Deliverable D1: 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

March 8, 2021

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 first year of the CoSEMoS project. Firstly, it will give a brief overview of the project, including its motivation and goals, in Section 1. It will detail the architecture and the interface specifications in Section 2. Furthermore, a description is given for core features such as the global metadata management and the components for HSM (Hierarchical Storage Management). Lastly, the workshop outcome is discussed in Section 3.

1 Project Motivation and Goals

Many scientific areas have become data-intensive. Large simulations and high volume streaming systems generate rapidly growing data sets that are increasingly difficult to manage and find insights in. The amount of data produced globally doubles approximately every two years, resulting in exponential growth. For these growing data sets, time to analyze results is driven based on the ability to sift through the data volumes most efficiently.

Besides the complexity introduced through hierarchical storage systems, the software stack impedes data retrieval as well. In the HPC (High-Performance Computing) environment, a typical I/O (Input/Output) stack looks similar to Figure 1. As can be seen, the stack is separated into layers that are mostly isolated from each other. While this allows exchanging individual layers, it leads to performance and management issues. For instance, different layers may pursue different directions on how and where to optimize the data access. One such example is MPI-IO's Two-Phase protocol, which tries to work around performance problems caused by parallel file systems.

To make filtering raw data easier, I/O libraries such as NetCDF (Network Common Data Format), HDF5 (Hierarchical Data Format) and ADIOS (Adaptable IO System) [Lofstead et al., 2008] are used. Unfortunately, the associated self-describing data formats (SDDF), such as BP (Binary Packed), are storage layer agnostic, leaving significant potential performance untapped.

Additionally, most parallel file systems offer a POSIX (Portable Operating System Interface) I/O interface, which enforces strict semantics that can severely impact the performance in distributed environments. For example, realizing atomic updates requires heavy use of locking strategies. Moreover, POSIX treats file data as an opaque byte stream, which makes it impossible to utilize structural information on the lower layers of the I/O stack.

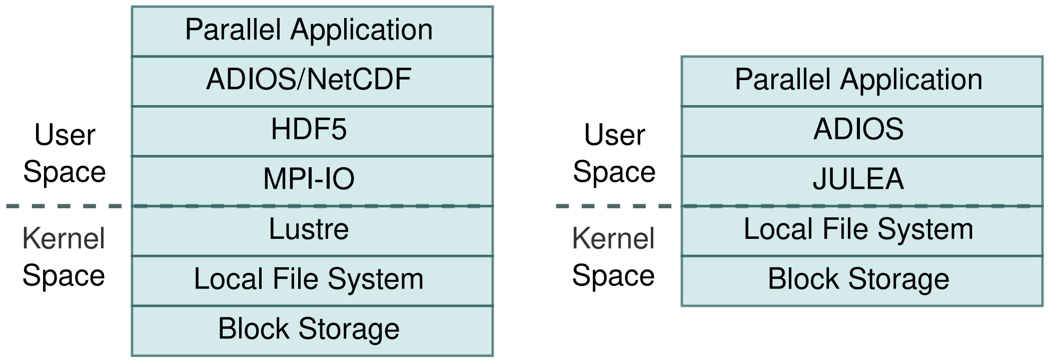

Figure 1: Exemplary HPC I/O stack with I/O libraries, self-describing data formats, I/O middleware and parallel file system for the current state (left) and when using JULEA (right).

1.1 Problem Summary

In order to keep up with the increasing data volumes, new data analysis strategies are required that enable sifting through the data fast by querying metadata. The solution needs to be transparent to the application layer and the users, otherwise, wide-spread deployment is very unlikely. Also, it must be flexible enough that upcoming analysis tools can work with them, as it is not clear which data features might be of interest for future research. Lastly, it must not only work on tiered storage hierarchies but take advantage of the hardware features.

1.2 Proposed Solution

As a first step, we replace the file system below, with the storage framework JULEA [Kuhn, 2017]. Figure 1 shows the resulting I/O stack of this substitution, on the right, which is less complex and therefore easier to manage.

Furthermore, we propose to take a closer look at the data formats and separate their metadata and data. Established concepts use mainly two data categories, *file system metadata*, such as ownership and time stamps, and *data* which is the file itself and all of the data inside. However, we propose to further separate the data inside the file into *file data* and *file metadata*.

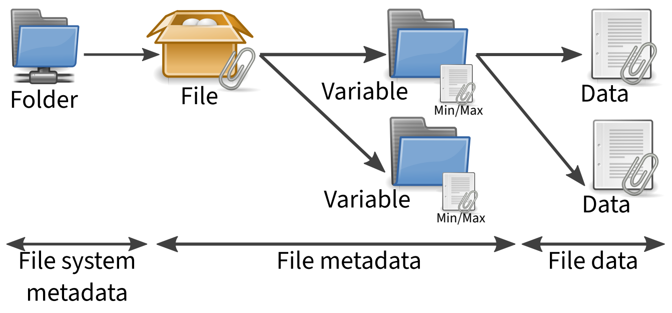

Figure 2: Different types of metadata and data for the BP format. Additional metadata such as the minimum and maximum values is stored for each variable.

Figure 2 shows this separation. A parallel file system manages files and folders and thereby file system metadata. The BP file contains multiple variables that in turn contain the file data. Also, there is additional file metadata, such as minimum and maximum values, which are managed by ADIOS2.

Currently, the file metadata is stored within the file on data servers. There, it is treated as data and can, therefore, only be accessed using optimizations for data retrieval such as large continuous reads. However, to make use of the information effectively, it has to be accessible as metadata, typically characterized by small and random accesses. We support accesses optimized for metadata by extracting it from the file and storing it in a database.

Using JULEA, we can store the file system metadata in key-value stores or databases, and the file data in object stores, thereby using established management solutions that proved to be most sensible for the two types of information.

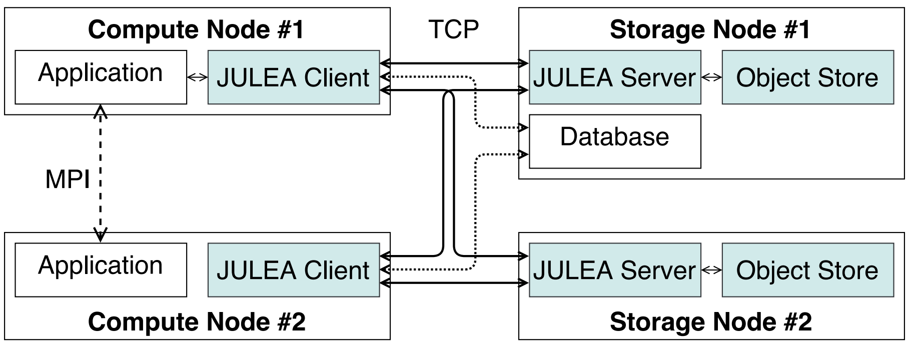

Figure 3: JULEA architecture with two distinct compute and storage nodes each.

Figure 3 shows the JULEA architecture. When using JULEA, the application interacts with one or several JULEA clients that communicate with the appropriate backends. Server processes on the appropriate nodes manage the backends. By offering a generic interface for every one of these concepts, the actual implementations can be easily exchanged. So it is possible, just by adapting the configuration, to switch the database backend, for example, from SQLite to MariaDB. Moreover, the backends can be deployed at different hardware levels, increasing the versatility of JULEA considerably.

1.3 Project Structure

To structure the project, it is split into work packages, which are further divided into tasks. Additionally, deliverables cover and subsume the generated content.

1.3.1 WP1: Application Interface

The highest level of CoSEMoS are application-facing interfaces. For backward compatibility, we will focus on existing SDDF interfaces such as ADIOS2 and HDF5. These interfaces are widely used in several scientific fields (for example, climate science and high-energy physics) and will, therefore, allow us to support a wide range of existing applications.

T1.1 **SDDF Interface**
Within this task, the ADIOS2 interface and existing applications will be analyzed to be able to quickly provide a minimal subset of the interface for running existing applications on top of it. The implementation will happen in the form of an ADIOS2 engine which executes the heavy I/O operations such as read and write accesses. An engine represents an exchangeable module with a clearly defined interface that provides support for ADIOS2's low-level read and write functionality.

T1.2 **Application Requirements and Semantics**
To be able to select the most appropriate database backend or storage technology, it is important to understand application and performance requirements. Valuable tunables will be collected and defined in collaboration with members of the scientific community and our partners. While JULEA contains general tunables useful for parallel I/O, a close collaboration with experts will give us insight into requirements that are specific to applications and data formats.

T1.3 **Data Analysis Interface**
To enhance the efficient and comfortable analysis of the data, an additional proof-of-concept data analysis interface will be realized. Tools such as CDO, CMOR (Climate Model Output Rewriter) or the WDCC's (World Data Center for Climate) CERA database will be able to make use of the performance improvements provided by the respective backends for the different metadata types. These tools currently have to extract the metadata from the SDDF files and optionally store them in a database for fast query performance. In addition to improved efficiency, this will also eliminate redundancies and inconsistencies caused by storing the extracted information in separate databases.

1.3.2 WP2: Storage Tier Selector and Global Metadata Manager

This work package hast two major targets: First, management of *structured metadata* to accelerate access for state-of-the-art applications and provide entirely new possibilities for data management. Second, placement of *data and metadata* to improve access performance and reduce costs. To this end, a new type of backend and client for structured metadata will be designed and implemented as part of this work package. Moreover, CoSEMoS will automatically determine the storage backend best suited for particular types of data and metadata using the Global Metadata Manager and the Storage Tier Selector.

T2.1 **Structured Metadata Backend**
JULEA already contains backends for object and key-value storage. However, support for structured metadata will enable new data management approaches using a query interface that is able to access both file system and file metadata. Defining schemas will also allow CoSEMoS to make use of indexes to speed up access for complex queries. Several of JULEA's key-value backends already use database systems with support for structured metadata but do not make use of their full potential.

T2.2 **Structured Metadata Client**
In addition to the backend for structured metadata, a new client will provide user-facing functionality to define schemas as well as all necessary data operations. It will be based on the existing key-value client and can thus make use of the existing implementation. The client's main purpose is to abstract the communication with the structured metadata backend and to provide developers with a convenient and efficient interface. While the client's first user will be the ADIOS2 engine designed and implemented in T1.1, the client will also be used by other JULEA clients.

T2.3 **Metadata Backend Selection**
The Global Metadata Manager will be a core component of JULEA and handle the selection of appropriate backends depending on access and I/O requirements by coordinating with the Storage Tier Selector (T2.4). 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 Global Metadata Manager will 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.

T2.4 **Data Storage Tiering**
The Storage Tier Selector will enable CoSEMoS to support and make use of multiple storage tiers such as node-local SSDs and burst buffers. Therefore, using these tiers efficiently requires appropriate policies to be defined in coordination with the Global Metadata Manager (T2.3). Similar to the metadata backends, JULEA currently only supports a single active data backend at the moment. Support for loading multiple data backends as well as the selection and routing logic will be implemented as part of this task.

1.3.3 WP3: Evaluation and Dissemination

This work package is concerned with evaluating CoSEMoS's compatibility and performance with selected existing applications and benchmarks. Additionally, as community involvement is a major factor when introducing or modifying interfaces, it will explicitly deal with gathering feedback and disseminating results.

T3.1 **Compatibility Tests**
CoSEMoS's interface will initially provide an ADIOS2 interface. To ensure proper compatibility with the existing API, we will run Continuous Integration to make sure that changes to CoSEMoS do not cause problems for applications and associated data processing tools. As CoSEMoS will depend on external software components that are in active development (specifically, ADIOS2), Continuous Integration will help us notice

potential problems with our dependencies' interfaces early. Additionally, we will define small test configurations for each application and use Continuous Integration to trigger runs on the Scientific Computing group's cluster.

T3.2 **Case Study**
To optimize CoSEMoS for real-world scenarios, we will perform a larger case study using an existing complex application. We will focus on applications already using ADIOS or the upcoming ADIOS2, such as NASA's GEOS-5. The concrete application to use will be determined in the course of the project. Access to additional climate applications will be possible through DKRZ.

T3.3 **Workshop Organization**
To ensure that CoSEMoS's design meets the requirements of the targeted communities using self-describing data formats, we will organize workshops in M9 and M27. The first workshop will be used to present the initial design and gather feedback. Specifically, we will show the SDDF interfaces and their optional extensions (T1.1). The second workshop will give an overview of CoSEMoS's design and implementation; we will focus on performance and management improvements enabled by CoSEMoS's novel architecture.

1.3.4 Partners

We will closely cooperate with the following institutions and researchers:

- Prof. Dr. Thomas Ludwig is the director of the German Climate Computing Center (DKRZ) and a professor at Universität Hamburg.
- Johann Lombardi is a Principal Architect in the High Performance Data Division at Intel. He leads the effort at Intel to develop a storage stack for Exascale HPC and Data Analytics.
- Uwe Schulzweida from Max Planck Institute for Meteorology is one of the main developers of CDO (Climate Data Operators).

2 Architecture

In the following, the design approach, as well as the implementation details, are specified. The core component is the ADIOS2 engine which handles the separation of the file data and file metadata. Figure 4 shows a typical JULEA configuration we use. The application is distributed across two compute nodes using MPI. It uses the ADIOS2 library for its I/O, internally employing our engine, which in turn uses the JULEA client to forward the metadata and data to different backends. 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.

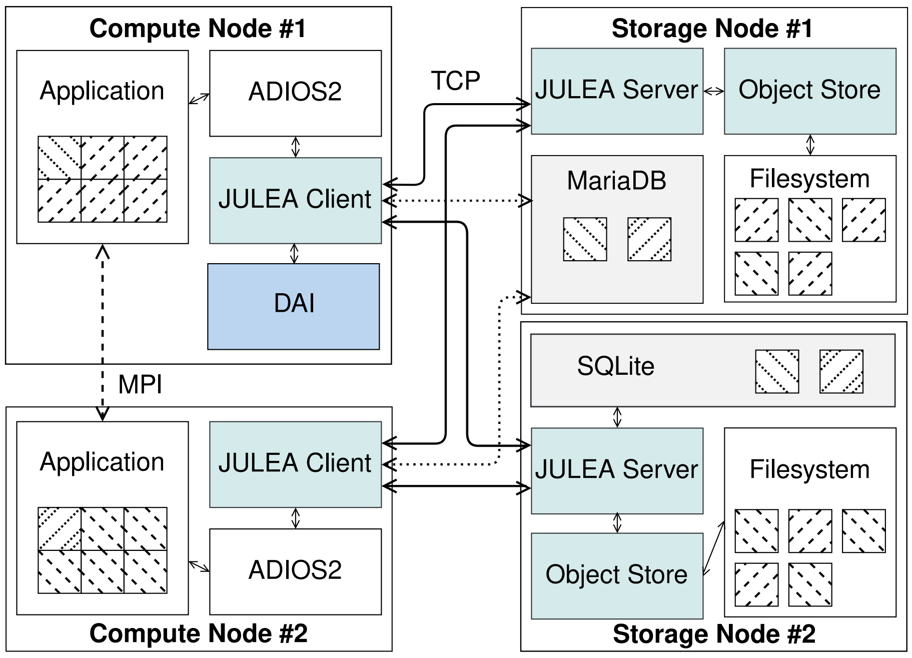

Figure 4: CoSEMoS setup with two compute and two storage nodes. The JULEA components are coloured in blue-green. The application uses an SDDF interface (T1.1), here ADIOS2, which in turn uses the JULEA client (T2.2). Depicted in gray are two database backends that can be interchanged but will not be used both at the same time. Marked in blue is the data analysis interface (DAI, T1.3) which can query file metadata and data directly on top of JULEA without using ADIOS2's interface.

2.1 SDDF Interface (T1.1)

To understand where and how the separation of file metadata and data is performed in our implementation, an overview of the ADIOS2 architecture is required.

2.1.1 ADIOS2 Internal Structure

In Figure 5, the core components are depicted. The initialization returns an ADIOS2 object, which administers the interaction between an application and the library. The management of the application-specific parameters is monitored and fine-tuned by the IO object that functions as a central control point for user-defined settings [Oak Ridge National Laboratory, 2018]. Furthermore, the IO object handles the variable and attribute definitions and keeps track of them. The used Engine determines the actual writing and reading behavior. When opening a file at the IO object, an Engine object of the set engine is generated and returned. It will perform the data output and input and communicate with the system's I/O resources. Whether an engine serves as a writer or a reader is defined by the file open mode. The engine component is built such that it also constitutes the point of application to implement new I/O behavior.

2.1.2 ADIOS2 Interface

Applications typically interact with ADIOS2 based on the following rough pattern. First, the application sets the engine to use for either writing or reading. Then, the file is opened in the

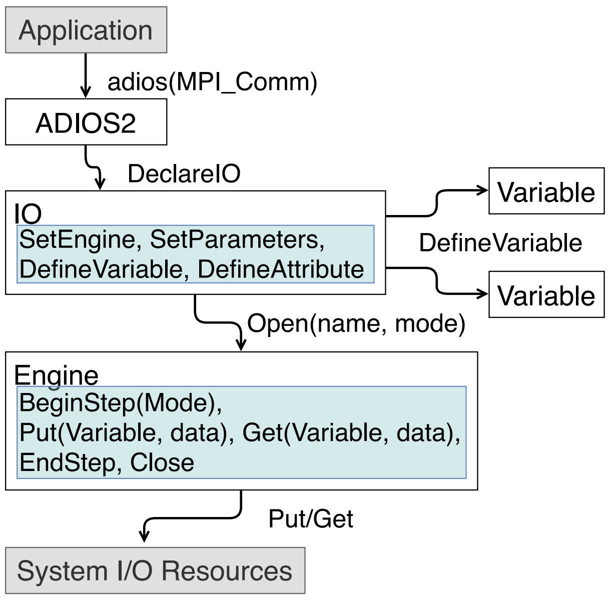
Application
adios(MPI_Comm)
ADIOS2
DeclareIO
IO
SetEngine, SetParameters, DefineVariable, DefineAttribute
DefineVariable
Variable
Variable
Open(name, mode)
Engine
BeginStep(Mode), Put(Variable, data), Get(Variable, data), EndStep, Close
Put/Get
System I/O Resources

Figure 5: ADIOS2 component overview based on [Oak Ridge National Laboratory, 2018]

corresponding I/O mode. Next, one or more variables are defined or inquired depending on whether it is preparation for writing or reading data. Applications typically perform their I/O in steps which can be anything like logical steps or time steps. ADIOS2 provides the option to specify the data to work on depending on the used variable concept, for example, local arrays in contrast to global arrays. Then, the actual I/O operation is performed using a write or read operation. Finally, after all the variables are handled the step is finished.

2.1.3 ADIOS2 Engines

We implemented two ADIOS2 engines, one that uses JULEA's key-value backend and one that uses the database backend. Both of them work with applications parallelized with MPI.

Figure 4 shows how the application output is split into file metadata and data and then stored separately. That means that the BP file is not stored as a file anywhere in JULEA. We allow fine-grained access to the ADIOS2 data blocks by using unique IDs to reference them. Therefore, migrating the database or the object store is uncomplicated as it does not require path updates. The actual client and database implementation can be exchanged easily.

Table 1 shows the exact metadata that is stored. It is a shortened list of features that the ADIOS2 variables support. The *BlocksArray* stores the number of steps and the *BlockIDs*. There are several ways that a variable can be read depending on its variable shapes, the selection type and the reading mode. The vector's shape, start and count define the variable dimensions in local and in the global space. *Shape* describes the physical dimension, *Start* the local offsets and *Count* the local size. With these parameters, five different variable shapes are available, namely, a global single value, a global array, a local value, a local array and in future ADIOS2 versions also a joined array. We implemented all four of them.

In ADIOS2, the metadata for a variable includes the global minimum and maximum while each block saves the local ones. Single values are directly stored in the variable, which we realized

Feature	Type	V/B
File Name	String	V
Variable Name	String	V
BlocksArray	Vector	V
ShapeID	enum	V
Type	String	V
SelectionType	Vector	V, B
BlockID	size_t	V, B
Shape	Vector	V, B
Start	Vector	V, B
Count	Vector	V, B
MemoryStart	Vector	V, B
MemoryCount	Vector	V, B
Min	T	V, B
Max	T	V, B
Value	T	V, B
Mean	T	V, B
isValue	Boolean	V, B
ConstantDims	Boolean	V, B
ReadAsJoined	Boolean	V, B
ReadAsLocalValue	Boolean	V, B
RandomAccess	Boolean	V, B

Table 1: Variable and block metadata currently stored JULEA (V denotes variable features and B denotes block features).

by storing it in the database. This approach is used because the overhead of retrieving a single value from the object store far outweighs the costs of storing it directly in the database.

To speed up querying the database builds an index on the file name, the variable name, the step, the block and the unique ID as well as minimum, maximum and mean value. The used database backend determines the actual index concept. We are planning to make this support more flexible in the future via our data analysis interface (DAI) which is highlighted in blue in Figure 4. Users will be able to specify values or expressions that should be calculated and stored in the database for faster post-processing.

Even though we store additional information, we make sure that we still support export to the BP formats. That means that we had to conform to the BP file specifics to make the changes transparent to the application layer. When exporting data to BP files, any custom metadata will be lost.

2.1.4 HDF5 Interface

We also support HDF5 applications through different VOL plugins that behave similar to the ADIOS2 engines.

HDF5 offers the The Virtual Object Layer (VOL) plugin interface that can be used to adapt its I/O behavior by enabling fine-grained control over the management of HDF5's different data structures. Specifically, it is possible to implement functions for creating, opening, reading, writing, deleting and closing attributes, datasets, datatypes, files, groups, links and objects. This allows separating file data and file metadata in a way that structural information can be used by the storage system for intelligent decisions. Additionally, file data and file metadata can be handled appropriately by storing file data in an object store, while file metadata is stored in a database.

Our HDF5 client uses the Virtual Object Layer. By using the VOL, it is possible to gain fine-grained access to file data and file metadata while still conforming to the HDF5 standard. This allows us to keep the native SDDF format. Therefore, this approach is transparent to existing applications because we do not introduce a separate generic storage format for self-describing data. HDF5's VOL allows defining functions for attributes, datasets, datatypes, files, groups, links and objects. For our implementation, we have focused on the most important areas of files, groups, datasets and attributes for now. This allows covering a wide range of HDF5 functionality without having to implement very specific niche features.

More information is available in [Kuhn and Duwe, 2020].

2.2 Database Client (T2.2)

We changed the client name from *Structured Metadata Client* to *Database Client*. The most important operations for the client are the schema definition, insertion, update and removal of data. The following structures constitute the core of the client. They are detailed in the works of Straßberger and Warnke [Straßberger, 2019, Warnke, 2019]:

- `JDBSchema`, `JDBSelector`, `JDBEntry`, `JDBIterator`

All of them provide the following functions: `new`, `delete`, `ref` and `unref`. The specific functions for each structure are mentioned in the following.

JDBSchema

- `add_field`, `get_field`, `get_all_fields`, `add_index`, `equals`, `create`, `get`

JDBSelector

- `add_field`, `add_selector`

JDBEntry

- `set_field`, `insert`, `update`, `get_id`

JDBIterator

- `next`, `get_field`

2.3 Metadata Backend Selection and Data Storage Tiering (T2.3, T2.4)

Since writing the proposal, we adapted our approach and the design of the Global Metadata Manager (GMM) and the Storage Tier Selector (STS). These changes are attributed to our work in the last year. Currently, neither the GMM nor the STS exist as a separate layer. 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. First work is underway to evaluate the behaviour of ENZO[1] when using the HDF5 VOL plugins for JULEA. This will allow us to determine useful metrics for the metadata backend selection. Furthermore, these logs will be used as a basis for the future storage tier selection.

Figure 6 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 is an obvious goal we will approach in the future. Also, the possibility to create multiple instances of a backend is realized.

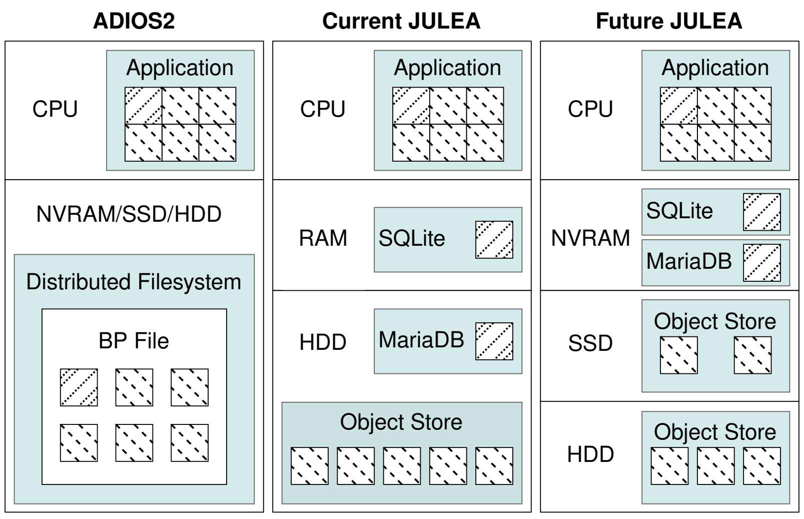

Figure 6: Different storage components on corresponding hardware layers: ADIOS2 writes its BP files into a parallel file system where they are stored as a byte stream across the storage devices the file system is configured to use. It is not easily possible to put different parts of the BP file onto 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).

[1] `https://enzo-project.org/`

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

2.3.1 Next Steps

Using the MH, 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 will 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.

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.

3 Workshop

We changed the approach for the first workshop due to the circumstances of this year. Moving the project from Hamburg to Magdeburg did not allow us to follow through as intended. For a long time, it was not clear when exactly the relocation of the project would take place, which hindered the organization of the workshop. Therefore, we decided to refrain from organizing a local workshop and to take the topic to the workshop program of an international conference. Together with storage experts from Sandia National Laboratory, University of Illinois at Urbana-Champaign and Intel Corporation, we submitted a workshop proposal to Euro-Par 2020, which got accepted. CHAOSS (Challenges and Opportunities of HPC Storage Systems)[2] was aimed at researchers, developers of scientific applications, engineers and all those interested in the evolution of HPC storage systems in general.

The main objective of this workshop was to explore the intersection of self-describing data formats and other data organizations with multi-tier memory and storage hierarchies. The goal was to find the right balance for different workloads. The workshop was a venue for papers exploring topics related to data organization and management, along with the impacts of multi-tier memory and storage for optimizing application throughput.

Unfortunately, due to the COVID pandemic and the longlasting uncertainty whether the conference and the workshop would be able to take place, we could only advertise the workshop a few weeks before the submission deadline. That resulted in only two submissions, one of which had to be rejected. The CHAOSS workshop, therefore, was held together with the

[2] `https://wr.informatik.uni-hamburg.de/events/2020/chaoss`

ParaMo workshop[3] to reach a larger audience. The talks were prerecorded, while a live online discussion allowed interaction between the presenters and the audience. About 30 people attended the combined workshop.

Since then, another workshop proposal has been accepted for the EuroSys 2021. CHEOPS (Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems)[4] 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.

The insights that we hoped to get from organizing a workshop specifically addressing our project topics had to be gained differently. Through private discussions with one of the main developers behind the ADIOS2 concept, namely Jay Lofstead [Lofstead et al., 2008], we were able to validate the feasibility not only of the concept but also our actual implementation strategies. Furthermore, talks with developers from the Max Planck Institute for Meteorology[5] and the German Climate Computing Center[6] helped to clear requirements from the application side. Through online discussions with application developers, we were able to get large-scale simulations running that use ADIOS2 for their I/O.

[3] `https://2020.euro-par.org/program/workshops/`

[4] `https://cheops-workshop.github.io/`

[5] `https://mpimet.mpg.de/`

[6] `https://www.dkrz.de/`

References

[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.

[Kuhn and Duwe, 2020] Kuhn, M. and Duwe, K. (2020). Coupling Storage Systems and Self-Describing Data Formats for Global Metadata Management. In *International Conference on Computational Science and Computational Intelligence (CSCI 2020)*. Conference Publishing Services (CPS). To be published.

[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.

[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.

[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`.

[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`.