Dissecting Self-Describing Data Formats to Enable Advanced Querying of File Metadata

Kira Duwe
Otto von Guericke University Magdeburg
Magdeburg, Saxony-Anhalt
kira.duwe@ovgu.de

Michael Kuhn
Otto von Guericke University Magdeburg
Magdeburg, Saxony-Anhalt
michael.kuhn@ovgu.de

ABSTRACT

In times of continuously growing data sizes, performing insightful analysis is increasingly difficult. I/O libraries such as NetCDF and ADIOS2 offer options to manage additional metadata to make the data retrieval more efficient. 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. Splitting the BP file content 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 a custom ADIOS2 engine. Moreover, our data analysis interface allows speeding up metadata queries by a factor of up to 60,000 in comparison to the ADIOS2 API and data formats.

CCS CONCEPTS

• **Information systems** → **Distributed storage**; **Hierarchical storage management**; • **Computer systems organization** → **Client-server architectures**.

KEYWORDS

Metadata management, Self-describing data formats, ADIOS2, JULEA, Metadata querying

ACM Reference Format:
Kira Duwe and Michael Kuhn. 2021. Dissecting Self-Describing Data Formats to Enable Advanced Querying of File Metadata. In *The 14th ACM International Systems and Storage Conference (SYSTOR '21), June 14–16, 2021, Haifa, Israel.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3456727.3463778

SYSTOR '21, June 14–16, 2021, Haifa, Israel
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8398-1/21/06.
https://doi.org/10.1145/3456727.3463778

1 INTRODUCTION

Large-scale simulations and high-volume streaming systems generate rapidly growing data sets that are increasingly difficult to manage and find insights in. For these growing data sets, it is vital to sift through the data volumes most efficiently [5]. However, the hardware hierarchy of high-performance computing (HPC) systems complicates the data analysis. Another factor increasing the complexity is the software stack on top. The stack splits into several mostly isolated layers. While this allows exchanging them, the isolation leads to performance and management issues; e.g., how and where to optimize the data access. Furthermore, the stack is built on the outdated POSIX (Portable Operating System Interface) I/O interface.

To make filtering raw data easier, I/O libraries such as NetCDF (Network Common Data Form), HDF5 (Hierarchical Data Format), and ADIOS (Adaptable IO System)[18], and the according *self-describing data formats (SDDFs)* are used. The goal is to provide self-explanatory data that can be exchanged between researchers easily while enabling sifting features to cope with the large data sizes. This is accomplished by the options to annotate the data with information about the experiment. All of these formats support the notion of variables and attributes. While HDF5 focuses on hierarchical structuring, ADIOS2 aims for a very flat namespace. Though differing in the exact metadata associated with a variable, the variable concepts can be mapped onto or transformed into one another.

1.1 Separating file metadata

Parallel file systems manage files and folders and the resulting *file system metadata.* Its distinction to *file content* is long established. However, we propose to also split the file content into *file metadata* and *file data.* File metadata is the core feature of an SDDF. It is the structural information about the data as well as annotations and additional statistical metadata like minima.

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. Therefore, we dissect the file format and store the file metadata in a database and the data in an object store.

In climate research, simulations can reach several petabytes (PB) for large ensemble runs that need to be stored for decades to validate the models [19]. For cost and maintenance reasons, the data is archived in tape storage. The ADIOS2 formats BP3 and BP4 store the minima and maxima of every step and block for all variables. Still, to locate all variables in a specific value range, all files have to be read. By storing the file metadata separately, only accesses to the variables meeting the query expression are required instead of scanning the complete data. This is where we see great untapped potential.

1.2 Contribution

In prior work, we showed that storing the separated file metadata of HDF5 in a key-value store is beneficial even for trivial tasks as iterating over all attributes [15]. In this paper, we show this separation also works for ADIOS2 formats and we extend this idea to enable complex queries by using a database instead. The database allows us to work directly with the data, without having to serialize and deserialize the values every time they are written or read. Furthermore, we enhance the statistics ADIOS2 manages, like minima/maxima, and show the benefits of precomputing and storing values such as the average in the database as well.

Summary of our contribution: 1). We developed a design for how to split up BP3/BP4 to enable querying on file metadata without data duplication. 2). We implemented this separation inside the ADIOS2 library by building an engine that stores the file metadata in a database in JULEA and the file data in an object store. Our engine also stores the mean values for a variable, highlighting the flexibility of our concept over static file formats. 3). We evaluated our engine in comparison to BP3 and BP4 for parallel and distributed writing and reading. We showed the performance benefits for querying in a post-processing scenario can be as high as a factor of 60,000. 4). All code is open source[1].

2 ADIOS2 & JULEA

The ADIOS2 data model is focused on variables and attributes, completely discarding any other object types for hierarchical structuring such as groups or links, as found in, e.g., HDF5 [9]. The data is arranged in steps which are subdivided into blocks. In contrast to BP3, BP4 uses an index table to keep track of the chunked metadata and is reported to maintain a relatively constant overhead with an increasing step number [6]. By introducing the concept of data characteristics, the BP formats can capture additional statistics as the value range.

The actual writing and reading behavior of ADIOS2 is determined by the used engine. It will perform the data output and input and communicate with the system's I/O resources. The engine component is built such that it constitutes the point of application to implement new I/O behavior. We realized the format dissection by implementing our own ADIOS2 engine.

JULEA. To realize the file metadata separation, we needed to exchange the parallel file system below with a storage design consisting of dedicated backends using integrated databases, object stores, and key-value stores. Had we continued to use Lustre, the database could only be used on top of the file system instead of replacing it. We chose JULEA, which is a storage framework that provides building blocks for a custom storage system [14]. JULEA runs completely in userspace, thereby avoiding the need for root administrative privilege and easing debugging and development.

Figure 1 shows the JULEA architecture. When using JULEA, the application interacts with one or several clients that communicate with the backend managed by server processes on the storage nodes. As the interfaces are generic, the actual implementations can be exchanged, e.g., from SQLite to MariaDB. Currently, the object store is a prototype built to store the data directly in POSIX, which brings along the expected limitations discussed in the introduction. However, it is a sufficient basis to evaluate the format dissection in general. Also, the network communication is based on TCP/IP, considerably increasing the overhead.

3 DESIGN AND IMPLEMENTATION

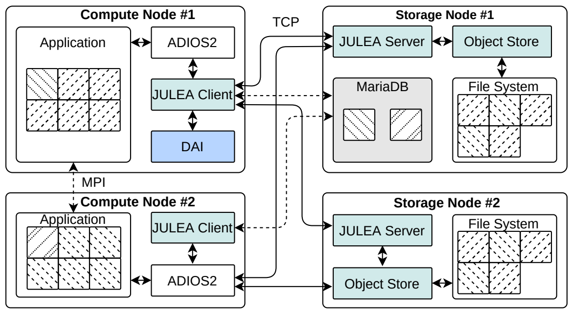

Figure 1: Benchmark setup with two compute and storage nodes.

The core component of our work is the design and implementation of an ADIOS2 engine that handles the separation of the file data and file metadata. Figure 1 shows one of the

[1]https://github.com/julea-io/adios2

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 2. Nevertheless, we will examine the possibilities for a distributed database in the future.

Data Distribution. Figure 1 also shows how the application output is split into file metadata and data and then stored separately. Currently, 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 evaluate SQLite and MariaDB as the database backends.

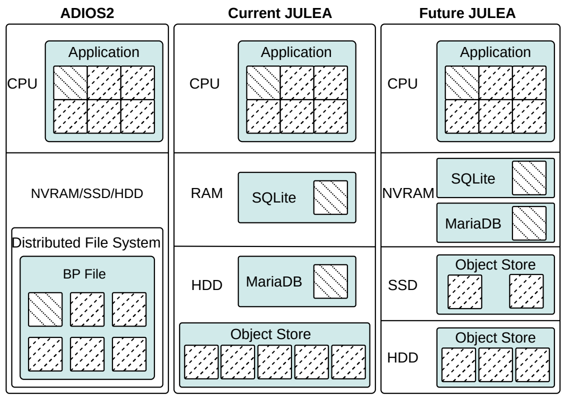

Figure 2: Different storage components on corresponding hardware layers

The separation of file metadata and data allows us to use JULEA's versatile configuration features as depicted in Figure 2. So, the database can be stored on faster hardware than the object store without the need for large amounts of fast hardware. This is possible because the database does not hold the data and therefore is small compared to the original file size.

JULEA-DB engine. We have implemented an ADIOS2 engine (JULEA-DB) to realize the format dissection. 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 ADIOS2 but 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. Currently, our data analysis interface (DAI) (see Figure 1) is a rudimentary prototype. This custom metadata can only be accessed through the JULEA client and will be lost when exporting data to BP files.

4 EVALUATION

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.

4.1 Write and Read Performance

To evaluate the write and read performance, we used the heat transfer application from the ADIOS2 examples [16]. 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, i.e. 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.

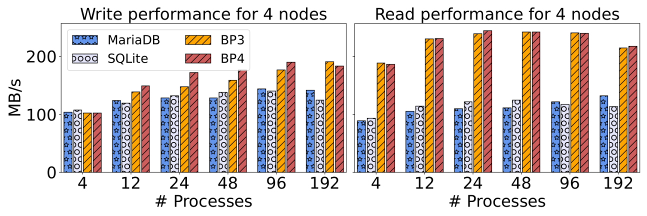

Figure 3: Write and read performance for 4 nodes with 1 to 48 processes per node and matrix size of 4096^2.

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.

4.2 Query Time

The main focus of this paper 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.

To validate the usefulness of the data analysis interface (DAI) depicted in Figure 1 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 the previous figures. 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 60000 when using our engine with the DAI layer.

5 RELATED WORK

To improve the metadata management, large-scale research projects often develop domain or even system specific solutions that are therefore not widely applicable [2, 7, 8, 10, 20, 24]. Other popular attempts to improve efficient data querying and retrieval focus on marking relevant regions of the data with labels [22, 23]. *EMPRESS2* is another approach for using labels to improve the metadata management [17]. It offers custom metadata tags enabling users to highlight interesting areas of data before storing. Thereby, the post-processing can be eased, as scanning the complete data is no longer required. It outperforms established self-describing formats, such as HDF5, in metadata querying considerably. Lawson et al. show the value of a relational database management system (RDBMS) for the management of scientific

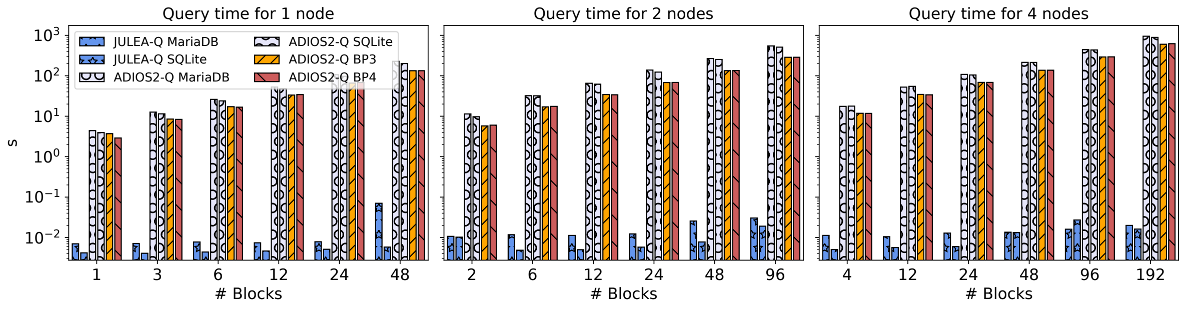

Figure 4: Query time for ADIOS2-Query and JULEA-Query for 1, 2 and 4 nodes. The number of blocks equals the number of processes for writing and reading. The matrix size is 4096^2. Note the log scale for the time (y-axis).

metadata in HPC. As Zhang et al. emphasize, having to maintain a separate database introduces several problems [3, 27]. Besides EMPRESS they discuss other projects using RDBMSs for metadata management like *BIMM* [13] and *SPOT* [25]. An approach to circumvent the required transformation of metadata is *JAMO*, which uses the document database MongoDB, yet still shares the same drawbacks as the RDBMS-based proposals. Most importantly, using a database typically leads to duplication of metadata. By replacing the file system with JULEA, we can employ an integrated database and thereby avoid metadata duplication. Also, this removes the challenge of keeping the data consistent across the files and the database. Zhang et al. bypass these challenges by introducing *MIQS (Metadata Indexing and Querying Service)* for HDF5. MIQS constitutes a schema-free solution that maintains an in-memory index for each process. The relation between attributes and file paths, however, needs to be stored redundantly. Finally, the ADIOS developers have evaluated different block indexing techniques to speed up the ADIOS metadata management. As of now (ADIOS2 version 2.6), the query interface [21] is not deployed in ADIOS2. The index concepts follow a similar line as to MIQS by building an index on a file allowing direct metadata access and faster data access [11, 26]., Though it improves the querying performance in general, the index processing time impacts the performance, as a large number of blocks slows down the used indexing mechanisms considerably.

6 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) [12].

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 is underway for an object store backend using Ceph's BlueStore [1] to avoid POSIX whenever possible [4]. Also, the network communication is based on TCP/IP, considerably increasing the overhead. It will be extended with libfabric support in the future. Based on our concept of dissecting SDDFs, we will use structural information in the form of file metadata for intelligent hierarchical storage management.

ACKNOWLEDGMENTS

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 417705296. We thank the reviewers and our shepherd for their constructive feedback and help.

REFERENCES

[1] Abutalib Aghayev, Sage A. Weil, Michael Kuchnik, Mark Nelson, Gregory R. Ganger, and George Amvrosiadis. 2019. File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, Tim Brecht and Carey Williamson (Eds.). ACM, 353–369. https://doi.org/10.1145/3341301.3359656

[2] Solveig Albrand, Thomas Doherty, Jerome Fulachier, and Fabian Lambert. 2008. The ATLAS metadata interface. In *Journal of Physics: Conference Series*, Vol. 119. IOP Publishing, 072003.

[3] Suren Byna, M. Scot Breitenfeld, Bin Dong, Quincey Koziol, Elena Pourmal, Dana Robinson, Jérome Soumagne, Houjun Tang, Venkatram Vishwanath, and Richard Warren. 2020. ExaHDF5: Delivering Efficient Parallel I/O on Exascale Computing Systems. *J. Comput. Sci. Technol.* 35, 1 (2020), 145–160. https://doi.org/10.1007/s11390-020-9822-9

[4] Kira Duwe and Michael Kuhn. 2021. Using Ceph's BlueStore as Object Storage in HPC Storage Framework. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems* (Online Event, United Kingdom) *(CHEOPS '21)*. Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/3439839.3458734

[5] Kira Duwe, Jakob Lüttgau, Georgiana Mania, Jannek Squar, Anna Fuchs, Michael Kuhn, Eugen Betke, and Thomas Ludwig. 2020. State of the Art and Future Trends in Data Reduction for High-Performance Computing. *Supercomput. Front. Innov.* 7, 1 (2020), 4–36. https://doi.org/10.14529/jsfi200101

[6] Michael Feldman. 2019. Will HPC Centers Say ADIOS To POSIX I/O? https://www.nextplatform.com/2019/09/04/will-hpc-centers-say-adios-to-posix-i-o/. Accessed: 2020-05-18.

[7] Jerome Fulachier, O Aidel, S Albrand, F Lambert, Atlas Collaboration, et al. 2014. Looking back on 10 years of the ATLAS Metadata Interface. Reflections on architecture, code design and development methods. In *Journal of Physics: Conference Series*, Vol. 513. IOP Publishing, 042019.

[8] Jerome Fulachier, Fabian Lambert, and Jerome Odier. 2017. ATLAS Metadata Interface (AMI), a generic metadata framework. In *J. Phys. Conf. Ser.*, Vol. 898. 062001.

[9] William F. Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip E. Davis, Jong Choi, Kai Germaschewski, Kevin A. Huck, Axel Huebl, Mark Kim, James Kress, Tahsin M. Kurç, Qing Liu, Jeremy Logan, Kshitij Mehta, George Ostrouchov, Manish Parashar, Franz Poeschel, David Pugmire, Eric Suchyta, Keichi Takahashi, Nick Thompson, Seiji Tsutsumi, Lipeng Wan, Matthew Wolf, Kesheng Wu, and Scott Klasky. 2020. ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management. *SoftwareX* 12 (2020), 100561. https://doi.org/10.1016/j.softx.2020.100561

[10] Matthias Grawinkel, Lars Nagel, Markus Mäsker, Federico Padua, André Brinkmann, and Lennart Sorth. 2015. Analysis of the ECMWF Storage Landscape. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*, Jiri Schindler and Erez Zadok (Eds.). USENIX Association, 15–27. https://www.usenix.org/conference/fast15/technical-sessions/presentation/grawinkel

[11] Junmin Gu, Scott Klasky, Norbert Podhorszki, Ji Qiang, and Kesheng Wu. 2018. Querying Large Scientific Data Sets with Adaptable IO System ADIOS. In *Supercomputing Frontiers - 4th Asian Conference, SCFA 2018, Singapore, March 26-29, 2018, Proceedings (Lecture Notes in Computer Science)*, Rio Yokota and Weigang Wu (Eds.), Vol. 10776. Springer, 51–69. https://doi.org/10.1007/978-3-319-69953-0_4

[12] Frank Kaspar, Uwe Schulzweida, and Ralf Mueller. 2010. "Climate data operators" as a user-friendly processing tool for CM SAF's satellite-derived climate monitoring products. https://doi.org/10.13140/RG.2.2.20422.68165

[13] Daniel Korenblum, Daniel Rubin, Sandy Napel, Cesar Rodriguez, and Chris Beaulieu. 2011. Managing biomedical image metadata for search and retrieval of similar images. *Journal of digital imaging* 24, 4 (2011), 739–748.

[14] Michael Kuhn. 2017. JULEA: A Flexible Storage Framework for HPC. 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 (Lecture Notes in Computer Science)*, Julian M. Kunkel, Rio Yokota, Michela Taufer, and John Shalf (Eds.), Vol. 10524. Springer, 712–723. https://doi.org/10.1007/978-3-319-67630-2_51

[15] Michael Kuhn and Kira Duwe. 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). In press.

[16] Oak Ridge National Laboratory. 2021. Heat Transfer Application ADIOS2. https://github.com/ornladios/ADIOS2/tree/master/examples/heatTransfer. Accessed: 2021-05-18.

[17] Margaret Lawson and Jay F. Lofstead. 2018. Using a Robust Metadata Management System to Accelerate Scientific Discovery at Extreme Scales. In *3rd IEEE/ACM International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems, PDSW-DISCS@SC 2018, Dallas, TX, USA, November 12, 2018*. IEEE, 13–23. https://doi.org/10.1109/PDSW-DISCS.2018.00004

[18] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 2008. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE@HPDC 2008, Boston, MA, USA, June 23, 2008*, Yoonhee Kim and Xiaolin Li (Eds.). ACM, 15–24. https://doi.org/10.1145/1383529.1383533

[19] Jakob Lüttgau, Michael Kuhn, Kira Duwe, Yevhen Alforov, Eugen Betke, Julian M. Kunkel, and Thomas Ludwig. 2018. Survey of Storage Systems for High-Performance Computing. *Supercomput. Front. Innov.* 5, 1 (2018), 31–58. https://doi.org/10.14529/jsfi180103

[20] Sangmi Lee Pallickara, Shrideep Pallickara, and Milija Zupanski. 2012. Towards efficient data search and subsetting of large-scale atmospheric datasets. *Future Generation Comp. Syst.* 28, 1 (2012), 112–118. https://doi.org/10.1016/j.future.2011.05.010

[21] Norbert Podhorszki, Qing Liu, Jeremy Logan, Jingqing Mu, Hasan Abbasi, Jong-Youl Choi, and Scott A. Klasky. 2018. ADIOS 1.13.1 USER'S MANUAL. https://users.nccs.gov/~pnorbert/ADIOS-UsersManual-1.13.1.pdf. Accessed: 2020-05-15.

[22] Hyogi Sim, Youngjae Kim, Sudharshan S. Vazhkudai, Geoffroy R. Vallée, Seung-Hwan Lim, and Ali Raza Butt. 2017. Tagit: an integrated indexing and search service for file systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, November 12 - 17, 2017*, Bernd Mohr and Padma Raghavan (Eds.). ACM, 5:1–5:12. https://doi.org/10.1145/3126908.3126929

[23] Houjun Tang, Suren Byna, Bin Dong, Jialin Liu, and Quincey Koziol. 2017. SoMeta: Scalable Object-Centric Metadata Management for High Performance Computing. In *2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017*. IEEE Computer Society, 359–369. https://doi.org/10.1109/CLUSTER.2017.53

[24] Ani R Thakar, Alex Szalay, George Fekete, and Jim Gray. 2008. The catalog archive server database management system. *Computing in Science & Engineering* 10, 1 (2008), 30.

[25] Craig E. Tull, Abdelilah Essiari, Dan Gunter, Xiaoye Sherry Li, Simon J. Patton, and Lavanya Ramakrishnan. 2013. The SPOT Suite project. http://spot.nersc.gov/.. Accessed: 2020-10-09.

[26] Tzu-Hsien Wu, Jerry Chi-Yuan Chou, Norbert Podhorszki, Junmin Gu, Yuan Tian, Scott Klasky, and Kesheng Wu. 2017. Apply Block Index Technique to Scientific Data Analysis and I/O Systems. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, Madrid, Spain, May 14-17, 2017*. IEEE Computer Society / ACM, 865–871. https://doi.org/10.1109/CCGRID.2017.37

[27] Wei Zhang, Suren Byna, Houjun Tang, Brody Williams, and Yong Chen. 2019. MIQS: metadata indexing and querying service for self-describing file formats. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019*, Michela Taufer, Pavan Balaji, and Antonio J. Peña (Eds.). ACM, 5:1–5:24. https://doi.org/10.1145/3295500.3356146