Optimizing Metadata Exchange: Leveraging DAOS for ADIOS Metadata I/O

Ranjan Sarpangala Venkatesh
Georgia Institute of Technology
ranjansv@gatech.edu

Greg Eisenhauer
Georgia Institute of Technology
eisen@cc.gatech.edu

Norbert Podhorszki
Oak Ridge National Laboratory
pnorbert@ornl.gov

Dmitry Ganyushin
Oak Ridge National Laboratory
ganyushindi@ornl.gov

Scott Klasky
Oak Ridge National Laboratory
klasky@ornl.gov

Ada Gavrilovska
Georgia Institute of Technology
ada@cc.gatech.edu

***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 application modifications.**

***Index Terms*—HPC, workflow I/O, metadata, ADIOS, DAOS, object storage, data management**

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 emerging 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 different 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. ADIOS has been used in several recent Gordon Bell Prize-winning applications, including the particle-in-cell code WarpX [11] and the climate simulation E3SM [12].

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:

- 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 WarpX application (§II).
- We present the complex design space in using DAOS Key-Value and Array objects for the transfer of ADIOS metadata (§IV), and the concrete decisions incorporated in the new DAOS-based engines we contribute to ADIOS (§V).
- Using benchmarks with metadata I/O patterns representative of E3SM and WarpX, we evaluate the DAOS metadata engines in ADIOS to measure the impact of metadata size, number of ranks, and of long-running applications on committing and acquiring metadata, and on end-to-end metadata transfer time (§VI).
- Our new DAOS array-based engine provides 2.3× faster ADIOS metadata end-to-end transfer time than POSIX at scale. For the WarpX application, we show that use of DAOS can reduce metadata I/O time by more than 4×, from more than 20% of the application I/O time, down to just 5%.

Please note that while this study examines a metadata pattern in ADIOS where each writer rank generates metadata for its data items, and each reader rank uses this metadata to locate required data, we believe that this pattern is generic enough to represent the metadata storage and access needs of other HPC I/O middleware like HDF5 and PnetCDF. *While we have grounded our experiments in ADIOS and compare to existing ADIOS POSIX-based implementation of metadata storage, we believe that these results are generalizable to other systems with similar metadata storage and access needs.*

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. Metadata is generally presumed to be relatively small as compared with the data and therefore not of particular interest in HPC I/O.

To counter this perspective, we introduce the performance benchmarks of Figure 1, which illustrates characteristics of

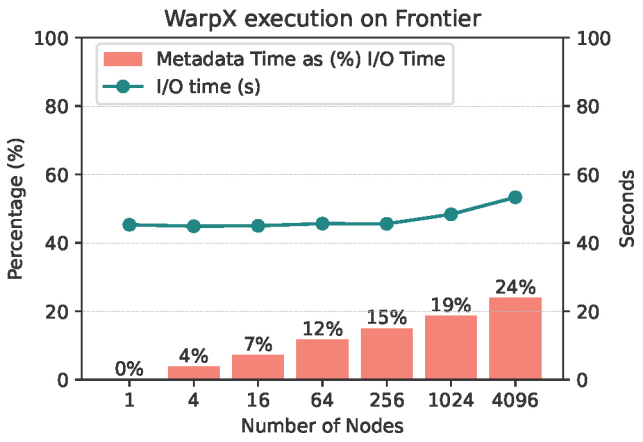


Fig. 1: The Escalating Relative Cost of ADIOS Metadata Time in Comparison to Overall I/O Time

WarpX I/O at various scales (measurements performed on Frontier). In particular, the figure shows the overall I/O time (green line, scale at right) and relative contribution of the metadata gather time within that I/O time (percentage represented by orange bars). With increasing scale, the per-rank I/O time remains relatively constant with ADIOS writing data from all CPU ranks. However, the ADIOS metadata gather and write time increases to as much as 24% of the total I/O time. These trends are not unique to WarpX, but have also been captured for other metadata heavy applications, including E3SM.

In the existing ADIOS POSIX-based implementation of metadata handling, all writer metadata is gathered via MPI to a single rank and written into a POSIX file, from which it will be later accessed by reading applications. In this work we will examine the implications of replacing this file-based approach with an object-based approach implemented with DAOS. As mentioned above, other self-describing HPC I/O middleware may handle metadata differently, such as bundling it with data, but as metadata access needs tend to be different than that of data, separation is useful. We also assert that this general metadata storage/access pattern, where metadata generated on individual writer ranks must somehow be made accessible to each reader rank to enable data access, is generic enough to represent the metadata handling of other HPC I/O middleware.

Note that while one might also consider how to improve file-based metadata storage and access approaches, that is not our goal in this work. Instead we focus on how this specific situation provides opportunities to benefit from the use of the DAOS object model. To that end, we primarily use the existing implementation as motivation and as a performance baseline. This is particularly useful because 1) the existence of the DAOS POSIX interface allows us to compare two disparate interfaces in the same hardware environment, and 2) the existing POSIX implementation was adequate for ADIOS in less demanding circumstances, so it's not so wildly deficient as to be an inappropriate comparison.

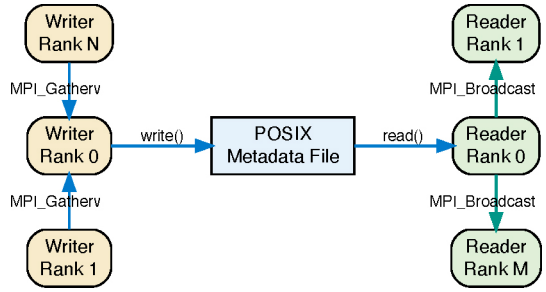


Fig. 2: ADIOS - POSIX engine

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 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.

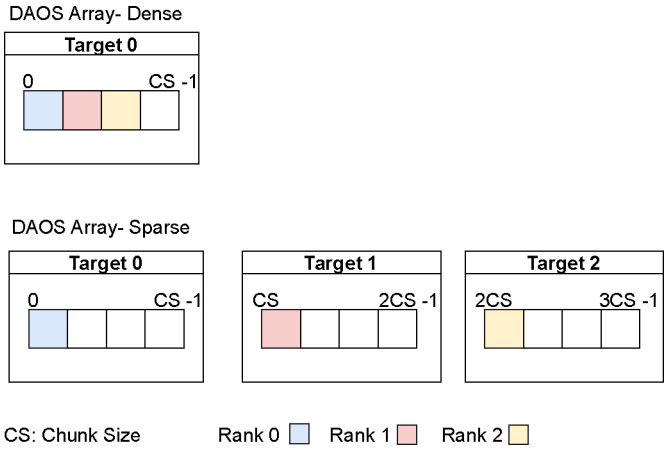


Fig. 3: DAOS array layout

For evaluating the performance and design tradeoffs 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 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

Although the DAOS KV and Array interfaces provide simple interfaces, there are many different options regarding how they could be used and configured. Deciding on the options that would be most suitable for a given API and will be effective in providing optimal end-to-end performance is not straightforward. Therefore, to best leverage DAOS's KV or Array-based interfaces, requires careful consideration of the design space along several dimensions, summarized in Figure 4.

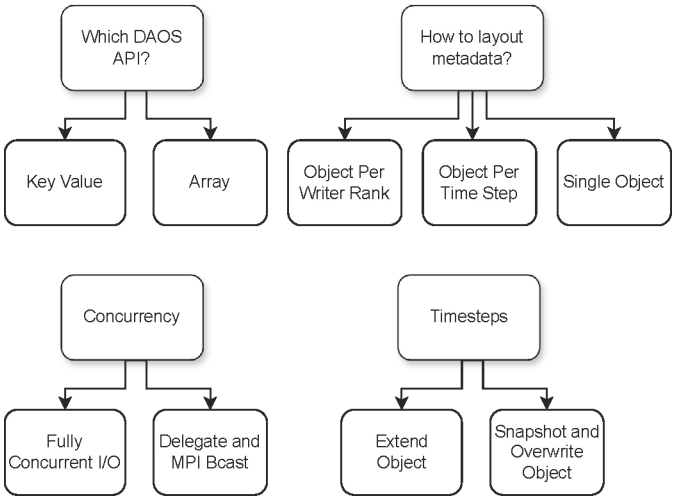


Fig. 4: Design Space Layout

Before we delve into the options, it's important to note how DAOS objects differ from the common notion of an object. A DAOS KV object is not a single key-value pair, but rather a "store" in itself, where many key-value pairs can be stored in the same DAOS KV object. On the other hand, a DAOS array object hosts a single array. However, they are similar to files with extensible extents and they could be dense or sparse.

Furthermore, there are distinctions in the I/O semantics between KV and Array that have performance implications. Metadata written into DAOS KV objects does not require coordination among writers. Each writer can update its metadata with a unique key string. However, a DAOS array may require coordination among writer ranks to agree on array offsets if the object is shared. On the reader side, DAOS array can take advantage of vectored I/O, which enables reading of metadata from all writers from multiple extents at once into a single contiguous memory block. However, in the case of KV, each metadata blob would require a separate DAOS KV operation. Given these pros and cons, we evaluate both design options with KV and Array.

Metadata Layout

The layout of metadata over DAOS objects creates different tradeoffs in stabilization and acquisition time. For both DAOS APIs we considered three metadata layout options:

- Object Per Writer Rank,
- Object Per Time Step, or
- Single Object.

Object Per Writer Rank: As the number of writers scales to thousands of ranks, an equal number of DAOS objects have to be created. DAOS is known to efficiently handle the creation of a large number of objects. However, before actual

writing or reading can happen, these objects have to be opened. Simultaneous large numbers of opens can add to the startup time of the application. All opens can be made at once in the beginning, and the same handle can be reused for the entire run of the simulation, meaning the associated overhead is a function of the application scale, and does not increase over time. An important limitation of this option is that the DAOS array does not benefit from vectored I/O on the acquisition side since each writer rank has a separate object. Hence, the memory registration costs would be similar to KV.

Object Per Time Step: In this option, the DAOS array would benefit from vectored I/O on the acquisition side since all metadata of an ADIOS timestep is stored in the same object. On the reader side, each timestep needs a new open call. This repeated creation and opening of new objects would impact both stabilization and acquisition time for long-running applications.

Single Object: In this option during startup, writer rank 0 performs the object creation, issues a single open call on the object, obtains an object handle, and broadcasts it to all other ranks. The same handle is reused across timesteps. A similar set of steps happens on the reader side. Since a single object is used, there are fixed overheads associated with object creation, regardless of scale or duration of the experiment. Importantly, the DAOS array benefits from vectored I/O on the acquisition side.

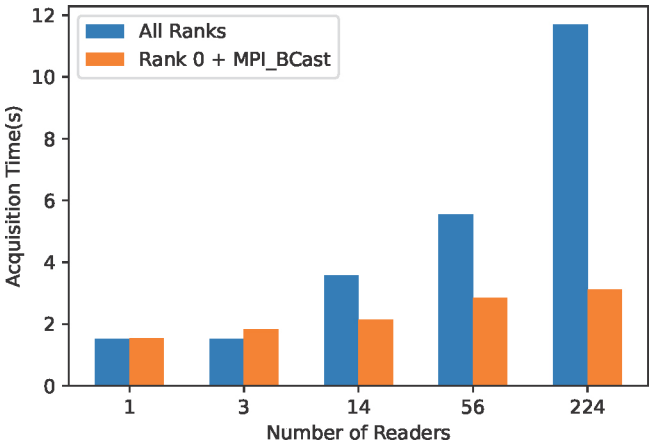


Fig. 5: Comparing the performance of the 'Everyone Rank Reads' approach versus the 'Rank 0 Reads + `MPI_Bcast`' method, implemented using DAOS KV. It is important to note that the total amount of metadata to be consumed remains constant despite an increasing number of reader ranks.

Reader and Writer Concurrency

In ADIOS, every reader requires metadata from every writer. We considered two approaches, both using asynchronous Get() as synchronous Get() are too slow for consideration with large number of writers. 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 × N sets of complete ADIOS metadata reads. The delegation approach, on the other hand, requires only M sets of complete ADIOS metadata reads, followed by N MPI Bcast operations. Figure 5 shows metadata acquisition time with 224 writers and an increasing number of readers using the 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. The DAOS array would also have the similar tradeoffs as the KV. Hence, in our design and implementation of DAOS metadata engines, reader-side delegation has been incorporated.

On the writer side, metadata writes are parallelized to avoid serialization which has been shown to be expensive in the motivation.

ADIOS Timestep Management

Given that ADIOS operates on a timestep basis, how we represent timesteps over time 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.

The reuse with snapshot method simplifies the passing of metadata offsets in the case of the DAOS array to readers for a given timestep. However, on the reader side, before the metadata can be accessed, it requires a transactional open() of the snapshot.

Design Decision

Considering the above tradeoffs and the requirements for ADIOS metadata I/O, we prioritize the following design choices. Given the ease of use, performance benefits with vectored I/O for the DAOS array, and minimized startup costs and overhead in the transfer of DAOS object metadata to readers (distinct from ADIOS metadata), we choose a single object to host all metadata. Metadata stabilization is parallelized with concurrent writes from all writer ranks. However, the metadata acquisition is delegated to reader rank 0 and broadcasted. The ADIOS timestep management is based on extending DAOS objects. In Section VI we justify this choice by evaluating the feasibility of a snapshot-based mechanism.

V. DAOS Engines for ADIOS Metadata I/O

Based on the design decisions derived from the design space explorations described in the previous section, we examine

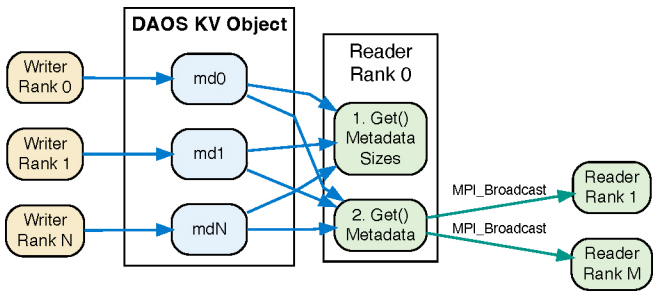


Fig. 6: ADIOS DAOS-KV engine: Each writer's metadata is stored as a key-value pair in a single object.

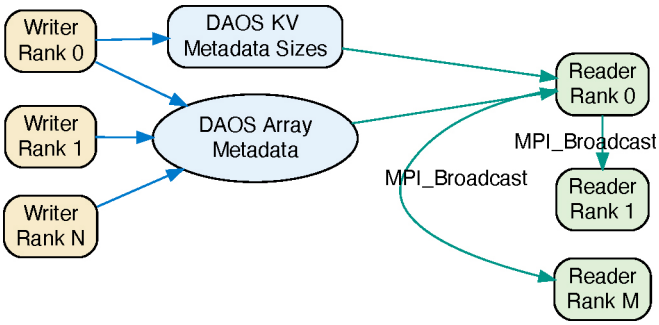


Fig. 7: ADIOS DAOS-Array engine: Each writer's metadata is stored as an extent in a single array.

how they are applied within KV and Array-based ADIOS metadata I/O engines.

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.

As shown in Figure 6, 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. 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*

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)`.

As shown in Figure 7, 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 experimental 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 acquisition 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?

The ultimate goal is to determine which DAOS engine is most efficient for metadata transmission.

B. Methodology

Experimental testbed. The experiments are 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 2.60GHz 8368Q CPUs 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.

Design of experiments. In order to capture the impact of the different DAOS engines on both stabilization and acquisition time, we first evaluate the two phases separately (§VI-C). Subseqeuntly we combine these measurements to report the impact on end-to-end measurements (§VI-E). These measurements consider scenarios with different levels of parallelism, and with different per-rank and per-timestamp metadata size. We further investigate the implication of long-running simulations on the performance of different engines (§VI-F), and their runtime overheads, particularly considering with the sparse DAOS-Array engine (§VI-G).

Benchmarks. The motivating observations regarding metadata-related I/O issues were made on ORNL's Summit and Frontier systems, neither of which has DAOS. While our next steps involve continuing this investigation on Aurora, for the evaluations presented in this paper, we only had access to a DAOS system via the UK Cambridge Cluster. In order to focus the evaluation on the metadata I/O behavior, and avoid the complexities of configuring complex application workflows, we relied on benchmarks that capture the real applications metadata exchanges. In ADIOS, every writer rank at the end of every timestep produces a metadata blob, which contains information about all the variables written by it. Hence, for the purposes of our evaluations, we used an ADIOS tool [14] that generates the same size of metadata per writer rank per timestep. 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 count.

We increase the number of ranks ranging from 64 to 1024. The writers and readers executed over compute partitions, while DAOS servers were deployed on a separate set of storage nodes. The number of readers was set to match the number of writers.

To corroborate the trends captured with the benchmark results, we also include the stabilization time measurements with a complete application, WarpX, performing laser wakefield simulation. The relative performance of the DAOS engines observed in WarpX is similar to those observed with the benchmarks.

Measurements. To gather timing data within ADIOS we use Caliper, a tool designed for performance measurement and code instrumentation in high-performance computing, [15]. We use Caliper 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.

C. Impact of DAOS Engines on Stabilization and Acquisition Times

We provide a breakdown of the metadata stabilization and acquisition time for application scenarios with different amount of metadata generated per rank, per iteration. The first scenario corresponds to the E3SM climate modeling application, with 56 KB metadata sizes. In the second scenario, we focus on much smaller metadata, 5 KB, in order to assess how metadata size impacts the relative differences among different DAOS engines. The smaller metadata size is similar to the metadata sizes in WarpX, for which we present measurements later in this section.

1) Large Metadata - E3SM (56KB)

For the case with metadata size of 56KB, shown in Figure 8, both the DAOS-KV and DAOS-Array result in stabilization times 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.

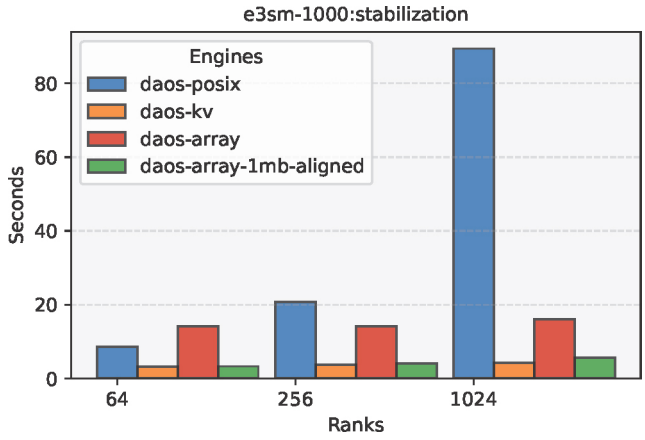


Fig. 8: Comparison of Stabilization Time across DAOS Engines in E3SM (56KB)

Figure 9 shows the acquisition time for the E3SM case. 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 1.2× slower than DAOS-Array at 1024 ranks. The 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.

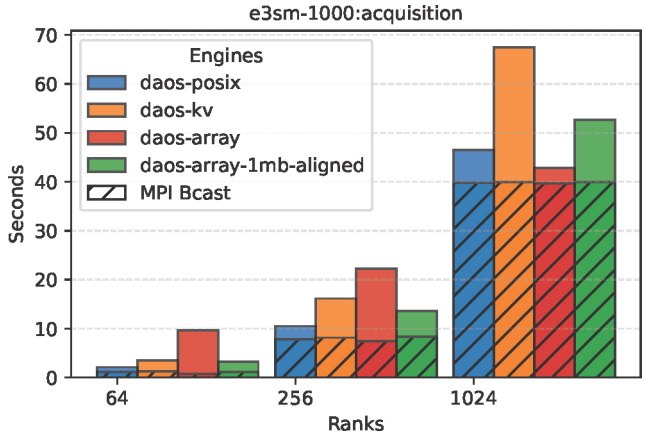


Fig. 9: Comparison of Acquisition Time across DAOS Engines in E3SM (56KB)

D. Small Metadata - 5KB

Figure 10 depicts the metadata stabilization time for a small metadata size of 5KB. Unlike in the previous case, the stabilization time for DAOS-Array here is 25× 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 storage 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.

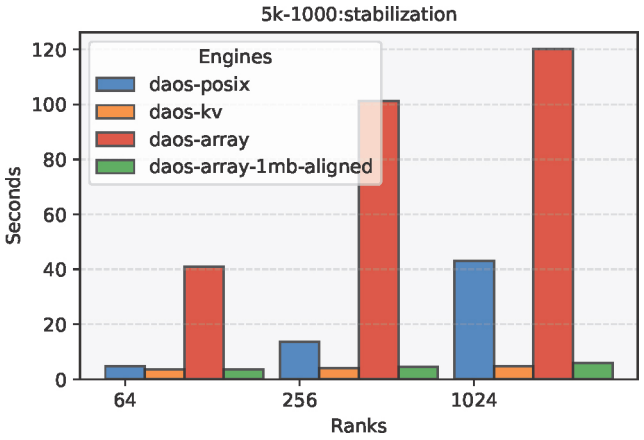


Fig. 10: Comparison of Stabilization Time across DAOS Engines for small metadata (5KB)

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. We explicitly align the 5KB writes 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 11 shows similar `daos_array_write()` times for 5KB with alignment and 56KB, unlike the distinct difference observed between 56KB and 5KB in Figures 8 and 10.

Going back to Figure 10 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 25× faster than DAOS-Array and on par with DAOS-KV.

Figure 12 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 1.33× and 3.35× slower than DAOS-Array and DAOS-Array-ChunkSize-aligned, respectively. Although

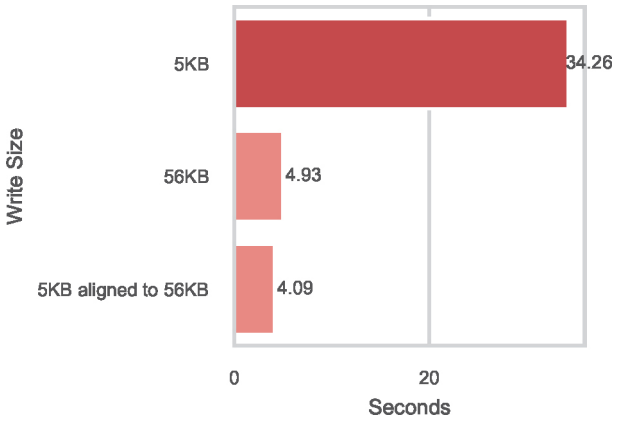


Fig. 11: Comparison of `daos_array_write()` times for Aligned 5KB and 56KB Metadata Sizes

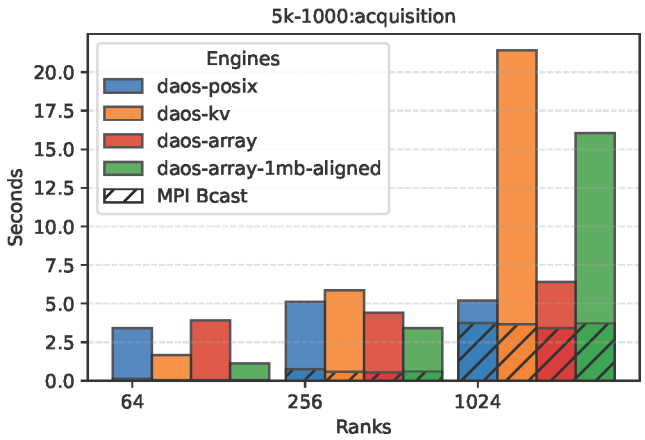


Fig. 12: Comparison of Acquisition Time across DAOS Engines for small metadata (5KB)

DAOS-Array-ChunkSize-aligned has a clear advantage on the writer side compared to DAOS-Array, on the reader side, it is 2.5× slower than DAOS-Array. This is because in DAOS-Array, the metadata is laid out contiguously, allowing for larger chunk size (1MB) reads from individual targets, as opposed to numerous small 5KB reads from many targets. Note DAOS-POSIX is faster than DAOS-Array-aligned. This is because in ADIOS the POSIX engine relies on an optimization where metadata is prefetched on a read up to 16MB at a time, even beyond the current timestep. We have not integrated this optimization in the DAOS array case; as a result, DAOS-POSIX has a faster acquisition time for 5KB. We were able to quantify the impact of this optimization on Argonne's Aurora machine, where comparing DAOS-POSIX with and without this read-ahead optimization resulted in a ∼2.5× difference. We confirmed that, on Aurora, the difference between DAOS-POSIX and DAOS-Array was comparable to the difference observed in the Figure 12 shown here. Note that, regardless

of this small gain, the end-to-end time of DAOS-POSIX (see Fig. 13b) is still slower than Array and KV and is dominated by stabilization.

E. End-to-End Transfer Time

We aggregate the above observations from the different DAOS engine options, and present the end-to-end transfer times for the two application scenarios. The results are shown in Figures 13a and 13b. For the E3SM (56 KB) case, the end-to-end transfer time of DAOS-Array-ChunkSize-aligned is almost equal to DAOS-Array, while being 2.3× 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 2.1× 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 13a 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. Consideration of Long-running Applications

The initial set of results at 1000 timesteps helps understand tradeoffs in stabilization and acquisition times at a reasonable timescale and eliminates inefficient engines. To further evaluate these observations in the context of long-running application, we perform additional experiments where we increase the number of timestamps from 1000 to 10,000.

Overwrite DAOS Array vs. Snapshots. First, we evaluate the viability of the DAOS container snapshot mechanism as a means to provide ADIOS I/O, as opposed to extending a DAOS array object. 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 14 illustrates the comparison of stabilization times using DAOS array with and without snapshots across an increasing number of ADIOS timesteps, using only 64 writer ranks. With snapshots, writers overwrite their metadata at the same offsets, whereas without snapshots, metadata is written at new offsets for each timestep.

We observe that 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 achieves low stabilization times, even at large number of ADIOS timesteps.

Revisiting large metadata I/O for long running applications. We also investigate the changes in stabilization and acquisition costs over time for the two best-performing engines from the results in the previous section, DAOS-Array-ChunkSize-aligned and DAOS-KV-async-get. We measure and analyze two metrics – Per Timestep Time and Per Byte Transfer Cost, as we increase the number of timesteps from 1000 to 10,000. Figure 15a shows that the average per-timestep stabilization time 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 15b, 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. This is because every reader rank reads the entire metadata, while the writer rank only writes its own metadata. However, the key point here is that, unlike the container snapshot, DAOS-Array-ChunkSize-aligned is well-suited for extended ADIOS applications.

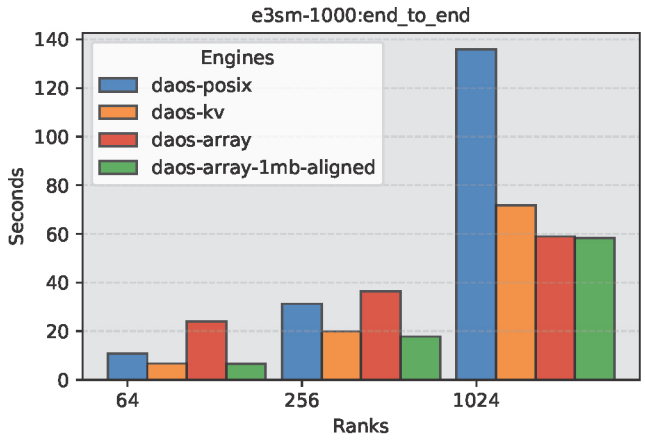
In addition to measuring the impact of a large number of ADIOS timesteps on metadata transfer performance, it is also important to measure the scaling cost associated with the number of ranks. Therefore, we measured the stabilization and acquisition time per byte. The Per Byte Time is calculated for each rank as the total data stabilized or acquired, divided by the time. Figures 16a and 16b show that the Per Byte Time increases marginally with an increasing number of ranks, and the total metadata quadrupled.

G. Storage overheads of using sparse DAOS array

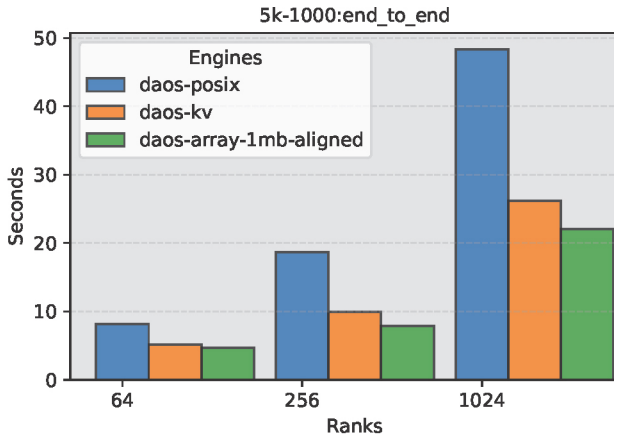
The DAOS array chunk size serves as a unit of storage distribution, not allocation. It ensures that all array elements from one chunk boundary to the next are guaranteed placement on the target. However, when writing extents of sub-chunk size, the entire chunk size is not allocated. Although additional 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, confirming that the use of sparse array does not introduce additional storage overheads.

H. WarpX Laser Wakefield Simulation

To evaluate the DAOS metadata engines with a complete application, we ran WarpX to perform a laser wakefield simulation for 1000 iterations, writing ADIOS data output every 10



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



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

Fig. 13: Comparison of End-to-End Transfer Time across DAOS Engines for different metadata sizes.

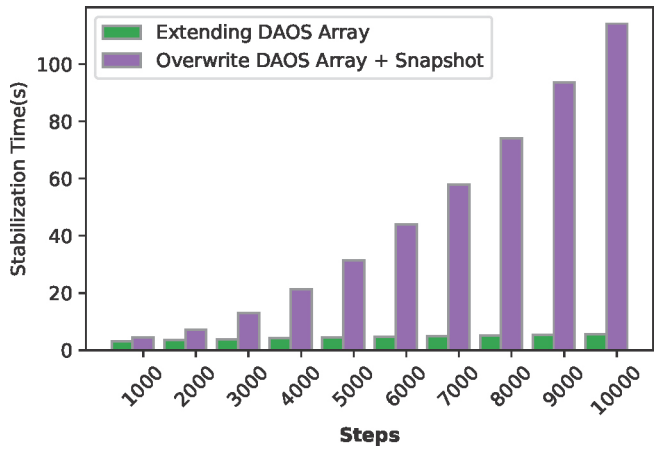
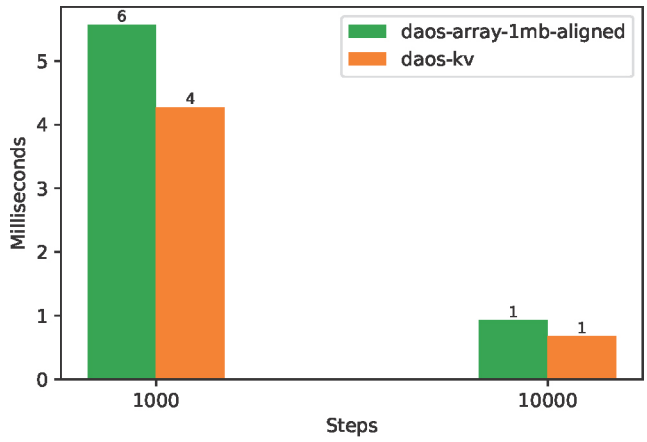
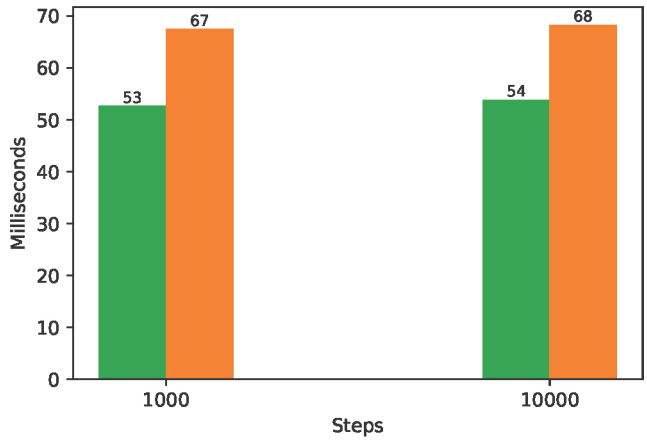


Fig. 14: Comparison of Stabilization Time with and without DAOS Snapshots using 64 Writer Ranks



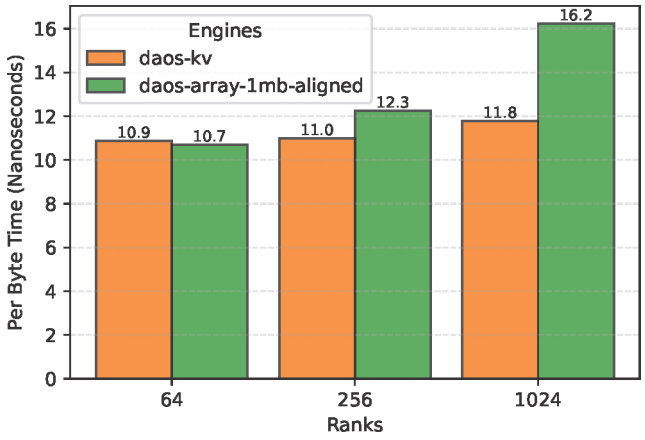
(a) Per Step Stabilization Time



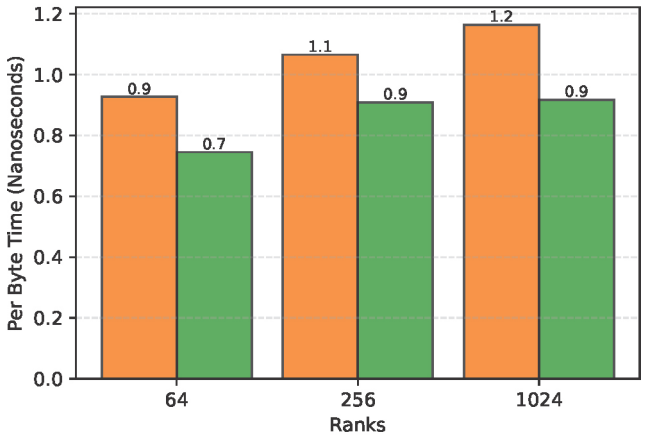
(b) Per Step Acquisition Time

Fig. 15: Evaluating the impact of increasing Steps on Stabilization and Acquisition Time per Timestep for E3SM (56KB)

iterations. The simulation setup includes a discretized grid of 128 x 128 x 256 cells, employing Adaptive Mesh Refinement (AMR) with a maximum grid size and blocking factor of 16, to optimize computational efficiency. Figure 17 shows the stabilization time as a percentage of the execution time. The metadata size of this WarpX execution is a few KBs. These results follow the similar performance trend seen in Figure 10 for 5KB stabilization. In this case, WarpX is executed on the Cambridge CS3D machine, and the absolute measurements are not directly comparable to the results from ORNL's Frontier shown in Figure 1. However, the baseline DAOS-POSIX measurement for 1024 also corresponds to ∼21% of the total I/O time. In the case of DAOS-KV-async-get and DAOS-Array-ChunkSize-aligned, this times drop down to ∼4.5%. This confirms that careful use of the DAOS interfaces provides opportunities to curtail metadata I/O costs.



(a) Per Byte Stabilization Cost



(b) Per Byte Acquisition Cost

Fig. 16: Measuring the efficiency of Stabilization and Acquisition with increasing ranks across DAOS engines using per byte cost for E3SM (56KB)

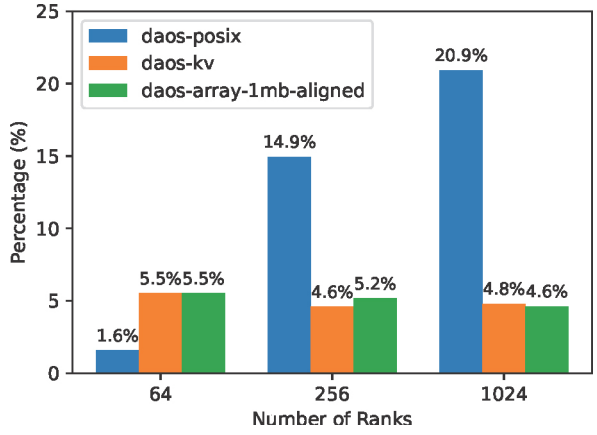


Fig. 17: WarpX: Stabilization Time as Percentage of Execution Time

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 WarpX and 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 tradeoffs on the writer and reader sides.

The new DAOS-Array-ChunkSize-aligned engine provides improved end-to-end transfer time for different metadata sizes with large number of ADIOS timesteps. We compared 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 up to 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. These new engines have been open sourced and available at [16].

VIII. Related Work

Jialin Liu et.al [17] evaluated different object stores in the context of HPC I/O. Their findings underscored the improved scalability of object stores compared to POSIX. The study employed three HDF5 Virtual Object Layer plugins specifically designed for Ceph/RADOS [18], Openstack Swift, and Intel DAOS. Their evaluation shows that the predominant I/O granularity in most object stores is the entire object, contrasting the more refined granularity observed in POSIX. In contrast, 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. also highlighted the need 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 similar requirements for ADIOS. Another study [19] delves into the optimal utilization of Optane for transporting data within HPC workflows.

A recent study [20] 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. With file-based storage, HDF5 object instantiation requires coordination amongst ranks, leading to resource-intensive I/O collectives. When integrated with DAOS, the time taken for HDF5 object creation is significantly reduced. In addition, the integration with the DAOS Key-Value interface enabled a novel HDF5 map feature. The DAOS HDF5 VOL also exhibited support for asynchronous I/O, which in turn amplified storage bandwidth utilization. The results in this

paper are complementary to these results, since our focus is on understanding the tradeoffs among the low-level DAOS interfaces, specifically in the context of metadata I/O.

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, applications like WarpX can devote 24% 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.

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) and the Cambridge Open Zettascale Laboratory. These facilities are operated by the University of Cambridge Research Computing Service (www.csd3.cam.ac.uk), using resources provided by Dell EMC and Intel, Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/T022159/1), The Excalibur Project (https://www.scd.stfc.ac.uk/Pages/Excalibur-Programme.aspx), and DiRAC (www.dirac.ac.uk). Funding from the latter two is provided by The Science and Technology Facilities Council.

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] Patrick Riley, "Berkeley Lab-Led WarpX Project Key to 2022 Gordon Bell Prize," https://cs.lbl.gov/news-media/news/2022/berkeley-lab-researchers-lead-two-gordon-bell-finalist-teams/, 2022.

[12] 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.

[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] "ADIOS2 PerfMetaData," https://github.com/ornladios/ADIOS2/blob/master/testing/adios2/performance/metadata/PerfMetaData.cpp.

[15] 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.

[16] R. S. Venkatesh and G. Eisenhauer, "DAOS Object Based Metadata Engines in ADIOS2," https://github.com/ranjansv/ADIOS2, 2024.

[17] 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.

[18] 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.

[19] 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.

[20] 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.