designed and developed. Simple policies such as LRU have been implemented so far.

2.2. Adaptable Semantics

An in-depth evaluation of the semantics requirements of HPC applications has been performed in [Wang et al., 2021]. The authors examined whether the assumptions hold true that these applications do not require POSIX semantics as there has not been any large-scale study. They split the different semantics requirements into the following categories: strong consistency, commit consistency, session consistency and eventual consistency. Furthermore, they investigated the applications' access patterns and found three high-level patterns: consecutive, strided and strided cyclic. We will take these results as a foundation to implement further semantics templates for JULEA.

2.3. ADIOS2 Engines and HDF5 VOL Plugins

The core component of our work is the design and implementation of an ADIOS2 engines and HDF5 VOL plugins that handle the separation of the file data and file metadata. Figure 2 shows one of the JULEA configurations that we evaluated in detail. The application is distributed across two compute nodes using MPI. It uses the ADIOS2 library for its I/O, internally employing our engine that uses the JULEA client to forward the metadata and data to different backends. Note that the original BP file is not stored as a file anywhere in JULEA. The data is split into chunks and spread across the distributed object store servers, similar to Lustre's striping. The object store makes use of the underlying local file system to store the data. The metadata, however, is stored in a database. We do not use a distributed database for now, as the file metadata size is small enough to fit into a single instance. Thereby, we can avoid the overhead of keeping a distributed database consistent. The performance bottleneck for a large number of MPI processes can be mitigated by moving the database to a faster hardware layer which is shown in Figure 1. Nevertheless, we will examine the possibilities for a distributed database.

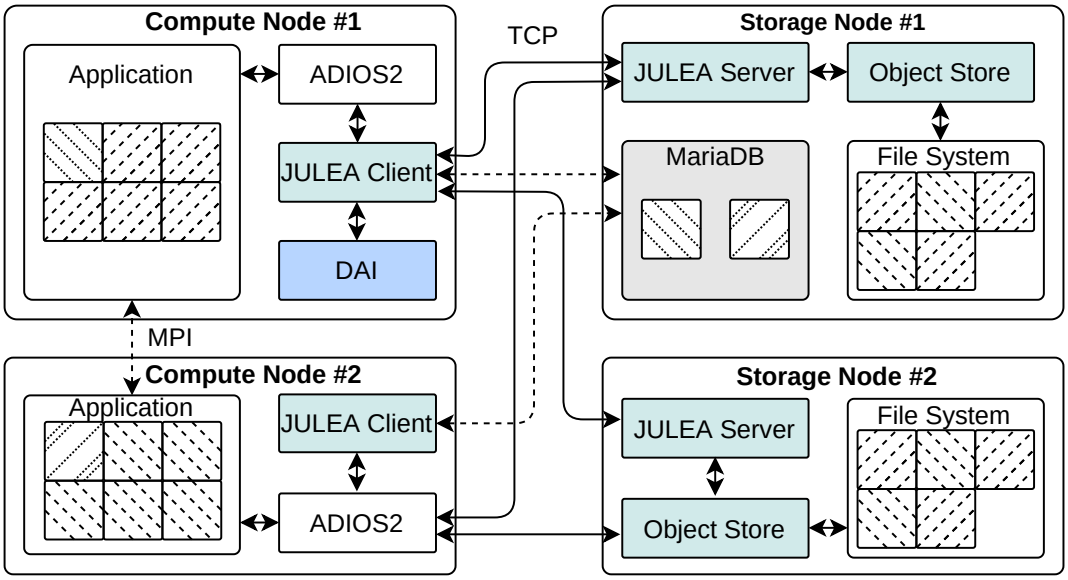

Figure 2: Benchmark setup with two compute and storage nodes.

Data Distribution Figure 2 also shows how the application output is split into file metadata and data and then stored separately. In the evaluated configuration, SQLite is run in the main

memory while MariaDB resides on HDDs. In the future, JULEA will make full use of new technologies such as NVRAM to store the databases. The data chunks are identified by a unique ID that is assigned to the specific combination of the file name, variable name, step, and block. This allows fine-grained access to the data. Data identification through the ID also means that migrating the database or the object store is uncomplicated as it does not require path updates. We evaluated SQLite and MariaDB as the database backends.

JULEA-DB Engine We have implemented two ADIOS2 engines (JULEA-KV and JULEA-DB) to realize the format dissection. The JULEA DB backend prototype has been implemented in [Straßberger, 2019] and [Warnke, 2019]. In the following, we will focus on JULEA-DB. In ADIOS2, the metadata for a variable includes the global minimum and maximum while each block saves the local ones. We store the step and block information as well as the dimensions of the data arrays and their location in the global MPI space shared across all processes.

The computation of the mean value is not part of the original ADIOS2 format. It is added to highlight the gained flexibility over the SDDFs. We see great potential in the option to customize additional file metadata where users can specify values or expressions that should be calculated and stored in the database for faster post-processing. It will be part of the extended DAI functionality in the future that will offer to pre-compute additional and derived metadata as well as tag areas of interest. Currently, our data analysis interface (see Figure 2) is a rudimentary prototype. Any custom metadata can only be accessed through the JULEA client and not the original I/O libraries as this would have required changes to the user-facing interface. This means custom metadata will be lost when the original file formats are exported.

3. Evaluation of ADIOS2 Engines and DAI

This section has been published in [Duwe and Kuhn, 2021a].

We evaluated the performance of the JULEA-DB engine using SQLite and MariaDB in comparison to the BP3 and BP4 engines. Our main focus was not necessarily to achieve a competitive performance but to illustrate the value of the format dissection for analyzing and post-processing. However, as I/O constitutes one of the most severe bottlenecks in HPC systems, the writing performance is still very relevant.

Setup Each compute node is equipped with 4 × AMD Opteron 6344, 128–256 GB of main memory and a 1 TB HDD (with a maximum throughput of roughly 130 MB/s). The Lustre system uses 10 × 2 TB HDDs for a total capacity of 20 TB, while the metadata is stored on a single 160 GB SSD for fast access. The compute nodes are connected to the Lustre nodes via 1 Gbit/s Ethernet but also have a 40 Gbit/s InfiniBand connection with each other. Since the compute nodes are not equipped with SSDs, we opted to put SQLite databases into the main memory. MariaDB was run on one of the compute nodes' HDDs, though.

3.1. Write and Read Performance

To evaluate the write and read performance, we used the heat transfer application from the ADIOS2 examples [Oak Ridge National Laboratory, 2018]. It solves the 2D Poisson equation using finite differences for the temperature distribution in homogeneous media. The data rates shown in Figure 3 are the mean values of the individual I/O times of each process per step over all steps. To reduce cache influences, we dropped the cache in between writing, reading, and querying. Also, we explicitly sync the data so that it is written in the measured time frame. In the case of JULEA, we synchronize the writing of every block, that is, the data of each process. For the BP engines, we adapted the POSIX file transport to flush at the end of each step. This means, that the BP engines are synced less than JULEA which needs to be considered when comparing the results. Note that we have ten storage nodes and only four compute nodes. When using BP3 and BP4, we write to the Lustre storage nodes. However, when evaluating the JULEA-DB engine, we cannot use them as we replaced Lustre with JULEA to simplify the I/O stack. We can give a rough estimate of this impact as we performed measurements using one node where BP3 and BP4 wrote to both the local HDD and Lustre. The local HDD is limited in terms of parallel accesses in contrast to the ten storage nodes of Lustre.

The most notable difference was reached using 24 processes for BP4 where local writes achieved 30 MB/s in contrast to 54 MB/s using Lustre. When reading, Lustre achieved 60 MB/s and thus about double the local performance of 30 MB/s. We also evaluated different matrix sizes and found that all configurations behaved similarly for sizes 1024^2, 2048^2, and 4096^2. So, we only present the results of the largest matrix size as it is the most relevant for HPC systems.

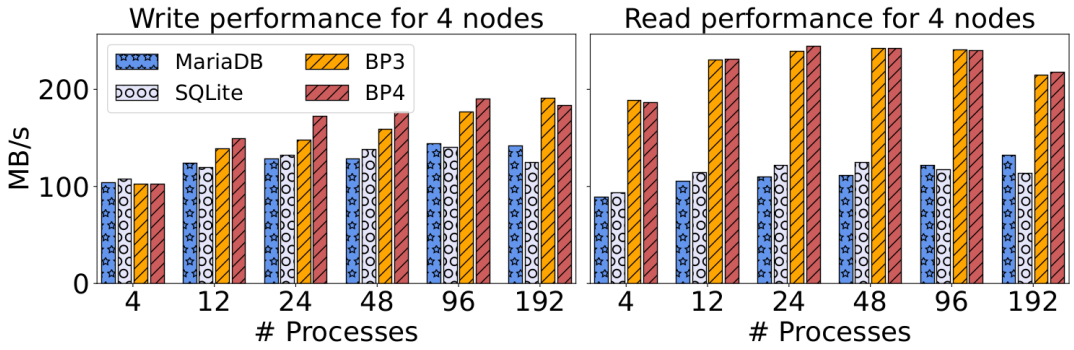

Figure 3: Write and read performance for 4 nodes with 1 to 48 processes per node and a matrix size of 4096^2.

In Figure 3 the data rates for four nodes are shown. The BP data rates grow as expected by a factor of four. JULEA, however, only achieves an improvement by a factor of two for several reasons. One is the mentioned disadvantage of the local storage in contrast to Lustre. Also, the very strict data syncing, the overhead of the TCP/IP network stack and the object store being emulated by POSIX contribute to it. Both the reading and writing performance for JULEA unsurprisingly benefit from running the database in the RAM in the case of SQLite compared to the MariaDB server running on the HDD.

3.2. Query Time

The main focus here is to demonstrate the value of the format dissection it has for querying and post-processing. For this, we developed two query applications, ADIOS2-Query and JULEA-

Query. The question we want to answer in our example query is: *What block does have the largest difference between its mean value in step 1 and step 5?*

ADIOS2-Query uses the ADIOS2 API to query the data. To answer the question, it needs to read the complete data of step 1 and step 5, and compute the mean value for each block. It then needs to determine the index of the block, with the largest difference between its two mean values. The matching results for the variable steps 1 and 5 are read from the file in Lustre.

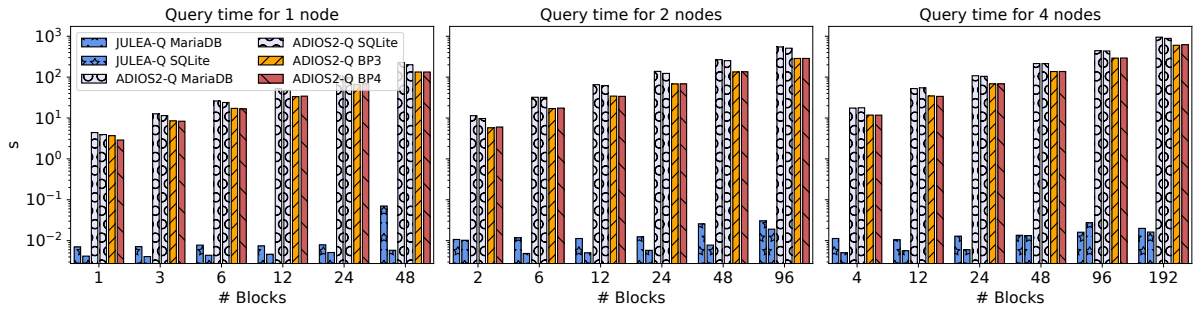

Figure 4: Query time for ADIOS2-Query and JULEA-Query for 1, 2 and 4 nodes. The number of blocks equals the number of processes for writing and reading. The matrix size is 4096^2. Note the log scale for the time (y-axis).

To validate the usefulness of the data analysis interface (DAI) depicted in Figure 2 we also wrote a query application directly interfacing the JULEA storage system. Using the JULEA API, we can access the mean value that has been precomputed by the JULEA-DB engine when writing the data blocks. This query application, therefore, does not need to access the object store containing the data. It is sufficient to read the mean values from the respective database and compute the differences accordingly to ADIOS2-Query. The measured time contains the time to read the mean values and the computation time. The time to precompute the means is captured in the writing performance of the JULEA engine. In Figure 4 the query time is depicted for 1, 2 and 4 nodes. The number of blocks is determined by the number of MPI processes of the writing application and is therefore identical to the labels of the x-axis of Figure 3. As Figure 4 clearly shows, JULEA-Query is incredibly fast and does at maximum not even take a second to finish. To have a direct comparison, we not only evaluated the BP engines for ADIOS2-Query but also the JULEA-DB engine. Using the ADIOS2 API, there is no way to access the additional metadata stored in the database. Therefore, the JULEA-DB query time consists mostly of the time it takes to read steps 1 and 5, which also applies to the BP engines. When querying 192 blocks, both SQLite and MariaDB take 0.01s while BP3 and BP4 need 621s and 601s, respectively. This is a performance improvement of roughly 60,000 when using our engine with the DAI layer.

3.3. Conclusion and Future Work

In summary, we were able to show the possible performance improvement for data querying that can be achieved by separating the BP file format into file metadata and file data. This dissection has allowed us to exploit storage characteristics for improved performance, especially for query scenarios. Specifically, the database can be stored on faster storage technologies such as SSDs, while the object store can use traditional HDDs. We have kept our approach transparent to the application layer by implementing a custom ADIOS2 engine. By storing the

file metadata in the appropriate backends in JULEA, we also gain the flexibility to introduce new metadata that can be precomputed and stored along with existing metadata, such as the minimum and maximum. This would not be possible otherwise without changing the file format itself. Our prototype engine precomputes the mean value of each block to demonstrate the possibility of metadata extension. Moreover, our data analysis interface allows speeding up metadata queries by a factor of up to 60,000 compared to the ADIOS2 API and data format.

One of the key aspects we will focus on in the future is to extend the data analysis interface (DAI) so that users can specify values or custom operations that should be calculated and stored in the database for faster post-processing. For common operations, it might then not be necessary to access data at all or at least cut down the number of blocks considerably, as shown by the JULEA-Query application. We will work on supporting external tools such as the climate data operators (CDO) [Kaspar et al., 2010].

Furthermore, there are several aspects in JULEA that need improvement. The object store currently sits on top of a full-featured POSIX file system, which we are planning to optimize in the future by using a proper object store backend. Work was done on an object store backend using Ceph's BlueStore [Aghayev et al., 2019] to avoid POSIX whenever possible [Duwe and Kuhn, 2021b]. Also, the network communication is based on TCP/IP, considerably increasing the overhead. Work is under way to make use of libfabric for enhanced networking support. Based on our concept of dissecting SDDFs, we will use structural information in the form of file metadata for intelligent hierarchical storage management.

4. Workshops

Since the last report in March 2021, two workshops have been held, one at EuroSys 2021[1] [Kuhn et al., 2021] and another one at EuroSys 2022[2] [Kuhn et al., 2022]. They were organized together with storage experts from DDN (Jean-Thomas Acquaviva, Konstantinos Chasapis) and ENSTA Bretagne (Jalil Boukhobza). The Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS) is a merger of the WOPSSS workshop previously held at ISC-HPC in Frankfurt and the CHAOSS workshop. By combining them, we will reach a broader audience and increase the visibility of the workshop and its topic. CHEOPS 2021 attracted more than 60 participants with its online format and received eight submissions, six of which were accepted. In 2021, Suren Byna from LBNL gave the keynote about *Parallel I/O at Exascale with HDF5 and Proactive Data Containers*. CHEOPS 2022 was mostly in-person with a few online attendees and attracted more than 20 participants overall. It received seven submissions, five of which were accepted. This year's keynote was given by Darrell Long talking about *Lethe – Learning how to forget*.

4.1. Other Cooperations and Exchange

The insights that we hoped to get from organizing a workshop specifically addressing our project topics had to be gained differently due to the uncertainties of workshop organization during the pandemic. Talks with developers from the Max Planck Institute for Meteorology[3]

[1] https://cheops-workshop.github.io/2021.html
[2] https://cheops-workshop.github.io/2022.html
[3] https://mpimet.mpg.de/

and the German Climate Computing Center[4] helped to clear requirements from the application side. Furthermore, the presentation of the thesis proposal as well as the doctoral symposium at Otto von Guericke University Magdeburg provided feedback from various professors inside and outside the HPC community.

4.2. Survey of Application Users

Furthermore, a user study was performed among scientists from research institutes around the world. It consists of three parts. Background information about the users such as the scientific field they work in came first. The second part was concerned with the usage of self-describing data formats, ranging from questions which formats are used to specific details such as the typical number of variables in a file. Finally, the third part is about the users' preferences regarding data access interfaces and typical post-processing operations. The results will be published after the thesis is written.

References

[Aghayev et al., 2019] Aghayev, A., Weil, S. A., Kuchnik, M., Nelson, M., Ganger, G. R., and Amvrosiadis, G. (2019). File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In Brecht, T. and Williamson, C., editors, *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 353–369. ACM. (Cited on page 9)

[Duwe and Kuhn, 2021a] Duwe, K. and Kuhn, M. (2021a). Dissecting self-describing data formats to enable advanced querying of file metadata. In Wassermann, B., Malka, M., Chidambaram, V., and Raz, D., editors, *SYSTOR '21: The 14th ACM International Systems and Storage Conference, Haifa, Israel, June 14-16, 2021*, pages 12:1–12:7. ACM. DOI: 10.1145/3456727.3463778. (Cited on pages 2 and 6)

[Duwe and Kuhn, 2021b] Duwe, K. and Kuhn, M. (2021b). Using Ceph's BlueStore as Object Storage in HPC Storage Framework. In Kuhn, M., Duwe, K., Acquaviva, J., Chasapis, K., and Boukhobza, J., editors, *CHEOPS '21: Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems, In Conjunction with EuroSys 2021, Online Event, United Kingdom, April, 2021*, pages 3:1–3:6. ACM. DOI: 10.1145/3439839.3458734. (Cited on page 9)

[Duwe et al., 2020] Duwe, K., Lüttgau, J., Mania, G., Squar, J., Fuchs, A., Kuhn, M., Betke, E., and Ludwig, T. (2020). State of the Art and Future Trends in Data Reduction for High-Performance Computing. *Supercomput. Front. Innov.*, 7(1):4–36. DOI: 10.14529/jsfi200101. (Cited on page 2)

[Kaspar et al., 2010] Kaspar, F., Schulzweida, U., and Mueller, R. (2010). "Climate data operators" as a user-friendly processing tool for CM SAF's satellite-derived climate monitoring products. (Cited on page 9)

[4] https://www.dkrz.de/

[Kuhn, 2017] Kuhn, M. (2017). JULEA: A Flexible Storage Framework for HPC. In Kunkel, J. M., Yokota, R., Taufer, M., and Shalf, J., editors, *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*, volume 10524 of *Lecture Notes in Computer Science*, pages 712–723. Springer. (Cited on page 2)

[Kuhn and Duwe, 2020] Kuhn, M. and Duwe, K. (2020). Coupling Storage Systems and Self-Describing Data Formats for Global Metadata Management. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1224–1230. DOI: 10.1109/CSCI51800.2020.00229. (Cited on page 2)

[Kuhn et al., 2021] Kuhn, M., Duwe, K., Acquaviva, J., Chasapis, K., and Boukhobza, J., editors (2021). *CHEOPS '21: Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems, In Conjunction with EuroSys 2021, Online Event, United Kingdom, April, 2021*. ACM. (Cited on page 9)

[Kuhn et al., 2022] Kuhn, M., Duwe, K., Acquaviva, J., Chasapis, K., and Boukhobza, J., editors (2022). *CHEOPS@EuroSys 2022: Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems, Rennes, France, 5 April 2022*. ACM. (Cited on page 9)

[Lobedank, 2021] Lobedank, K. (2021). HDF5-Zugriffsmusteranalyse zur Datenbankabstrahierung. (Cited on page 3)

[Lofstead et al., 2008] Lofstead, J. F., Klasky, S., Schwan, K., Podhorszki, N., and Jin, C. (2008). Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In Kim, Y. and Li, X., editors, *6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE@HPDC 2008, Boston, MA, USA, June 23, 2008*, pages 15–24. ACM. (Cited on page 2)

[Oak Ridge National Laboratory, 2018] Oak Ridge National Laboratory (2018). ADIOS 2: The Adaptable Input/Output System version 2 - Documentation. `https://adios2.readthedocs.io/en/latest/index.html`. Accessed: 2020-04-30, Revision 559a7fb7. (Cited on page 7)

[Straßberger, 2019] Straßberger, M. (2019). Structured metadata for the JULEA storage framework. Bachelor's thesis, Universität Hamburg. `https://wr.informatik.uni-hamburg.de/_media/research:theses:michael_strassberger_structured_metadata_for_the_julea_storage_framework.pdf`. (Cited on page 6)

[Wang et al., 2021] Wang, C., Mohror, K., and Snir, M. (2021). File System Semantics Requirements of HPC Applications. In Laure, E., Markidis, S., Verbanescu, A. L., and Lofstead, J. F., editors, *HPDC '21: The 30th International Symposium on High-Performance Parallel and Distributed Computing, Virtual Event, Sweden, June 21-25, 2021*, pages 19–30. ACM. (Cited on page 5)

[Warnke, 2019] Warnke, B. (2019). Integrating self-describing data formats into file systems. Master's thesis, Universität Hamburg. `https://wr.informatik.uni-hamburg.de/_media/research:theses:benjamin_warnke_integrating_self_describing_data_formats_into_file_systems.pdf`. (Cited on page 6)