Optimizing Metadata Exchange: Leveraging DAOS in the Realm of ADIOS

***Abstract*—In HPC I/O middleware like the Adaptable I/O System (ADIOS) often mediates data transfers between applications. The metadata I/O generated by such systems often presents significant scaling and performance limitations. This work seeks improvement opportunities for metadata I/O by leveraging the DAOS storage systems, a recent storage system solution deployed on high-end systems such as the Aurora supercomputer. We investigate the tradeoffs and the design space for integrating I/O engines for the ADIOS middleware based on the different storage mechanisms supported by DAOS. We present a new DAOS-Array-ChunkSize-aligned engine which provides up to 2.3× improved performance than when using the existing DAOS-POSIX interface, without requiring any applications modifications.**

I. Introduction

With the exponential increase in data volume in HPC, I/O overheads have emerged as a critical bottleneck. Several endeavors have been made to mitigate this, aiming both to enhance I/O bandwidth and foster higher concurrency [1], [2]. Nevertheless, the role of metadata management in HPC, often underemphasized, is vital [3]. The scale and intricacy of applications have simultaneously elevated both the volume and complexity of metadata. Recent research indicates that metadata can profoundly influence the entire HPC I/O efficiency [4]. Importantly, merely scaling up hardware resources is not a panacea for the metadata conundrum. We foresee this pattern continuing, especially as we transition into the exascale epoch, marked by the rise of machines like the Aurora supercomputer [5].

To address more generally I/O bottlenecks in HPC and datacenter systems, and to better leverage the performance capabilities of emergine storage devices, the storage computing recently introduced a software storage stack, Intel DAOS (Distributed Asynchronous Object Storage) [6]. DAOS has been designed to simplify the integration of software stacks with new memory and storage technologies, including persistent memory (PMEM) and NVMe devices [7]. DAOS minimizes software overhead when interfacing with PMEM, thus capitalizing on the performance benefits these technologies offer over conventional storage devices like HDD/SSD [8]. The system endorses zero-copy and asynchronous I/O operations. Fundamentally, DAOS introduces an object interface that inherently supports key-value and array formats. Moreover, to ensure compatibility with pre-existing systems, it incorporates a POSIX emulation layered atop its native DAOS array object interface.

While substantial emphasis has been placed on delivering the benefits of DAOS for HPC I/O [9], less attention has been placed on the unique requirements of HPC metadata I/O. "Metadata" is a term with specific meanings at virtually every layer of a software stack, but in this context we are specifically talking about metadata produced by I/O systems such as ADIOS[10], MPI-IO, HDF5 and NetCDF which support the execution of complex HPC workflows. In the context of these I/O middleware systems, metadata is produced during the course of writing specific data that is later to be used by the downstream workflow components (i.e., readers) to locate that data. Given the growing performance challenges related to metadata I/O, in this work we focus on investigating the implications of using DAOS for metadata I/O.

Specifically, we focus on the ADIOS high-performance I/O system. ADIOS is widely used at Oak Ridge National Laboratory and several other national labs for its ability to efficiently manage large-scale data, especially in environments where performance and scalability are critical. Its flexible framework and support for various data formats make it an essential tool in advancing complex scientific research, including real-time data processing and large-scale simulations in domains like climate modeling and astrophysics. Most recently, ADIOS was the driving force behind the I/O stack in the Gordon Bell Prize-winning(2023) E3SM, a climate simulation application [11].

In ADIOS, each writer rank produces metadata that is archived in stable storage. In order to fulfill ADIOS reader-side semantics generally every reader rank must have access to *all* the metadata produced by each writer rank. Thus the access pattern for metadata tends to be different than that of the application data that it describes, because in a multi-rank application each reader tends to access only a subset of the whole data, while requiring access to the whole of the metadata in order to locate that data. This further raises the need for metadata-specific study on the performance implications of new storage technologies, with DAOS being the focus of our work. Prior implementations of ADIOS metadata storage and access employ POSIX files and are bound by the constraints inherent to POSIX design. Even though DAOS is equipped with key-value and array object interfaces, and has showcased its efficacy in recent performance evaluations [9], discerning the optimal strategy to employ these interfaces concerning ADIOS metadata transfer remains a complex challenge.

This paper presents the following significant contributions:

- In section II, we demonstrate the difficulties of ADIOS metadata handling, structured around POSIX semantics, and their substantial impact on the metadata end-to-end transfer time in large-scale settings with the E3SM application.

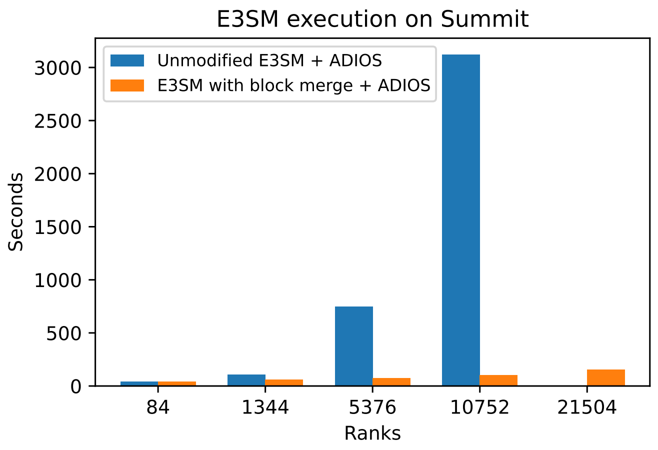


Fig. 1: Writing data from every rank in unmodified E3SM vs. a modified E3SM which pre-aggregated data to write from fewer ranks. The performance differences were traced to overheads in handling ADIOS-level metadata.

- In section IV, we present the complex design space in using DAOS Key-Value and Array objects for the transfer of ADIOS metadata.
- In section VI, we evaluate DAOS metadata engines in ADIOS to measure the impact of metadata size, number of ranks, and a large number of ADIOS timesteps. Further, we analyze the trade-offs in committing and acquiring metadata on end-to-end metadata transfer time.
- Our new DAOS array-based engine provides 2.3X faster ADIOS metadata end-to-end transfer time than POSIX at scale.

II. Motivation

As noted above, most prior work in HPC I/O has focused specifically on maximizing the rate at which large volumes of data can be moved to storage, while largely neglecting issues surrounding the metadata that systems like ADIOS produce in order to identify and provide access to individual data blocks. Generally metadata is presumed to be relatively small as compared with the data and therefore not of particular interest in HPC I/O.

In order to challenge this view we present the set of application timings shown in Figure 1, which depicts different runs of the Energy Exascale Earth System Model (E3SM). E3SM, a state-of-the-art, fully-integrated model of Earth's climate, encompassing key biogeochemical and cryospheric processes [12]. The figure depicts the performance curves for two slightly different implementations of E3SM. The blue bars represent a "original" ADIOS implementation of E3SM I/O, where each application rank performs an ADIOS Put() operation of the data in its possession. Note that the wallclock time for a 960 timestep run is rising exponentially as we increase the number of MPI ranks (weak scaling), to the point where run at 21504 ranks couldn't be completed. In comparision, the orange bars represent a modified version of E3SM in which that same application data is aggregated so that only 1/6 of ranks perform a Put(), but now with a block 6x larger (this is called "block merge" in the figure). *The principal difference between the original and the block merge versions of E3SM are in the amount of ADIOS-level metadata created and processed. The amount of actual data is essentially the same, yet the overall performance is significantly better solely because the amount of metadata has been reduced.* This supports the assertion that in situations where both complexity of the metadata and the number of writing ranks are large, middleware-level metadata handling *can* have a significant performance impact on HPC I/O. And while E3SM developers were able to sidestep scalability problems with application changes, an application shouldn't have to be rewritten to perform its own data aggregation simply to minimize middleware metadata overheads.

Performance graphs Figure 1 are often the result of focusing on data-to-storage bandwidth and ignoring metadata processing as inconsequential. The data in this figure dates from an early stage in this project and some of of the metadata overheads that caused the exponential overheads in the graph were due to ADIOS metadata handling practices that have since been changed. However one significant issue that remains is that in the ADIOS POSIX-based implementation, all writer metadata is gathered via MPI to a single rank and written into a POSIX file, an operation that isn't required by ADIOS semantics and which seemed ripe for reconsideration. Specifically, we hoped to use DAOS object interfaces to produce a performance curve for metadata-heavy applications like E3SM while employing ADIOS in the way it was designed, writing from each rank with data without requiring application-level changes and data aggregation to avoid metadata overheads.

III. Background

A. ADIOS

ADIOS [13] is a middleware library tailored for HPC I/O, offering applications a high-level data abstraction and concealing the intricacies of data transport/storage/retrieval between application memory and various HPC mediums like networks, files, wide-area-networks, or direct memory access channels. Its primary aim is to equip exascale applications with optimal storage/network bandwidth on premier HPC computing assets. Although ADIOS can facilitate online code-coupling (for instance, a direct link between running data sources and sinks), this research primarily delves into I/O directed to and from stable storage. In ADIOS, stable storage is used as an endpoint for large scientific campaigns for checkpointing and also for post-processing simulation data.

In ADIOS, the I/O abstraction adopts a collective and timestep-based method, where synchronization occurs separately within the groups of reader ranks and writer ranks. This is managed through the use of Begin/EndStep() calls within each group. Within every I/O timestep, applications have the ability to Put()/Get() data to or from storage, with each action targeting a predefined ADIOS *variable*. These

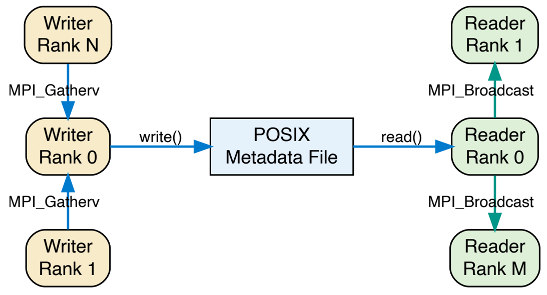


Fig. 2: ADIOS - POSIX engine

variables can represent various ADIOS data structures, ranging from individual named values to expansive multi-dimensional arrays that can be split across the ranks in an MPI-driven application.

ADIOS metadata. In ADIOS, the reader-side semantics facilitate data discovery. Consequently, whenever an ADIOS operation writes data, it concurrently generates metadata. This metadata encapsulates details such as the variable's name that has been outputted, its type, dimensionality, the virtual location (both start and count) within the ADIOS global array space, and the precise location and size of the data block in storage. Generally, to fulfill Get() requests in ADIOS, every reader rank should possess access to the complete metadata set produced by each writer rank for that specific timestep. This implies that while the actual data read patterns may vary based on the application's requirements, the metadata exchange typically follows a more uniform all-gather transfer mechanism.

Metadata Transfer in ADIOS. In the current ADIOS POSIX engine, shown in Figure 2, metadata created by all writer ranks is aggregated on rank 0 for each timestep, and then recorded to a specialized metadata file. On the reader's end, rank 0 acquires the writer's metadata and distributes it to other reader ranks using an MPI broadcast.

For evaluating the performance and design trade-offs in a DAOS-based metadata engine, we define metrics that track the movement of metadata from writers to readers. We use inclusive terminology since the "write" and "read" phases involve may non-I/O operations, such as `MPI_Gather()` and `MPI_Bcast()`.

- **Metadata Stabilization Time:** This duration encompasses all steps from the initial creation of metadata to the point where a writer has securely stored it in stable storage, confirming its durability and accessibility.
- **Metadata Acquisition Time:** This refers to the process of fetching the metadata from stable storage and making it available to all reader ranks, enabling their access and use of the data.
- **Metadata End-to-End Transfer Time:** The total of stabilization and acquisition times.

Given that our emphasis is on large-scale data, our analysis is primarily on the continual metadata transfer expense in each timestep, sidelining any singular costs.

B. DAOS

Distributed Asynchronous Object Storage (DAOS) is an open-source object store tailored for byte-addressable storage [7]. Designed to offer low latency and high bandwidth access to storage, DAOS caters to various use cases, including for data centers, traditional HPC, and AI/ML tasks. DAOS is designed to harness Storage Class Memory (SCM) and NVMe (Non-Volatile Memory Express) for improved performance and efficiency. Originally DAOS employed SCM for efficient handling of small, latency-sensitive I/O operations and metadata. Beyond this, NVMe drives supplement with added capacity and sheer bandwidth, while Open Fabrics Interfaces ensure low latency access. DAOS's data path is entirely in userspace, facilitating zero copy through Remote Direct Memory Access (RDMA). Furthermore, it intrinsically presents an object interface, atop which middleware layers like POSIX, MPI-IO, and HDF5 are provided.

DAOS Targets. Each DAOS server node segments its PMEM and NVMe storage into multiple DAOS targets. A distinct DAOS engine daemon associated with every DAOS socket facilitates I/O processes for these targets. For optimized performance and robustness, DAOS objects are distributed across these targets.

DAOS Containers and Snapshots. DAOS containers function as object address spaces, each distinguished by a unique container ID. Snapshots in DAOS are lightweight and are marked with the epoch corresponding to their creation time. After creation, a snapshot remains accessible for reading until it is deliberately removed. Additionally, the contents of a container can be reverted to the state captured in a specific snapshot.

DAOS Object. DAOS introduces Key-Value and Array objects.

Key-Value object offers put and get functions, with the value being a variable-length data block. A single Key-Value object can house numerous key-value pairs. The keys are hashed to determine the DAOS target for storage.

Array provides a logical one-dimensional array. Each DAOS array is defined by its *cell size* and *chunk size*, determined during its creation. The *chunk size* refers to a continuous sequence of elements stored in a target before transitioning to another target. The *cell size* specifies the size of a single element. All elements within a chunk reside in the same DAOS target. The DAOS array facilitates vectored I/O, allowing for read and write operations across multiple distinct extents within a single `daos_array_write/read()` call.

Figure 3 illustrates dense and sparse arrays written by three ranks with extents shorter than the chunk size. This is particularly relevant because the ADIOS metadata of individual writer ranks is typically smaller than the chunk size. In the dense array scenario, the ranks write contiguous extents, all on the same DAOS target. In contrast, in the sparse array scenario, the ranks write at chunk boundaries, and hence the extents are placed on three distinct targets. Given that DAOS targets are bound to limited compute resources, the layout of

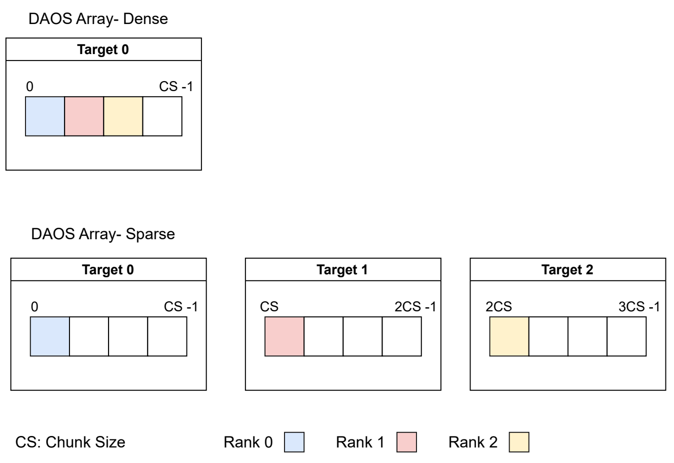


Fig. 3: DAOS array layout

the DAOS array influences performance, as will be discussed subsequently.

DAOS POSIX Emulation over DAOS Filesystem (DFS). DFS offers a structured POSIX namespace, supporting file and directory constructs. Essentially, DFS files are conceptualized using a DAOS array, with a *chunk size* of 1MB and a *cell size* of a single byte. Applications access the emulated POSIX namespace through a FUSE daemon. Additionally, an interception library using LD_PRELOAD ensures a complete OS bypass for POSIX read/write operations.

IV. Design Options with DAOS objects

To investigate the effect of DAOS's object APIs on ADIOS metadata I/O performance, we aimed to develop ADIOS metadata engines based on DAOS objects. The current ADIOS POSIX engine readily aligns with the DAOS POSIX emulation layer. However, leveraging DAOS's KV or Array-based interfaces requires careful consideration of several vital questions.

How to map metadata of writer ranks to DAOS objects?

In the case of DAOS KV, each writer rank uniquely identifies its metadata with a key string for every timestamp. This allows all writer ranks to execute `Put()` operations in parallel.

Conversely, for DAOS arrays, the metadata associated with each writer rank per timestamp is placed as an array extent. If a single DAOS array is employed across all ranks, synchronization between writer ranks becomes necessary to agree upon offsets. An alternative approach is to create separate a DAOS array object for each writer rank. However, this method could lead to the creation of a multitude of DAOS array objects as the number of writer ranks scales. Additionally, the DAOS array Object IDs must also be transferred to the reader ranks. Further, each of these objects must be opened creating additional overhead and state. Hence, we opted for a design that involves shared usage of a single DAOS array object.

How to provide ADIOS timesteps with DAOS Objects?

Given that ADIOS operates on a timestep basis, readers must be able to access all ADIOS metadata related to a specific timestep simultaneously. This can be accomplished in two ways. The first method involves extending the current DAOS objects. In the case of the DAOS array, this means writing new extents, and with KV, it's adding new entries with keys that identify the rank and the timestep. The second method leverages the reuse of DAOS objects in conjunction with the DAOS container snapshot mechanism. This involves overwriting previous extents or key-value entries by a rank in subsequent timesteps. The snapshot creation is delegated to writer rank 0. The metadata of a given ADIOS timestep is accessed by opening the associated snapshot.

How does the difference in I/O semantics of DAOS Array and Key Value impact acquisition time?

Each DAOS KV Get() operation necessitates the registration of an RDMA buffer to receive the metadata. As the number of writer ranks increases, so does the registration overhead in relation to the cost of reading the metadata. However, the DAOS array supports vectored I/O. On the reader side, this enables the reading of metadata from all writers from multiple extents at once into a single contiguous memory block, requiring only a single RDMA memory registration

Metadata acquisition - What is the tradeoff between speedup with concurrency and slowdown due to contention?

In ADIOS, every reader requires metadata from every writer. We considered two approaches, both using asynchronous Get(). The first approach is concurrent, where every reader rank accesses the metadata simultaneously. In the second approach, the task is delegated to reader rank 0, which then broadcasts the metadata to all other reader ranks. When using the concurrent approach and having M writers and N readers, the metadata acquisition entails $M \times N$ sets of complete ADIOS metadata reads. Figure 4 shows metadata acquisition time with 224 writers and increasing number of readers using DAOS KV object. The concurrent approach is clearly not scalable. The cost increase of MPI Bcast with the number of readers, when using delegation, is significantly less than the slowdown caused by concurrent access. Hence, in our design and implementation of DAOS metadata engines, reader-side delegation has been incorporated.

V. Implementation

A. *ADIOS* DAOS-*KV engine*

In the engine's initialization phase, writer rank 0 creates a DAOS KV object and broadcasts its object ID to all other ranks, a process conducted only once. Each writer, at the conclusion of every timestep, stores its metadata as a distinct key-value entry using `daos_kv_put()`. The key is formatted as "StepN-RankID", and the associated value is the metadata buffer.

On the reading side, reader rank 0 gathers all metadata in a two-step process. Initially, it employs `daos_kv_get()` with the key "StepN-RankID" and a NULL value buffer to ascertain the size of each writer rank's metadata.

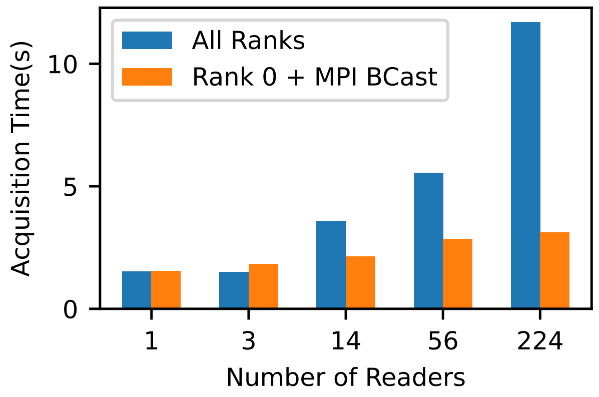


Fig. 4: Comparing the performance of the 'Everyone Rank Reads' approach versus the 'Rank 0 Reads + `MPI_Bcast`' method, implemented using DAOS KV, it is important to note that the total amount of metadata to be consumed remains constant despite an increasing number of reader ranks.

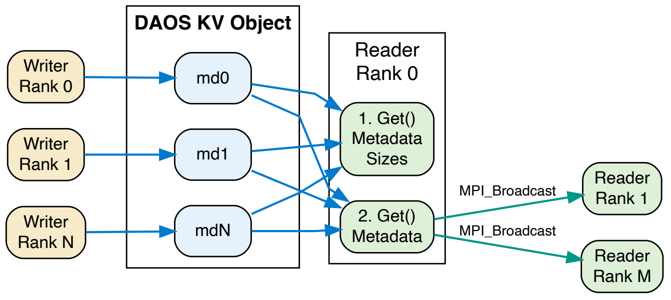


Fig. 5: ADIOS DAOS-KV engine

Once the total metadata size is known, rank 0 allocates the necessary memory. Then, it retrieves the actual metadata using `daos_kv_get("StepN-RankID", metadata_buffer)`. This metadata is subsequently broadcast to all other reader ranks.

B. *ADIOS* DAOS-*Array engine*

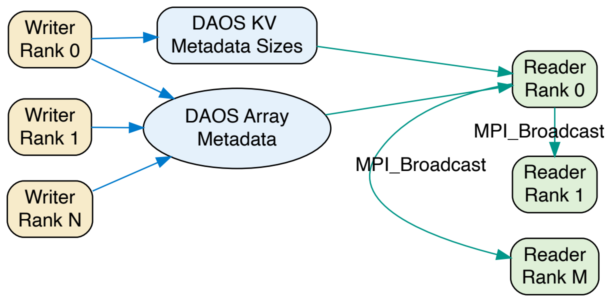


Fig. 6: ADIOS DAOS-Array engine

During the engine's initialization, writer rank 0 creates a DAOS array object and broadcasts its ID to all other ranks. The chunk and cell sizes are set at 1MB and 1 byte, respectively, akin to DFS. A chunk size of 1MB is sufficiently large for current workloads. However, it can be configured to a larger size as required. Additionally, a DAOS KV object is created for storing metadata sizes.

At each timestep, every writer rank shares their metadata sizes using `MPI_Allgather()`. Writers then compute their unique offsets in the DAOS array in rank order and proceed to write their metadata with `daos_array_write()`. Writer rank 0 records the metadata sizes in the KV object with `daos_kv_put("StepN", list_metadata_sizes)`.

On the reader side, rank 0 retrieves the `list_metadata_sizes` from the KV object to calculate the total metadata size. After allocating the required memory, it reads the metadata of all writers from the DAOS array, using `daos_array_read()`. The metadata is then broadcast to all other reader ranks.

VI. Evaluation

A. *Goals*

The expermental evaluation aims to answer the following questions regarding the implications of using DAOS for ADIOS metadata I/O:

- What is the impact of the DAOS-KV and DAOS-Array engines on the stabilization and aquisition time for different ADIOS metadata sizes?
- What is the impact of concurrency on the performance of the DAOS-based engines?
- What are the implications on overall end-to-end metadata transfer time at scale for the different DAOS engines?
- What is the impact of the different DAOS-engines for long running computations with large number of ADIOS timestamps?
- Finally, which DAOS engine is most efficient for metadata transmission?

B. *Setup*

The experiments were performed on the Cambridge Service for Data Driven Discovery (CSD3) Ice Lake cluster. Each compute node is a dual-socket, 38-core Intel® Xeon® Platinum server, with a 2.60GHz 8368Q CPU, with 512GB of DRAM. The system includes DAOS v2.2, configured in pooled mode across ten dual-socket 32-core Intel(R) Xeon(R) Platinum servers based on 2.2GH 8352Y CPUs. Each server has 4.2 TB of PMEM storage, segmented into 256GB Optane DIMMs, and 16 3.8TB NVMe drives. All servers are connected via dual HDR200 InfiniBand network.

C. *Methodology*

Design of the Experiment: For our tests, we separately ran the ADIOS writer vs. reader ranks, for 1000 timesteps each. The rank count was varied from 64 to 1024, distributed across a maximum of 56 compute nodes. The number of readers is the same as number of writers.

Setting Up Metadata Size: We experimented with metadata size ranging from 5K to 56KB per rank per timestamp. 56KB corresponds to the metadata sizes for E3SM. To configure the metadata size we changed the number of ADIOS variables and corresponding arrays. The per-rank metadata size is constant, thus the total metadata size grows proportionally with the rank cound.

Measurements: To gather timing data within ADIOS we used Caliper, a tool designed for performance measurement and code instrumentation in high-performance computing, [14]. Specifically, Caliper was utilized at the EndStep() and BeginStep() points of the DAOS engines to record metadata stabilization and acquisition times. Since ADIOS writers generate similar metadata across timesteps, the reported times represent the average per rank over 1000 timesteps.

D. *Results*

1) E3SM - 56KB

In Figure 7, with a metadata size of 56KB, both the DAOS-KV and DAOS-Array show stabilization times that are up to an order of magnitude faster than DAOS-POSIX. The stabilization time of POSIX is not sustainable with the growing number of ranks as it serializes metadata stabilization through rank 0. On the other hand, both DAOS-KV and DAOS-Array write metadata in parallel.

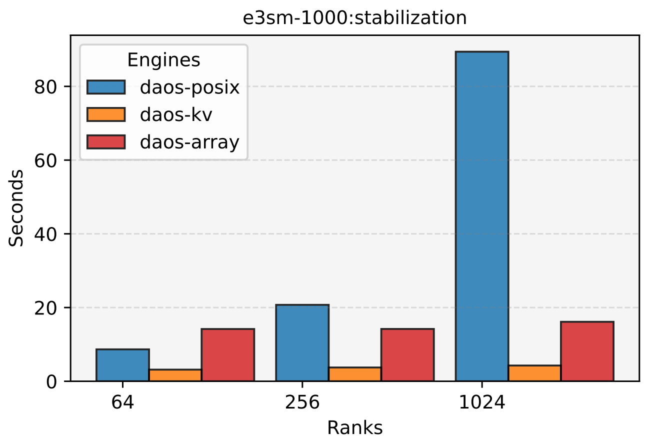


Fig. 7: E3SM(56KB) - Stabilization Time

Figure 8 shows the acquisition time for E3SM. The acquisition time consists of the shaded MPI Bcast time and the unshaded region, which represents the I/O read time. The MPI Bcast time depends on the number of readers, while the I/O time is affected by the number of writers. The choice of the DAOS engine influences only the I/O read time. The DAOS-KV-async-get acquisition time is 1.2X slower than DAOS-Array at 1024 ranks. In case of DAOS-KV-async-get issuing 1024 `daos_kv_get()` at each timestep proves to be expensive. Each invocation of `daos_kv_get()` requires a separate RDMA buffer registration. As the number of KV entries increases with increasing writer ranks, this overhead becomes significant. However, `daos_array_read()` performs vectored I/O for many extents into a single contiguous buffer, requiring only one RDMA buffer registration.

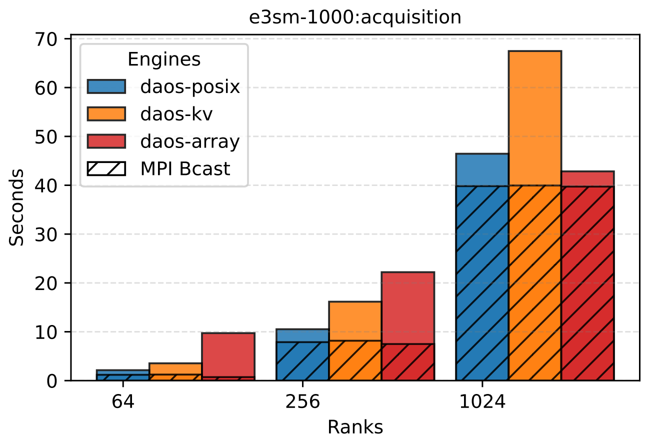


Fig. 8: E3SM(56KB) - Acquisition Time

E. *Small Metadata - 5KB*

Figure 9 depicts the metadata stabilization time for a small metadata size of 5KB. Unlike in E3SM, which has a 56KB metadata size as shown in Figure 7, the stabilization time for DAOS-Array here is 25X slower than for DAOS-KV. In the DAOS-Array engine, writer ranks write their respective metadata in rank order contiguously into the array. This results in a dense array, as illustrated earlier in Figure 3, reducing the number of targets where the metadata is stored. Each target has a limited number of RDMA buffers. The numerous concurrent `daos_array_write()` operations contend for these, causing the significant slowdown.

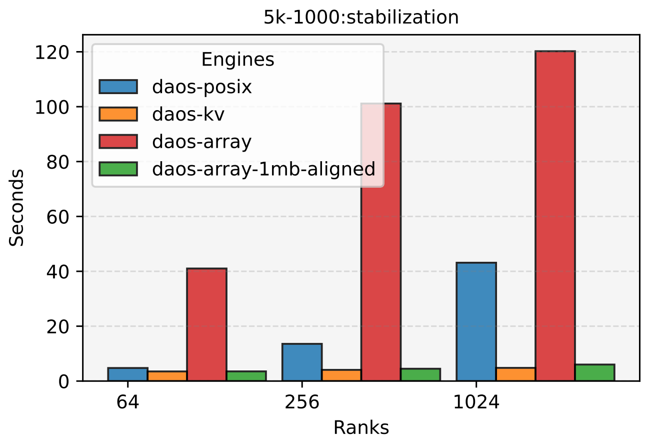


Fig. 9: 5KB - Stabilization Time

To further illustrate this DAOS-Array behavior, we created a custom benchmark with 5KB and 56KB metadata sizes, using 200 and 20 writer ranks, respectively. The number of timesteps are 100. The total data written in a timestep for both scenarios is approximately 1MB. However, 5KB writes this time are explicitly aligned to 56KB boundaries in the DAOS-Array. This alignment ensures a more sparse layout and distribution of data across more targets. For reference, we present the 5KB results with 200 writers without the alignment. Figure 10 shows similar `daos_array_write()` times for 5KB with alignment and 56KB, unlike the distinct difference observed between 56KB and 5KB in figures 7 and 9.

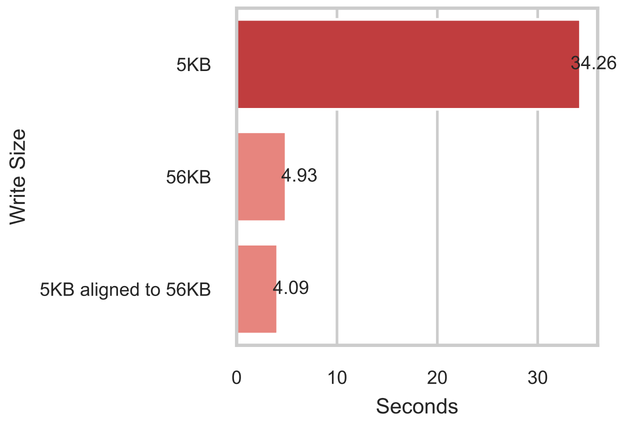


Fig. 10: Comparison of `daos_array_write()` times for Aligned 5KB and 56KB Metadata Sizes

Going back to Figure 9 the DAOS-Array-ChunkSize-aligned engine effectively addresses this issue. This ensures better metadata distribution across targets, thus reducing contention. As a result, the stabilization time for DAOS-Array-ChunkSize-aligned is significantly improved, being 25X faster than DAOS-Array and on par with DAOS-KV.

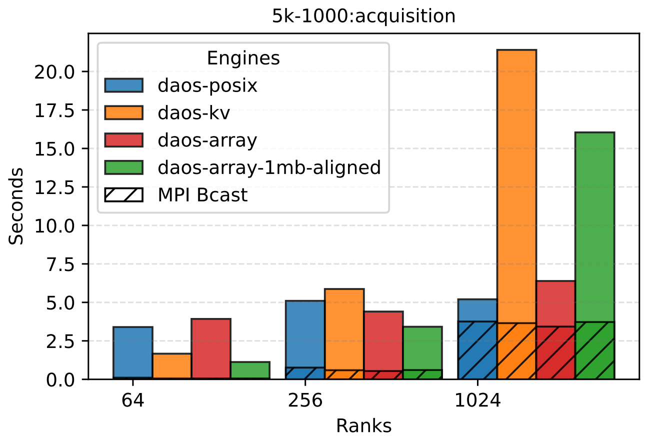


Fig. 11: 5KB - Acquisition Time

Figure 11 shows the acquisition times for 5KB metadata. In DAOS-KV, for `daos_kv_get()`, values smaller than 20KB are returned inline after the RPC call, without requiring RDMA transfers. However, the numerous inline executions are not as efficient compared to the vectored I/O provided by `daos_array_read()`. At 1024 ranks, DAOS-KV-async-get is 1.33X and 3.35X slower than DAOS-Array and DAOS-Array-ChunkSize-aligned, respectively. Although DAOS-Array-ChunkSize-aligned has a clear advantage on the writer side compared to DAOS-Array, on the reader side, it is 2.5X slower than DAOS-Array. This is because in DAOS-Array, the metadata is laid out contiguously, resulting in larger chunk size(1MB) reads from individual targets, as opposed to numerous small 5KB reads from many targets.

1) End-to-End Transfer Time

The end-to-end transfer times for E3SM and 5KB metadata are shown in Figures 12a and 12b, respectively. We evaluated E3SM with DAOS-Array-ChunkSize-aligned and its end-to-end transfer time is almost equal to DAOS-Array, while being 2.3X faster than DAOS-POSIX and showing a 23% speed-up over DAOS-KV-async-get. The stabilization time of DAOS-POSIX dominates its metadata end-to-end transfer time. For the 5KB metadata, DAOS-Array-ChunkSize-aligned again has the best metadata end-to-end transfer time, being 2.1X faster than DAOS-POSIX. The end-to-end time of DAOS-Array suffers due to poor stabilization time and is hence not shown in Figure 12a for readability. Although the acquisition time of DAOS-Array-ChunkSize-aligned is slower than that of DAOS-Array, the gains in stabilization time provide a 20% speed-up in end-to-end transfer over DAOS-KV-async-get.

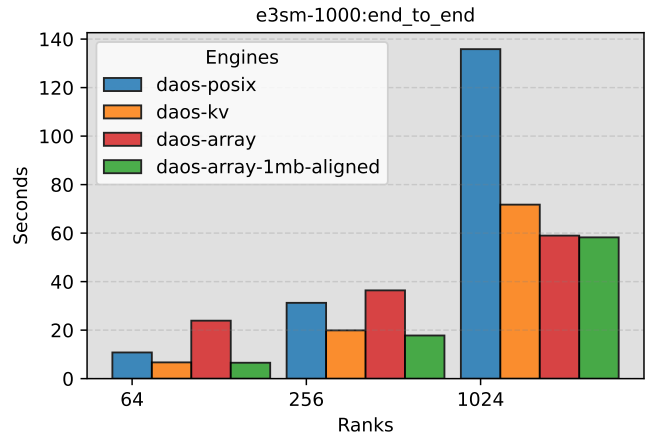
F. Stabilization and Acquisition Times at large number of ADIOS timesteps

Slow Stabilization - Overwrite DAOS Array + Snapshots

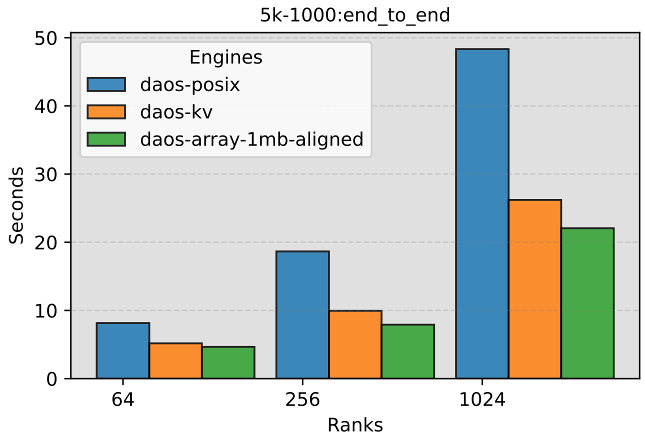
The DAOS container snapshot mechanism aligns well with the need for consistent ADIOS metadata of a given timestep. In this method, all writer ranks are programmed to write metadata at predetermined offsets determined by their rank order. At the end of each timestep, rank 0 captures a snapshot of the container. Figure 13 illustrates the comparison of stabilization times using DAOS array with and without snapshots across an increasing number of ADIOS timesteps, using only 66 writer ranks. With snapshots, writers overwrite their metadata at the same offsets, whereas without snapshots, metadata is written at new offsets for each timestep. The performance of the snapshot-based approach degrades with an increasing number of timesteps. In the DAOS array, overwritten extents are flagged for subsequent garbage collection. While snapshots prevent storage space reclamation, triggering garbage collection later necessitates a costly traversal of DAOS's internal structures. The cost of this overhead increases with the number of overwritten extents. Conversely, the non-snapshot approach avoids overwrites, eliminating performance declines due to garbage collection and consistently achieving optimal stabilization times, even at large number of ADIOS timesteps.

Revisiting E3SM at large number of ADIOS timesteps

The snapshot-based engine did not perform optimally beyond 1000 steps. To understand the performance of DAOS-Array-ChunkSize-aligned and DAOS-KV-async-get engines with long-running applications that produce a large number of ADIOS timesteps, we calculated the stabilization and acquisition times per ADIOS timestep. Our earlier results with E3SM-based benchmarks were limited to 1000 timesteps. We reran the experiment, extending it up to 10,000 timesteps. Figure 16



(a) End-to-End Transfer Time for E3SM(56KB)



(b) End-to-End Transfer Time for 5KB

Fig. 12: End-to-End transfer times for different metadata sizes

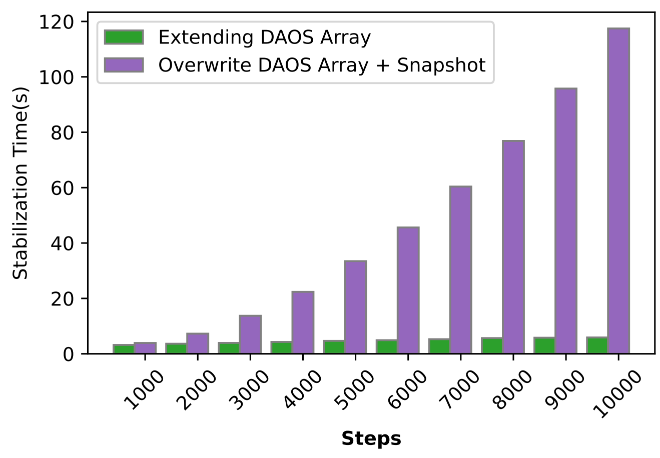


Fig. 13: Comparison of Stabilization Time with and without DAOS Snapshots using 66 Writer Ranks

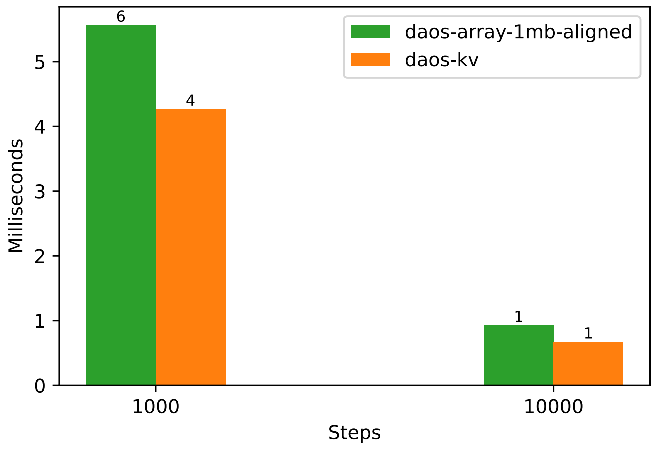


Fig. 14: Per TimeStep Stabilization Time for E3SM(56KB)

displays the per-timestep stabilization time, which decreases fivefold when moving from 1,000 to 10,000 timesteps. This decrease occurs because the initial setup costs are amortized as the number of timesteps increases. These costs include initial I/O tasks, such as connection setup and queue pair creation with the verbs provider. In Figure 17, the per-timestep acquisition time remains relatively constant from 1,000 to 10,000 timesteps. This is due to the fact that, although the same initial costs are encountered, they are amortized much faster since the amount of metadata consumed per reader rank is significantly larger than the metadata generated per writer rank. However, the key point here is that, unlike the container snapshot, DAOS-Array-ChunkSize-aligned is well-suited for extended ADIOS applications.

In addition to measuring the impact of a large number of ADIOS timesteps on metadata transfer performance, it is also important to measure the scaling cost associated with the number of ranks. Therefore, we measured the stabilization and

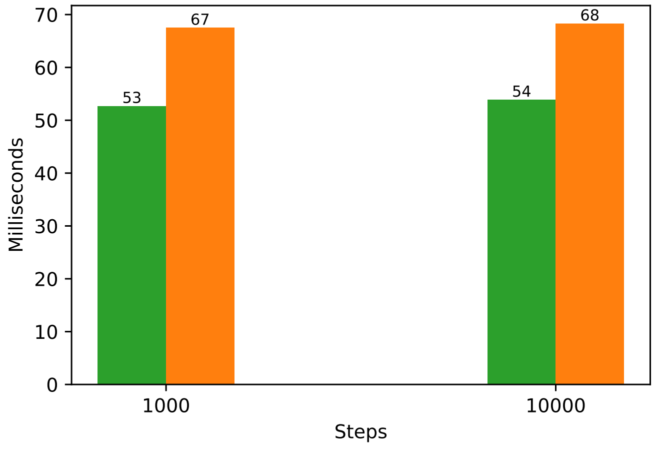


Fig. 15: Per TimeStep Acquisition Time for E3SM(56KB)

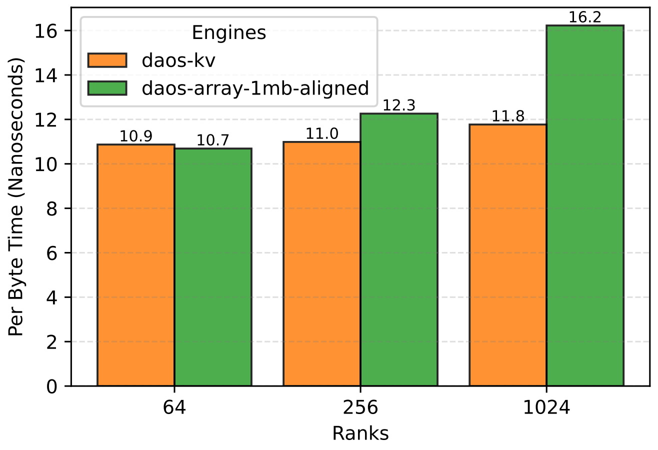


Fig. 16: Per Byte Stabilization Time over 10,000 Steps for E3SM(56KB)

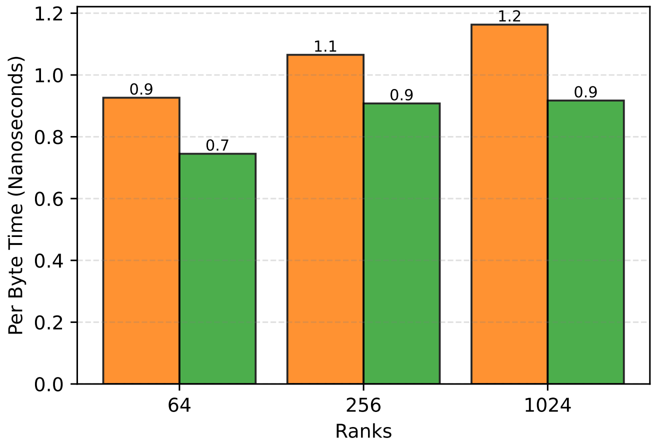


Fig. 17: Per Byte Acquisition Time over 10,000 Steps for E3SM(56KB)

acquisition time per byte. The Per Byte Time is calculated for each rank as the total data stabilized or acquired, divided by the time. Figures 16 and 17 show that the Per Byte Time increases marginally with an increasing number of ranks, and the total metadata quadrupled.

G. Storage overheads of using parse DAOS array

The DAOS array chunk size serves as a unit of storage distribution, not allocation. It ensures that all array elements from one chunk boundary to the next are guaranteed placement on the target. However, when writing extents of sub-chunk size, the entire chunk size is not allocated. Although additional DAOS object metadata storage overhead could occur, there is no measurable difference in free space available on DAOS after DAOS-Array and DAOS-Array-ChunkSize-aligned executions completed. We reran the E3SM benchmark with 56 KB metadata per rank per timestep using 1024 ranks increasing the timesteps to 20,000, creating an aggregate metadata of 1.14 TB. If DAOS-Array-ChunkSize-aligned allocated 1 MB chunks, this would require 20 TB of storage. However, the DAOS storage free space query reported that both DAOS-Array and DAOS-Array-ChunkSize-aligned had the same amount of free space.

VII. Summary of Contributions

In our evaluations, we have shown that POSIX-based metadata handling in ADIOS incurs a significant cost in real-world applications such as E3SM. The DAOS Array and KV APIs provide an opportunity to overcome the metadata transfer problem. However, implementing DAOS in the ADIOS context was not straightforward. We presented different design options concerning data layout and concurrency, and evaluated their trade-offs on the writer and reader sides.

The new DAOS-Array-ChunkSize-aligned engine provides optimal end-to-end transfer time for varying metadata sizes with large number of ADIOS timesteps. We comapred the performance of the DAOS-Array-ChunkSize-aligned engine with the DAOS-KV-async-get and the DAOS-POSIX engines. The stabilization times of both DAOS-KV-async-get and DAOS-Array-ChunkSize-aligned methods are an order of magnitude faster than DAOS-POSIX. Although DAOS-KV-async-get and DAOS-Array-ChunkSize-aligned have similar stabilization times, the acquisition time of DAOS-Array-ChunkSize-aligned is upto 23% faster than DAOS-KV-async-get. The end to end transfer time of DAOS-Array-ChunkSize-aligned is up to 2.3× faster than DAOS-POSIX. This new engine will be open sourced and integrated into ADIOS2 codebase. With this, applications like E3SM can run over DAOS without any code modifications, thus circumventing the need for application level data merge and thereby eliminating programmer effort to reduce metadata overhead.

VIII. Related Work

Jialin Liu et.al [15] undertook an assessment of object stores in the context of HPC I/O. Their findings underscored the superior scalability of object stores compared to POSIX. The study employed three HDF5 Virtual Object Layer plugins specifically designed for Ceph/RADOS [16], Openstack Swift, and Intel DAOS. It is observed that the predominant I/O granularity in most object stores is the entire object, contrasting the more refined granularity observed in POSIX. Nevertheless, the DAOS array API offers I/O descriptors that allow for selective access to sections of the object. In terms of both I/O bandwidth and associated costs, DAOS outperformed RADOS and Swift. Liu et al. highlighted the necessity for supplementary tools to map the hierarchical data model of HDF5 onto the flat namespace inherent to objects. While ADIOS may not support a hierarchical data model, it does accommodate multidimensional arrays, implying the presence of analogous requirements in ADIOS. Another study [17] delves into the optimal utilization of Optane for transporting data within HPC workflows.

A recent study [18] explored the performance of HDF5 over the DAOS object interface. This object-centric design enabled HDF5 to transition away from traditional block-based storage, thereby circumventing the constraints posed by POSIX. Within the realm of file-based storage, HDF5 object instantiation necessitated coordination amongst ranks, leading to resource-intensive I/O collectives. Yet, when integrated with DAOS, the time taken for HDF5 object creation witnessed a significant reduction. Additionally, the integration with the DAOS Key-Value interface paved the way for a novel HDF5 map feature. The DAOS HDF5 VOL also exhibited support for asynchronous I/O, which in turn amplified storage bandwidth utilization.

IX. Conclusions

In this paper, we have examined several approaches to storing ADIOS-level metadata in DAOS, examining both the KV and Array interfaces and comparing the performance of these approaches to the current POSIX-based metadata storage mechanism in ADIOS. Our measurements have shown that an Array-based approach with each ADIOS rank writing their metadata contribution at increasing offsets corresponding to the DAOS chunk size is the best of the approaches we studied and in fact is up to 2.3x faster than storing metadata in a POSIX file. While some of our experiments were carried out in a simulation environment that mimicked the activities of an ADIOS engine storing metadata, we have produced a usable DAOS engine in ADIOS that uses the Array interface to store ADIOS metadata in DAOS. At this time the DAOS engine, derived from the ADIOS BP5 engine, still uses POSIX files (via DAOS POSIX support) to store data, but we hope that the insights we have gained in this work will guide future work which will result in an all-DAOS object engine with improved data storage behaviour as well.

This paper also adds to the body of work exploring the performance impact of middleware-level metadata in HPC I/O. As noted in Section II, metadata-heavy applications like E3SM can devote 60-70% of data writing time is allocated to metadata overheads. We hope that this work enables future research, both in middlware-level metadata handling and generally in the use of DAOS in exascale HPC scenarios.

References

[1] P. Braam, "The lustre storage architecture," *arXiv preprint arXiv:1903.01955*, 2019.

[2] F. Schmuck and R. Haskin, "{GPFS}: A {Shared-Disk} file system for large computing clusters," in *Conference on file and storage technologies (FAST 02)*, 2002.

[3] S. R. Alam, H. N. El-Harake, K. Howard, N. Stringfellow, and F. Verzelloni, "Parallel i/o and the metadata wall," in *Proceedings of the sixth workshop on Parallel Data Storage*, 2011, pp. 13–18.

[4] R. Macedo, M. Miranda, Y. Tanimura, J. Haga, A. Ruhela, S. L. Harrell, R. T. Evans, J. Pereira, and J. Paulo, "Taming metadata-intensive hpc jobs through dynamic, application-agnostic qos control," in *23nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE*, 2023.

[5] Argonne National Laboratory, "Aurora Supercomputer," https://www.alcf.anl.gov/aurora, Argonne Leadership Computing Facility, 2023, accessed: July 18, 2023.

[6] J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent, and E. Barton, "Daos and friends: a proposal for an exascale storage system," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 585–596.

[7] M. Hennecke, "Daos: A scale-out high performance storage stack for storage class memory," *Supercomputing frontiers*, p. 40, 2020.

[8] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor *et al.*, "Basic performance measurements of the intel optane dc persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[9] IO500, "IO500 Submission 621," https://io500.org/submissions/view/621, IO500, 2023, accessed: July 18, 2023.

[10] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield *et al.*, "Hello adios: the challenges and lessons of developing leadership class i/o frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014.

[11] Association for Computing Machinery (ACM), "Using next generation exascale supercomputers to understand the climate crisis," https://www.acm.org/media-center/2023/november/gordon-bell-climate-2023, 2023, accessed: [insert date of access].

[12] "E3sm model description - version 1," https://e3sm.org/model/e3sm-model-description/v1-description/, Year of publication if available, otherwise approximate or omit, accessed: [Insert date of access here].

[13] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible io and integration for scientific codes through the adaptable io system (adios)," in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, 2008, pp. 15–24.

[14] D. Boehme, T. Gamblin, D. Beckingsale, P.-T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz, "Caliper: performance introspection for hpc software stacks," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 550–560.

[15] J. Liu, Q. Koziol, G. F. Butler, N. Fortner, M. Chaarawi, H. Tang, S. Byna, G. K. Lockwood, R. Cheema, K. A. Kallback-Rose, D. Hazen, and M. Prabhat, "Evaluation of hpc application i/o on object storage systems," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2018, pp. 24–34.

[16] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pp. 307–320.

[17] R. S. Venkatesh, T. Mason, P. Fernando, G. Eisenhauer, and A. Gavrilovska, "Scheduling hpc workflows with intel optane persistent memory," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 56–65.

[18] J. Soumagne, J. Henderson, M. Chaarawi, N. Fortner, S. Breitenfeld, S. Lu, D. Robinson, E. Pourmal, and J. Lombardi, "Accelerating hdf5 i/o for exascale using daos," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 903–914, 2021.