Data Management in the Continuum: Cross-facility Object-based Data Transfers

Jean Luca Bez*, Houjun Tang*, Chen Wang† Suren Byna ‡
*Lawrence Berkeley National Laboratory, †Nanyang Technological University, Lawrence Livermore National Laboratory,
‡The Ohio State University
*{jlbez, htang4}@lbl.gov, †chen.wang@ntu.edu.sg, ‡byna.1@osu.edu

***Abstract*—Scientific workflows are evolving from relying on a monolithic storage subsystem at a single High-Performance Computing (HPC) facility to using geographically distributed file systems, repositories, and cloud storage. As a result, storing, accessing, transferring, and managing scientific data have become highly complex and prone to performance inefficiencies. This paper delves into these challenges by exploring an optimized end-to-end interface designed to seamlessly connect various local and remote storage systems, enabling efficient data movement of objects across HPC–Cloud and HPC–HPC environments. We showcase this capability through an object-focused data management runtime system, discuss the effects of relaxed consistency semantics in distributed object scenarios, and illustrate its application in an earthquake simulation workflow. Besides reducing the amount of data by selectively transferring regions of interest, our facility-local results achieved a speedup of $45\times$ over an optimized HDF5 usage and $15\times$ over the HDF5 with caching by using the new interface in PDC-XF.**

***Index Terms*—object data transfers, cross-facility, continuum**

I. Introduction

HPC storage systems are increasingly hierarchical and complex, comprised of multiple levels of heterogeneous memories, non-volatile memory, hard disk drives, and tape archives [1]–[3]. Complicating matters further, scientific HPC applications are envisioning expanding their data storage and access models beyond a single storage subsystem located at a facility toward geographically distributed file systems, repositories, and cloud storage [4]. The evolving landscape of heterogeneous memory and storage subsystems, computing resources, and scientific applications makes managing, i.e., storing, accessing, and transferring, scientific data extremely complex and performance insufficient, impairing scientific discovery. Simple yet efficient methods of data management and seamless management through this distributed and hierarchical ecosystem are critical for the next generation of scientific applications.

Storage I/O demands in HPC facilities are historically supported by parallel file systems (PFS) such as Lustre [5], IBM Spectrum Scale (previously GPFS [6]), or BeeGFS [7]. These file systems conform to the POSIX standard, originally designed for single-CPU systems decades ago. Despite its origins, POSIX remains prevalent in large-scale facilities [2]. However, the strict consistency model of POSIX proves costly to enforce at scale [8], [9]. Furthermore, applications interfacing directly with the PFS or utilizing I/O libraries (e.g., HDF5 [10], PnetCDF [11], and ADIOS [12]) rarely require strong consistency guarantees [13]–[15]. In response, numerous newly developed PFS [16]–[18] have begun to relax consistency in exchange for I/O performance and scalability.

Most traditional HPC storage solutions are file-based, organizing and presenting data in a hierarchical structure centered around files. Object storage, however, maintains all objects in a flat namespace, providing superior scalability by reducing (or avoiding) the costs associated with path resolution, permission checking, etc. Moreover, object storage manages objects and their attributes in a self-contained unit (or container). It has gained traction in cloud systems with S3 [19], self-describing file formats such as HDF5 [10], and file-level object management systems such as Ceph [20] and DAOS [21]. Existing efforts have implemented object storage on disks, in NVRAM, in memory (e.g., PDC [22]), and in the cloud individually (e.g., S3 [19] and HSDS [23]), or attempts to expose the cloud storage as a local file system (e.g., S3FS [24], RioFS [25], YAS3FS [26], BlueSky [27], Agni [28], and Delve [29]). However, a unified end-to-end solution that manages heterogeneous hardware while optimizing data movement across facilities, i.e., HPC and cloud computing systems, and elevates data objects to first-class citizens are nonexistent.

In Fig. 1, we show a conceptual end-to-end object-focused data management scenario covering Ⓐ cloud object storage, Ⓑ data object movement between HPC and cloud, Ⓒ memory-storage movement, Ⓓ traditional HPC storage systems, Ⓔ HPC-focused data interfaces, Ⓕ transparent data transformations, Ⓖ cloud-focused storage, Ⓗ cross-facility data movement, and Ⓘ user-interfaces to interact with data.

To illustrate how an application would benefit from this geographically distributed environment, consider an earthquake

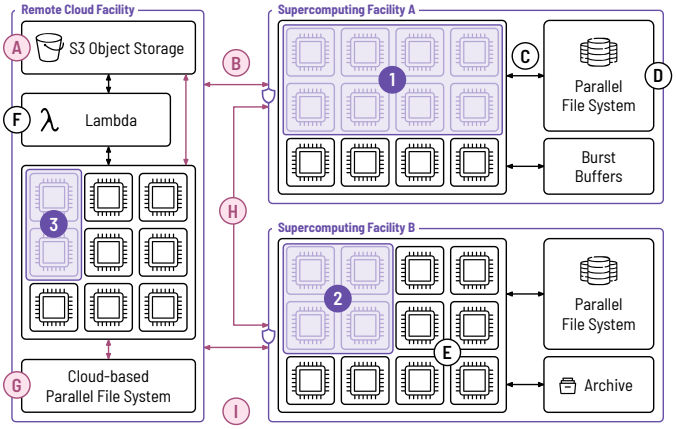


Fig. 1: End-to-end object-focused data management system for multi-facility (HPC and cloud) continuum. Our efforts focus on seamless object data transfers at runtime across facilities (in pink) to support data producers ① and consumers ② and ③ (in purple).

simulation workflow such as EQSIM [30]. ① The ground motion parallel simulation generates massive amounts of data in a single facility. ② A building infrastructure simulation or comprehensive analysis could consume the data while generating additional datasets in another cluster or facility. ③ Portions of these data variables, i.e., objects, covering regions of interest would be consumed in the cloud for visualization and analysis. However, not all data often needs to be moved across the wire. Furthermore, users are left to manage multiple isolated namespaces and use different interfaces to find and access the stored data. Many decoupled scenarios similar to our example scenario, including those mentioned in The Integrated Research Infrastructure (IRI) [4], would benefit from seamless data movement across facilities.

To realize this overarching goal of an end-to-end object-focused distributed data management system, we propose, demonstrate, and optimize an interface that enables seamless cross-facility data transfers in object-focused data management systems. Among the proposed goals in an object-focused data management continuum (Fig. 1), the Proactive Data Containers (PDC) runtime system [22] handles a few tasks in a single facility (i.e., Ⓒ, Ⓓ, Ⓔ, and Ⓕ). In this context, an *object* is a data variable such as a multi-dimensional array, similar to an HDF5 'dataset' [10], which is a logical structure that applications use for mapping data between the main memory and storage. In this paper, we describe a design and an implementation, called PDC-XF (PDC Cross-Facility), to support object-focused data movement at runtime in the cloud Ⓐ, between cloud and HPC facilities Ⓑ and between HPC facilities Ⓗ. Our contributions include:

- Design of an end-to-end object-focused data transfer layer to interface HPC and cloud storage at runtime;
- Exploration of data movement optimization options for S3-based cloud storage;
- Runtime capability for cross-HPC facility object movement while maintaining a single namespace;
- Study of relaxed consistency methods in transferring object data between storage locations;
- Lessons learned and existing challenges in realizing a live end-to-end cross-facility data movement infrastructure.

We evaluate PDC-XF with two use cases: a write-intensive general-purpose particle-in-cell HPC I/O kernel (VPIC [31]) and a state-of-the-art earthquake simulation workload (EQSIM [30]), and summarize our lessons learned.

In the remainder of the paper, we provide a brief background to PDC, an object-focused data management runtime system (§II), and describe our novel end-to-end object transfer layer supporting cross-facility storage backends (§III). We evaluate the proposed system, detailing various lessons learned.

II. Background

Given the abundance of storage solutions designed for single-facility environments, we build on an existing system that closely aligns with our goals. We extend Proactive Data Containers [22] (hence the name PDC-XF), an object-focused data management framework designed for HPC systems.

While we leverage the object-centric abstractions provided by PDC, we introduce a new interface tailored for cross-facility data transfers and the complexities of deep I/O hierarchies in modern HPC environments. In addition, the concepts and optimizations are not tied to PDC's architecture, although the mapping of PDC containers to S3-based ones is. In this section, we first provide background on the PDC abstractions integrated into our system (Section II-A), followed by a discussion of existing storage access interfaces and the motivation for our interface design (Section II-B).

A. Data Abstractions in PDC

PDC provides three abstractions for representing data, i.e., *containers*, *objects*, and *regions*. A container is a meta-object that stores a collection of related objects and provides a convenient way to access a group of data and metadata objects. An *object* is a data variable, similar to an HDF5 'dataset' [10], a logical structure that applications use to map data between the main memory and storage. Hence, in PDC, an *object* is a generic term to describe a byte stream of information. PDC allows storing and querying diverse types of metadata using *metadata objects* to describe the data objects. In scientific applications, a data object stores a large amount of information, typically in multi-dimensional arrays, such as variables in a 2D image, 3D grid, or an animation. A metadata object stores smaller data associated with the data object, which may include the object's name, time of data generation, ownership, relations to other objects, etc. Metadata objects can include attributes used to identify or enhance information about a data object. To facilitate parallel processing and flexible data placement for typical HPC applications, objects that are multi-dimensional arrays can be partitioned into smaller *regions*. An n-dimensional region is represented as an offset and the number of elements in each of the n dimensions. A region is typically the smallest operating unit in PDC. It can reside on any layer of the memory/storage hierarchy (i.e., main memory, NVRAM, disk, tape, etc.) that PDC accesses. PDC's multi-dimensional arrays can be mapped to HDF5, ADIOS, etc.

Among existing self-describing data formats, PDC container maps to an HDF5 group and an object to an HDF5 dataset or attribute. A region is similar to an HDF5 *hyperslab* described with a *dataspace*. PDC provides an object-focused interface to users, eliminating the notion of files, as seen in HDF5 and other I/O libraries. Due to similarities, applications that rely on HDF5-based abstractions could also benefit from PDC's runtime system. They could be ported to the PDC APIs or rely on a Virtual Object Layer (VOL) connector to interface with PDC without code changes.

PDC's data abstractions can also be directly mapped to an object store such as DAOS and Ceph. PDC interacts with applications at the user level without the need to install it as a file system, which differs from DAOS or Ceph. In cloud-based object stores such as S3, though conceptually, buckets could be mapped to containers and objects to objects, there are some caveats. A detailed comparison of PDC abstractions to cloud-

based object storage abstractions and the lessons learned on how to interface the two are discussed in Section III-B2.

B. Data Access Interfaces in PDC

Data access interfaces strive to provide a convenient and easy-to-use way to access data and resources. Within a single storage subsystem, scientific applications commonly rely on one of three standard interfaces: POSIX I/O, STDIO, and MPI-IO. These interfaces may be used directly or accessed through high-level I/O libraries such as HDF5 [10], ADIOS [12], or Python's NumPy stack.

POSIX remains the most widely adopted due to its portability and strong consistency guarantees. However, in HPC environments, these benefits often come at the cost of limited scalability. Similarly, the STDIO interface, while offering simplicity through built-in buffering, also suffers from scalability constraints, yet it remains surprisingly popular in HPC settings [2]. In contrast, MPI-IO provides relaxed consistency semantics and flexible APIs that align well with application-level access patterns. This design enables advanced optimizations such as collective buffering and data sieving [32]. However, these interfaces are file-based and often require performance tuning effort [33] to achieve the best performance.

The landscape of interfaces becomes even more fragmented when accessing data beyond a single storage subsystem, either within a facility or across facilities. Data transfer methods grow increasingly heterogeneous, including tools like `mput/mget` for HPSS [34], traditional commands such as `cp` and `mv`, services like Globus, as well as protocols like `scp`, `FTP`, and various REST APIs.

The original PDC system exposes an object-centric, lock-based interface [35] built atop POSIX I/O aimed at abstracting heterogeneous storage layers (e.g., SSDs and HDDs) within a single HPC facility. This interface requires users to explicitly acquire object-level locks before accessing data. While effective in localized settings, this design can be cumbersome and costly for cross-facility data management, where distributed lock coordination introduces complexity and overhead.

III. End-to-end Object Transfer Layer

Building on the concepts presented in Section II, we present the design of PDC-XF. We discuss how to interface with applications, seamlessly support multiple remote storage backends, and handle relaxed consistency.

A. An Application Cross-Facility Interface

PDC-XF exposes a new set of asynchronous *transfer APIs* that operate on *regions*. Before initiating a transfer, a user needs to create (*transfer_create*) a transfer request. The user prepares a buffer for the data transfer and specifies the access type (read or write), the associated object, and the source and destination locations. Importantly, no details about the underlying I/O device (e.g., memory or disk) are required; instead, the user specifies the logical source and destination locations. All subsequent transfer operations use this request handle. The PDC-XF client is aware of the potential transfer. The transfer begins when the user calls *transfer_start*, which initiates the transfer and returns immediately. Before returning, PDC-XF copies the user's data to its internal buffer, allowing the user to reuse the buffer.

Users can later check the status of a transfer with *transfer_status* or wait for its completion using *transfer_wait*. The transfer is globally visible after completing the *transfer_wait*, meaning all subsequent transfers can see the updated data. The combination of *transfer_create*, *transfer_start*, and *transfer_wait* offers users flexible control over buffer management and synchronization points. This design avoids the performance penalties associated with strict consistency models (discussed in Section III-C). Synchronization for propagating writes can be delayed until later (before *transfer_wait* returns) or potentially overlapped with computation.

Additionally, PDC-XF provides aggregated versions of the data transfer start and wait calls, which are helpful when applications access multiple objects in batches. Aggregated APIs simplify the process and enable further optimizations. For instance, internally, PDC-XF creates a large, aligned buffer to merge smaller I/O requests, sort transfers by their offsets to form efficient I/O patterns, and reduce the number of requests.

Essentially, the transfer interface provides an abstraction over the heterogeneous interfaces among various storage locations, as discussed in Section II-B. The runtime system hides the complexity of the *transfer* functions to move data asynchronously and transparently through the local storage hierarchy and remote storage. We rely upon this abstraction to break the barrier across facilities with PDC-XF.

B. Support for Multiple Storage Backends

We have designed an end-to-end object transfer layer in PDC-XF targeted to handle beyond traditional HPC-based parallel file systems (e.g., Lustre and GPFS) by supporting cloud-based object storage systems (e.g., S3) and to be extensible to object-based HPC file systems (e.g., DAOS) and other storage systems (e.g., NFS, tape, databases). Hence, covering the Ⓐ, Ⓑ, and Ⓗ scenarios in Fig. 1.

We show a high-level overview of our solution in Fig. 2. Our design achieves portability and performance through an internal layer called the *End-to-End Object Transfer Layer* (E2E-OTL), which maintains the common object-based abstraction as described in Section II-A but also provides a specific, independent, and optimized implementation to interact with the particularities of each targeted local or remote storage backend. Mapping object-based abstractions to multiple backends transparently and optimally is a key challenge in enabling full cross-facility support.

1) POSIX-based backend: In HPC systems, PDC stores regions of an object handled by a server as a file on POSIX-based file systems. One of the challenges of supporting POSIX-based file systems is using performance-tuning options while storing the data. HPC I/O researchers have long worked on optimizing storage performance on file systems [33]. PDC runtime system tries to apply optimizations automatically. The *transfer* functions initiate an asynchronous data transfer

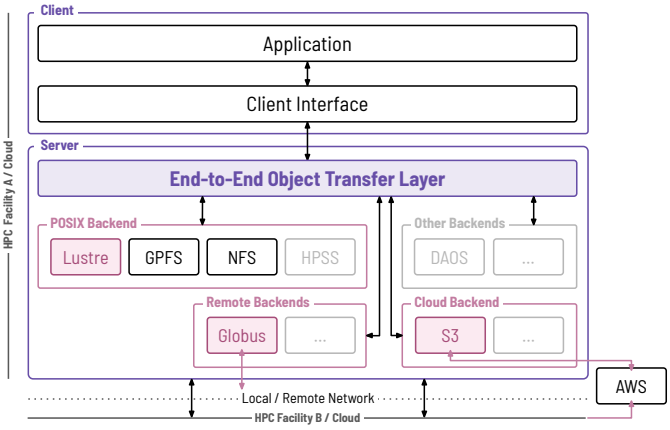


Fig. 2: End-to-end Object Transfer Layer (E2E-OTL) to support multiple backend storage interfaces. PDC-XF adds an S3 backend for the HPC-Cloud and a prototype of a remote backend atop Globus for HPC-HPC object data transfer.

between memory and storage using Mercury RPCs [36]. PDC runtime keeps the data in memory close to the application until it reaches a user-defined maximum size. Depending on the file system, the runtime chooses tuning parameters. For instance, depending on the region size, PDC automatically sets Lustre striping to maximize parallel access and avoid I/O congestion.

For cross-facility data transfers, we currently utilize the Globus service, which provides secure and high-throughput data movement across different HPC systems. To enable user-defined data migration, we provided command-line tools that allow a user to specify the object regions to be transferred. The selected data and their associated metadata are serialized into customized binary files. Our tools utilize the Globus Flows [37] feature to automate the transfer to another HPC system. Once the data transfer is complete, a user can run another command-line tool to import the data and resume their work on the new system. This approach ensures efficient data migration between HPC facilities by transferring only the requested regions and not the entire object.

2) Cloud-based backend: An application using PDC-XF can instead define a default backend (e.g., Lustre, GFPS, or NFS-based) or provide hints on a region-by-region basis to the runtime system to use cloud-based backend storage solutions such as Amazon Simple Storage Service (S3) or others that expose an API compatible to S3 (e.g., MinIO [38], Azure Blob Storage, or Google Cloud Storage).

As a comparison, besides S3, AWS also provides other storage solutions, such as the Elastic File System (EFS) based on elastic block storage and the FSx for Lustre, a fully managed parallel file system (PFS). However, both represent a trade-off between higher performance and higher cost [39] than S3. Hence, we opted to focus on S3 support for PDC-XF, also enabling other S3-compatible object storage solutions to be easily deployed in our E2E-OTL (Fig. 2).

S3 stores data as *objects* within *buckets*. An object can be viewed as a file, as well as any metadata that describes the file. A bucket is a container for objects. One could think that S3 abstractions, in principle, can directly map to HPC-focused object-storage solutions. However, there are several challenges. For instance, PDC *containers* cannot be mapped to S3 buckets because (*i*) bucket names are unique in S3 across all AWS accounts in all the AWS Regions within a partition; (*ii*) buckets can take hours to be completely deleted by AWS (i.e., it is not possible to re-use a bucket name until the S3 backend deletes it); (*iii*) there is a standard limitation of a hundred buckets per account; (*iv*) no support for hierarchical buckets (i.e., sub-buckets), though it is possible to emulate such behavior by using application-aware specific separators; and (*v*) strict naming conventions.

Furthermore, mapping PDC *objects* to S3 object would also present its challenges because (*a*) though it is possible to access parts of an object using range queries, once an object is written in S3 it can only be fully overwritten; (*b*) no direct support to search of filter metadata/data, instead one must retrieve all metadata/data first and then perform a search locally, build an external index, or combine S3 with Athena [40]. Hence, we map PDC *regions* into actual S3 objects, while containers are used solely to isolate instances.

C. Relaxed Consistency

A consistency model specifies an agreement between a user and a system, whereby the system assures that adherence to specified rules will result in consistent shared data and predictable outcomes when reading, writing, or updating. The consistency models employed by a distributed system can significantly influence its I/O performance. The overhead in maintaining a consistency model increases with the strictness of the model and the scale of the system. Broadly speaking, weaker models tend to yield better performance.

POSIX consistency stands as the prevailing consistency model in HPC storage systems. While enforcing POSIX consistency proves straightforward in a single-node environment, its scalability can pose significant challenges [13], [15], [41]. Consequently, we have observed the development of numerous new parallel file systems [16]–[18], [42] with relaxed consistency models aimed at enhancing performance and scalability.

Adopting a similar strategy, PDC-XF also diverges from adhering strictly to POSIX consistency. Given that PDC-XF is designed to accommodate different backends, each potentially governed by its consistency model, we have chosen a set of flexible transfer APIs (section III-A), allowing users to determine when and where a consistency point should occur. PDC-XF allows data transfers to be split into multiple phases—such as *transfer_start* and *transfer_wait*—enabling applications to overlap data movement with computation or other I/O tasks. Additionally, the *transfer_status* call can be used to check the progress of an ongoing transfer. Formally, PDC-XF ensures sequential consistency for programs when all *transfer* calls are properly synchronized [43]. This means users must ensure their applications does not perform any conflict accesses (concurrent data transfers to the same region and at least one is a write-type transfer). An update to a region becomes globally visible once the transfer is complete, which can be guaranteed using the *transfer_wait* call.

IV. Experimental Setup

A. Platforms

1) Perlmutter: Based on the HPE Cray Shasta platform, Perlmutter is a 79.23-PFlop heterogeneous system comprised of 3,072 CPU-only and 1,792 GPU-accelerated nodes housed at the National Energy Research Scientific Computing Center (NERSC). Perlmutter CPU-only nodes have 2 AMD EPYC 7763 (Milan) 64-core CPUs, 512GB of DDR4 memory, PCIe 4.0 NIC-CPU connection with 200G (25GB/s) bandwidth, and an HPE Slingshot 11 NIC. Perlmutter has an all-flash Lustre scratch file system with 35PB of disk space, an aggregate bandwidth of over 5TB/sec, and 4 million IOPS (4KiB random). It has 16 MDS (metadata servers), 298 I/O servers called OSSs, and 3,792 dual-ported NVMe SSDs.

2) AWS: We relied on an AWS ParallelCluster deployment for compute, scheduler, and shared PFS. All resources were allocated in *us-west-2* region using C5n instances, which feature the Intel Xeon Platinum 8000 series (Skylake-SP) processor. We deployed SLURM in a head node using a `c5n.2xlarge` (8 vCPUs, 21 GiB memory, and up to 25 Gbps network bandwidth) instance, and the compute nodes rely on `c5n.18xlarge` (72 vCPUs, 192GiB memory, and up to 100 Gbps of network bandwidth) instances.

B. Synthetic Workloads

1) VPIC-IO: This kernel is extracted from VPIC [31], a first-principles 3D electromagnetic relativistic kinetic particle-in-cell code used for simulating plasma physics phenomena. By default, each process writes a region of 8M (8×2^{20}) particles, where each particle has 8 properties, hence the total size is $n \times 8 \times 2^{20}$ where n is the number of processes.

2) BDCATS-IO: This read-intensive kernel is extracted from the BDCATS clustering system [44], which implements a DBSCAN clustering algorithm [45] for large (trillion) particle data. This application kernel reads data generated by VPIC-IO using the same I/O pattern as BDCATS.

C. EQSIM Workflow

The EQSIM (Earthquake Simulation) workflow [30], [46] aims to provide detailed exploration and practical tools for quantifying earthquake risk through regional-scale earthquake simulations and various types of data analysis. The three major components in an EQSIM workflow are 1) simulation of ground motions, 2) simulation of infrastructure response, and 3) interactive visualization.

The simulation code SW4 [47] can run on thousands of nodes with tens of thousands of GPUs, and a single simulation run can generate tens of TBs of data. The SW4 output contains the motion time-history of all grid points on the surface. The data is stored in HDF5 files and compressed by ZFP [48].

The simulation of infrastructure response uses ground motion data and applies different building models to study how much damage an earthquake can cause in different regions in an earthquake event. There are two main types of access during this process. The weak-coupling building analysis uses the motion time-history at a single location, while the strong-coupling analysis needs to access motions from a sub-region.

The data often needs to be downsampled to provide an interactive display for visual inspections and explorations. The grid spacing of large simulation runs can be just a few meters, while the domain can be over 100 km in length. Querying and retrieving the data with extreme values are also frequently used to help with data analysis. We implemented the data accesses with PDC APIs and evaluated the I/O performance in the following section.

V. Lessons Learned with PDC-XF

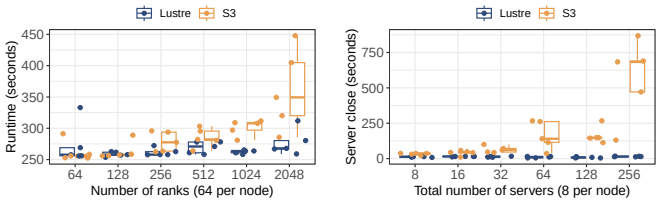
This section presents a study evaluating PDC-XF's data transfer API, encompassing support for multiple storage backends, cross-facility data transfer, and the impact of relaxed consistency semantics. We aim to share key insights and highlight data management challenges in the continuum.

A. Bridging HPC Storage and Cloud Storage

To evaluate the data transfer interface to bridge HPC and cloud storage, we ran VPIC-IO with multiple compute nodes on Perlmutter (Fig. 3) and multiple compute nodes on AWS *ParallelCluster* (Fig. 4). Each node on both platforms has 8 PDC servers, and each 64 application ranks (i.e., PDC clients). We defined five timesteps in VPIC-IO and an emulated compute time of 60s between them. We repeated each experiment 5 times to account for variability. The boxplots report the runtime distribution using the different supported storage backends described in Section III-B. All experiments have caching enabled in the PDC servers by default. "Lustre" represents access to the local shared PFS using the default system striping policy. For Perlmutter, that represents the local Lustre deployment, and for the AWS *ParallelCluster*, that represents a cloud deployment of Lustre using AWS FSx for Lustre. When computation and S3 storage are not co-located, we expect to see some overhead as the total data transferred increases, which should not be the case when computation is performed in AWS (closer to S3).

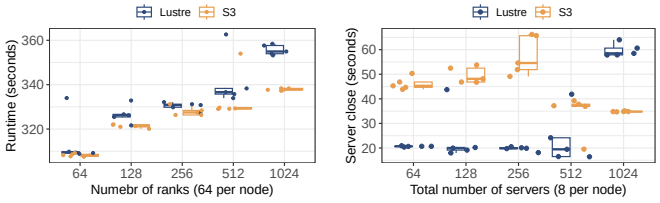
Considering the total application runtime, including the emulated compute time that could partially or totally hide the data transfer time due to the asynchronous behavior of PDC-XF data transfer, we observe a 7.59%, 4.16%, 16.80%, and 30.41% median overhead of pushing the data to S3 from 256, 512, 1024, and 2048 Perlmutter ranks (64 per node) respectively (Fig. 3). The overhead dominates the end-user's observed time as the computation and storage are not co-located, and the total data transferred increases. On the other hand, when co-locating the computation nodes with the Lustre deployment or the S3 storage in AWS, we observe up to $\approx 5\%$ improvement (Fig. 4(a)). Between the two PDC-XF backend solutions, the costs associated with S3 are much lower[1] than those of SSD-based FSx for Lustre deployment.

[1]e.g., considering prices as of August 2025, a 1 TB/month storage using (a) S3 Standard: \$23.55; (b) S3 Express One Zone: \$112.64 (4.8×); (c) FSx for Lustre (lowest available throughput of 125 MB/s): \$148.48 (6.3×).



(a) Application runtime (b) Server close time (final flush)

Fig. 3: VPIC-IO on Perlmutter.



(a) Application runtime (b) Server close time (final flush)

Fig. 4: VPIC-IO on the AWS *ParallelCluster*.

During the PDC-XF server close time, any data still in the server's cache is flushed to storage. Consequently, if the computation time cannot fully overlap the data transfer, that cost is paid at the end. In Fig. 3(b), we summarize the overhead of transferring the remaining data at server close time to S3 compared to Lustre in Perlmutter. For the same reason, we observe a median slowdown of $3.2\times$, $11.5\times$, $15.78\times$, and $46.4\times$ in the scenarios, i.e., using 32, 64, 128, and 256 PDC servers (8 per node), respectively. These results demonstrate that though the overhead at runtime might not be prohibitive, as PDC asynchronous behavior would allow overlapping the application's computation with the data transfer, the slow transfer to S3 is not fully overlapped and pushed to the server close time. On AWS, with both storage solutions closer to the compute nodes, we observed a $2.74\times$ and $1.91\times$ slowdown in the final servers' flush to S3 if compared to FSx for Lustre with 256 and 512 client ranks, respectively. However, with 1K ranks, the behavior is reversed, with S3 performing $\approx 68\%$ better than the deployed PFS. We attribute this to the scale of the deployed PFS, which is constrained by costs.

Observation #1

Data transfer between HPC and remote cloud storage can suffer significant performance overhead, worsening with larger datasets and non-co-location. While asynchronous transfers offer some overlap, the final data flush exposes latency. Hence, data locality is crucial for efficient hybrid HPC-Cloud workflows. Strategically moving regions of interest instead of the whole data could alleviate the perceived impact.

B. Optimizing for S3-based Storage

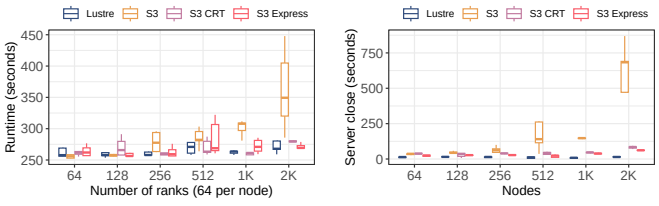
The initial results (Fig. 3 and Fig. 4) demonstrate the viability of PDC-XF by continuing to use an HPC-focused object-based runtime system such as PDC to manage data even across facilities, despite the differences in abstraction when representing objects and containers required by S3-based solutions. However, the performance impact of remote data transfer is very prominent. Though such a solution adds flexibility and connectivity across HPC and cloud facilities, those benefits are hindered by the overhead of the remote connection. We explore alternatives to speed up data transfer.

Seeking to reduce the remote data transfer costs in PDC-XF, we investigated the AWS Common Runtime (CRT) libraries to accelerate S3's throughput. For the CRT integration, the S3 SDK wraps the underlying *libcurl* REST API calls while exposing the same interface as the S3 Asynchronous Client. This approach offers improved throughput and reliability by leveraging the concept of multipart upload API and byte-range fetches, combined with enhanced connection pooling and Domain Name System (DNS) load balancing.

The multipart upload feature allows an object to be uploaded as a set of independent and possibly unordered parts, each representing a contiguous portion of the object's data. In network failures, only individual failed parts of a transfer are transparently retried. The upload can be broken into up to 10K parts, with equal sizes varying from 5MiB to 5GiB (there is no minimum size limit on the last part). PDC-XF can benefit from this feature to improve write performance to external S3-based storage by uploading multiple parts of an object in parallel. When reading objects, PDC-XF can benefit from partial reads enabled by the byte-range feature, which takes advantage of the Range HTTP header [49] to fetch only a specific range from an object instead of transferring it entirely to the client.

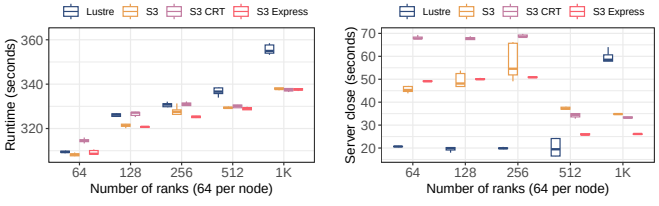
In addition to the traditional low-cost S3 cloud-based object storage, AWS has released S3 Express One Zone, a high-performance, single-zone S3 storage class built to deliver consistent, single-digit millisecond data access performance. Besides CRT, multipart upload, and byte-range requests, data in S3 Express is stored in a new directory bucket type designed for scalability. However, that comes at the expense of reduced redundancy by relying on a single availability Zone.

We integrated these optimizations in PDC-XF to account for the trade-offs between cost, performance, and redundancy and to flexibly cover the cross-facility scenarios of Fig. 1. Our evaluation with VPIC-IO under the same experimental setup is shown in Fig. 5 (Perlmutter) and Fig. 6 (AWS *ParallelCluster*).



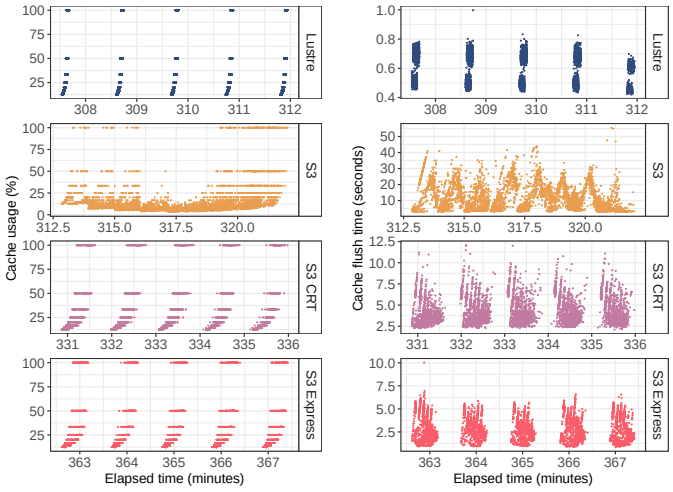
(a) Application runtime (b) Server close time (final flush)

Fig. 5: VPIC-IO on Perlmutter with different storage backends.

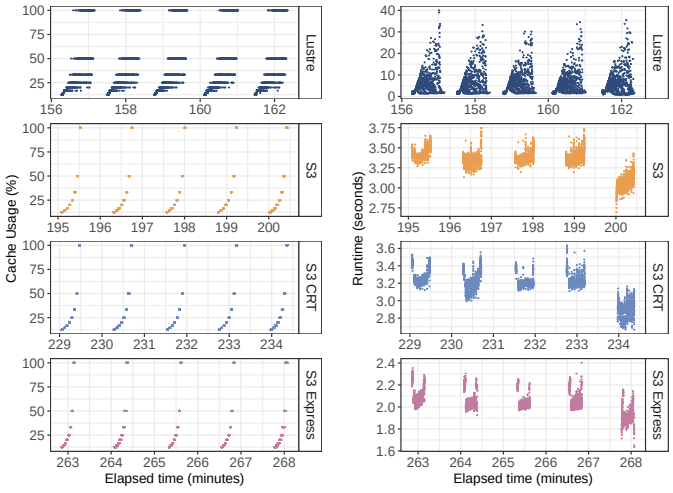


(a) Application runtime (b) Server close time (final flush)

Fig. 6: VPIC-IO on AWS *ParallelCluster* with different backends.



(a) Cache usage (Perlmutter) (b) Flush time (Perlmutter)



(c) Cache usage (AWS) (d) Flush time (AWS)

Fig. 7: Sample execution of VPIC-IO with five timesteps (arbitrary 60s of emulated computing time, smaller than the 1-2h interval in VPIC's [31] typical frequency of bursty I/O) running on Perlmutter and AWS *ParallelCluster* exercising the PDC-XF backends.

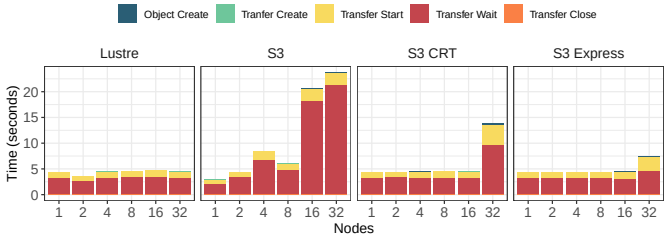


Fig. 8: PDC client time breakdown in Perlmutter with PDC-XF using the different backends. Results show the best of five repetitions.

In Perlmutter, with 1K ranks (over 16 nodes), the CRT optimizations improved the throughput by 15.1% compared to the baseline S3. When switching to S3 Express instead, we observed a median performance increase of 11.9%. However, as the scale increases, with 2K ranks, the optimized PDC-XF integration to S3 yields a median increase in throughput of 19.9% with CRT and 22.9% with Express.

PDC-XF does not maintain multiple replicas of the same

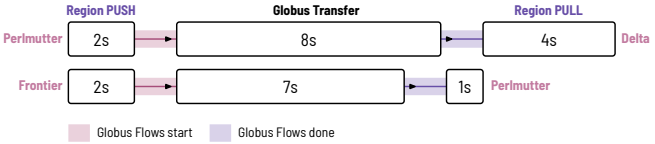


Fig. 9: Time breakdown for transferring data across HPC facilities using PDC-XF's cross-facility prototype.

data on cloud storage systems. Regarding cache coherence, PDC-XF assumes a write once, read many (known as WORM [13]) pattern, and does not expect updates. Fig. 7 depicts a sample execution of PDC-XF cache usage ratio over time (a) and (c) and the time taken by each request during the asynchronous data transfer (b) and (d) between the write phases while running VPIC-IO on Perlmutter and AWS *ParallelCluster*, respectively. Notice that the axes are not the same in all plots. These numbers help support the overall improvements observed when using S3 Express, with a median flush time of 2.96s, i.e., 2.92× improvement over the Standard S3 interface, but still 4.84× slower than accessing the local Lustre. On the other hand, in the AWS cluster, the median flush time of 2.02s with S3 Express was ≈ 25% faster than flushing to Lustre and ≈ 65% faster than the Standard S3, proving to be a cost-effective alternative.

Fig. 8 illustrates the PDC-XF transfer API's client-side time breakdown across various backend storage interfaces. The "transfer wait time" reflects the cumulative duration where data transfer was not fully overlapped with computation over five timesteps (not explicitly shown, but each of the four computation stages lasted 60 seconds). To fully leverage the object data transfer interface, it is crucial to consider the overhead from each storage backend and its geographical location. Despite these factors, PDC-XF effectively demonstrates the feasibility and trade-offs of such a seamless object interface. In Section V-E, we present a realistic scenario where transferring only relevant data regions to the cloud significantly mitigates the impact of large data movements.

While PDC-XF can interact with S3 from within an HPC facility, the other way around (i.e., cloud-initiated data transfer) is not supported yet due to security constraints and mismatching authentication and authorization policies across large-scale computing facilities. This might change in the foreseeable future as IRI [4] efforts advance.

Observation #2

Leveraging features such as directory-based buckets, multipart uploads, and byte-range fetches can substantially reduce the overhead of remote S3 accesses. When designing data movement strategies in the continuum, one must account for the overhead introduced by each storage backend and its geographical location. PDC-XF demonstrates the feasibility of such a seamless object interface and the trade-offs in data management.

C. Towards Cross HPC Facilities

We demonstrate PDC-XF's cross HPC facilities data migration capability using the EQSIM dataset with a 4634 × 200 × 200 region selection, which amounts to ≈ 1.4 GB data. We ran two tests transferring data among three HPC centers: Frontier@OLCF, Perlmutter@NERSC, and Delta@NCSA.

Fig. 9 shows the time breakdown for two cross-facility data migrations. A single binary data file is created from the origin and transferred to the target before loading it to PDC-XF servers' memory for immediate use. The *pdc_region_push* time is similar in both Perlmutter and Frontier as both systems have fast Lustre-based PFS, while the *region_pull* time is longer in the Delta system because of the slower file system there. The Globus transfer time measures the data transfer time and is similar in both cases, as the same amount of data is transferred. Globus Flows start is the time after PDC-XF submits a data transfer job to Globus and before the data transfer starts. It can vary from ≈ 20 seconds to a couple of minutes and requires Globus authentication. Globus Flows done is the time after the data transfer and before the user runs *region_pull* to import data into its local PDC-XF runtime. In comparison, transferring the entire 280 GB ZFP-compressed file through Globus takes over 14 minutes from Frontier to Perlmutter.

Observation #3

PDC-XF offers command-line tools for efficient data migration between HPC facilities. Users can quickly resume work by transferring only necessary data regions, rather than entire objects. PDC-XF relies on Globus, with plans to explore alternatives, such as the NERSC Superfacility API.

D. Towards Relaxed Consistency

As discussed in Section III-A, the transfer APIs offer a flexible way for users to manage the consistency point. We illustrate this by simulating two consistency models and utilizing VPIC-IO and BDCATS-IO to showcase their performance disparities. The two models, namely *strong consistency* and *session consistency*, are defined in [13] and are widely employed in the realm of HPC storage systems.

In our scenario, BDCATS-IO will not start until VPIC-IO has been completed, indicating that session consistency is sufficient for both. Employing a stronger-than-needed consistency would add unnecessary overhead, as we will demonstrate.

To simulate strong consistency, we implement each write and read operation using a combination of three transfer APIs: *create*, *start*, and *wait* (the common *transfer_* prefix is omitted for simplicity). These APIs are treated as a single atomic unit to ensure that updates are consistently the most up-to-date. For session consistency, we postpone the consistency point until the time of closure. When executing a write operation, we invoke *create*, followed by *start*, and we defer the *wait* call until the object is closed. Additionally, since there is no requirement to treat *start* and *wait* as an atomic unit, we can perform a batched start/wait call, employing *start_all* and *wait_all* to process all writes simultaneously. Reads typically follow the implementation used for strong consistency. However, in the case of BDCATS-IO, we once again utilize a batched start/wait call to process all variables concurrently.

When conducting large I/O with VPIC-IO and BDCATS-IO, we did not observe distinct performance differences among different consistency models. Therefore, we present the results of writing/reading a small number of particles: $8,192$ particles per timestamp for 100 timestamps, resulting in each rank accessing only 64KB per timestamp (400MB per node).

In Fig. 10, we show the total transfer time and its breakdown. The breakdown includes only the two most expensive calls, excluding the *create* and *close* calls, as their costs are negligible. In both VPIC-IO and BDCATS-IO, session consistency facilitates the use of a batched start call, aggregating I/O operations, and initiating bulk data transfer in a single request. This significantly reduces the overall RPC cost of the start operation. The same optimization also applies to wait calls. However, since the start cost is higher in strong consistency, by the time the wait call is executed, many data transfers may already be completed (as the server pulls the data in the background), resulting in a shorter measured wait time. Overall, session consistency reduced the end-to-end execution time by half, resulting in a $2\times$ bandwidth improvement compared to using strong consistency.

Observation #4

Employing a consistency model that is stronger than necessary can result in decreased performance. PDC-XF allows users to determine the most suitable consistency point for their specific application requirements instead of statically enforcing consistency points.

E. EQSIM Workflow Case Study

We use an M7 Hayward Fault earthquake simulation dataset from the EQSIM workflow and compare the performance of using PDC-XF and the existing HDF5-based data accesses in EQSIM. HDF5 uses ZFP lossy compression with the accuracy mode and has a size of 293GB, while its uncompressed size is 25TB. Three main HDF5 datasets store the velocity values of the fault normal, fault parallel, and vertical directions, and each is a 4D array, with the first dimension as time and the other three as the 3D domain. The grid has a 6.25m spacing in a 120×80km domain and contains 90s of motion data with the timestep interval of ≈ 0.019s. This dataset only includes the surface motions, so the z (last) dimension has a size of 1. HDF5 chunking is used with a chunk size of $(400, 600, 400, 1)$ in each dimension with a total size of each dataset of $(4634, 19201, 12801, 1)$. HDF5 implementation in EQSIM is highly tuned after numerous experiments. We imported this dataset to the PDC-XF system by mapping the HDF5 file as a container, the three datasets as objects, and the HDF5 chunks as regions. The same compression option is

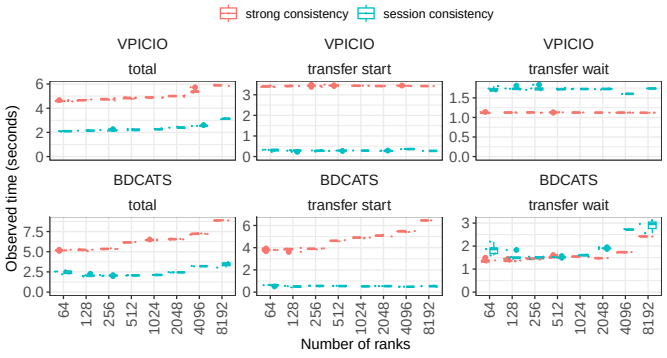


Fig. 10: Breakdown of transfer time under two consistency semantics. Batched start/wait calls substantially decrease the overall time.

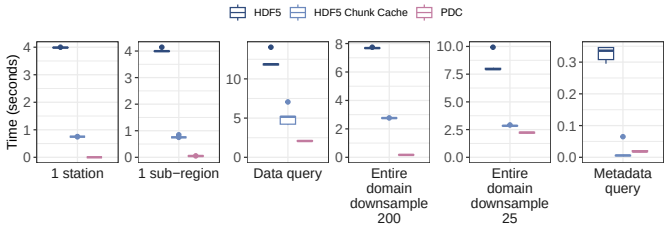


Fig. 11: Performance comparison for different EQSIM workflow access patterns on 128 Perlmutter nodes.

used. This direct mapping allows an almost seamless transition to accessing the data with the PDC-XF APIs.

We evaluate six access patterns commonly used by scientists on this large dataset: 1) get the motion time-history of one location (grid point) for weakly-coupled building response analysis, 2) get the motions of one bounding box selection (e.g., 200×200m, 32×32 grid) for strongly-coupled building response analysis, 3) find the grid points with values larger than a specific threshold for identifying extreme values, 4) get the motions of the entire domain with a downsampling factor of 200 (every 1.25km) for inspecting the motions on an interactive map 5) get the motions of the entire domain with a downsampling factor of 25 for generating a ground motion animation, and 6) get the latitude and longitude range of the grid points in a bounding box selection for mapping the data. Patterns 1 and 2 are accessed by one MPI rank, and Patterns 3 to 6 are collective accesses across all ranks. The data selectivities of Patterns 1 to 5 are 4e-7%, 4e-4%, 0.1%, 0.5%, 4%, and Pattern 6 retrieves all the (*lat*,*lon*) metadata. The current EQSIM workflow processes these patterns by reading, transforming, and filtering data from a PFS-based HDF5 file.

In Fig. 11, we compare performance (time) to execute these six access patterns. "HDF5" is the default HDF5 parallel data access, and "HDF5 Chunk Cache" enables HDF5 chunk caching with maximum cache size set to 4GB, and PDC-XF is our implementation of accessing the same data. All the runs are executed on 128 Perlmutter nodes, with PDC-XF using 1 server per node and 8 application clients per node, and each configuration is repeated 5 times. For "HDF5 Chunk Cache" and PDC-XF configurations, the data is pre-loaded in memory: HDF5 caches data on the client side while PDC-XF caches data on the server side. The pre-load time is similar in both options and close to the HDF5 time in Pattern 5. All the accesses use one of the three variables in the dataset, which is stored on Lustre with 128 OSTs for both HDF5 and PDC-XF. From Fig. 11, PDC-XF offers significantly better performance than HDF5 and mostly better performance than "HDF5 Chunk Cache". It achieves the highest speedup when the requested data is a small subset, with a median speedup of 45× over HDF5 and 15× over Chunk Cache.

We also demonstrate a cross-facility scenario in which EQSIM's simulation data is generated in a supercomputer (i.e., Perlmutter), and regions of interest are pushed to object-based cloud storage (i.e., S3) for subsequent analysis. EQSIM's data is imported into PDC-XF in Perlmutter, emulating the application's data generation while hinting to PDC-XF which regions would be needed for the analysis. PDC-XF uses the data transfer interfaces and backends discussed in Sections III-A and III-B to move only the necessary data to S3.

For the analysis at AWS, we restarted PDC-XF over 4 compute nodes, with 1 server per node and 8 application ranks per node. Table I presents the median runtime for 10 repetitions of each analysis pattern while reading regions previously flushed to S3 with the Express One Zone backend.

Observation #5

We evaluated the effectiveness of PDC-XF's object-based data management and optimizations with a real-world use case, under both HPC-only and HPC-Cloud hybrid scenarios. Compared with the existing HDF5 approach, PDC-XF offers over 10× speedup over HDF5 in data access operations. Additionally, by supporting HPC-Cloud data migration, users can seamlessly switch their data analysis or visualizations to the cloud.

VI. The Road Ahead

The Integrated Research Infrastructure (IRI) report [4] states that a "seamless integration of computing, networking, instruments, and experimental facilities is required to support these emerging workloads." Furthermore, the task force raised a set of principles for such an effort to succeed. We emphasize *flexibility* of exposing resources as simple services, *performance* without needing to know the data locations, *scalability*, *transparency* via seamless transfers, *extensibility* of data-related access across storage technologies. The current data management landscape has significant shortcomings in achieving this vision. Integrating multiple HPC and cloud facilities for scientific discovery is a complex challenge. Still, our insights pave the road to further R&D on closing some of these gaps by relying on object-based abstractions to seamlessly and proactively move data within and across facilities.

In this effort, we explored and demonstrated addressing some of these challenges. PDC can be seen as an end-to-end object-focused data management runtime system and service that performs efficient data movement in heterogeneous memory and storage backends. PDC-XF further pushes the boundaries seamlessly in integrating HPC and cloud systems. Bottlenecks in data movement arise from limitations in network infrastructure due to disparate performance and technologies, particularly when transferring massive datasets common in scientific research. The scale of data and poor-performing access patterns [2], [3] have a more pronounced impact in distributed scenarios. Hence, data locality is crucial for efficient hybrid HPC-Cloud workflows. Strategically moving regions of interest, as demonstrated by PDC-XF, rather than the entire dataset, could mitigate the perceived impact. Furthermore, providing applications with the flexibility to define consistency models allows a runtime system, such as PDC-XF, to balance performance requirements.

TABLE I: EQSIM analysis pattern runtime in AWS.

Analysis Pattern	Median Time (s)
1 Station	0.0999
1 Sub-Region	0.2060
Entire Domain Downsample 200	1.1580
Entire Domain Downsample 25	73.1927
Metadata Query	0.0008

We also acknowledge the existing policy-related issues concerning security in such scenarios, particularly those related to authorization and authentication in supercomputing facilities that are being explored by NERSC Superfacility API [50] and AWS Identity and Access Management (IAM). Lastly, when cloud components are connected to traditional research facilities, the cost implications of data movement and storage tiers also present significant hurdles, requiring careful optimization strategies. Overcoming these technical, economic, security, and management challenges is crucial for realizing an integrated HPC-cloud research infrastructure.

VII. Related Work

Solutions such as S3FS [24], RioFS [25], and YAS3FS [26] rely on mounting S3 buckets via FUSE to provide seamless access to S3-based cloud storage solutions, with performance implications. BlueSky [27] has similar goals but operates as a network file system (NFS) backed by cloud storage. Agni [28] and Delve [29] represent initial attempts to overcome performance shortcomings of previous solutions. Skyplane [51], [52] focuses on optimizing bulk data transfers by intelligently routing entire objects between cloud object stores. In contrast, PDC-XF is a runtime system built for fine-grained data movement and management of scientific data objects. PDC-XF works with application-defined logical objects, enabling the transfer of specific, often multi-dimensional, sub-regions.

There is no comprehensive, end-to-end solution that effectively optimizes data movement across diverse computing environments, including HPC and cloud systems, while treating scientific data objects as first-class entities. We also note the lack of generic, object-focused runtime data management systems. For example, HEPnOS [53] is a domain-specific (High Energy Physics) in-memory storage system. While PDC [22], [54] shows promise for HPC workloads, it is limited to single systems. Efforts like Zambeze [55], an automated distributed framework, and the Interplanetary File System (IPFS) [56], a distributed content-addressed file system, complement our work but differ in approach. Zambeze uses a lazy-transfer model with file URIs, extending Globus Flows for cross-facility. IPFS focuses on storing and serving files within a global peer-to-peer network. In contrast, PDC-XF proactively moves data using object-based abstractions and regions.

Interfaces: Cloud-based object storage solutions typically rely on REST [57] APIs atop the HTTP protocol, such as AWS S3 [19] API. In contrast, HPC-based solutions rely on custom (e.g., DAOS) or traditional POSIX-like (e.g., Lustre, GPFS) interfaces. DAOS [21] provides C-based APIs that applications need to build against (with Python and Go bindings) to interface with the object storage: the multi-level key-array API (native object interface with locality feature), key-value API, and array API (implements a one-dimensional array of fixed-size elements). Ceph [20] object storage interface is built on top of *librados*, providing two interfaces: S3-compatible and Swift-compatible. Both share a namespace, which enables writing data to a Ceph Storage Cluster using one API and then retrieving that data with the other API. It is possible to use different language-specific APIs that rely on the RESTful API to interface with the storage backends. Globus [58] offers a fire-and-forget data transfer relying solely on file abstractions.

libCOS [59] provides native object storage support through the unified MPI I/O interface by intercepting calls and converting them into S3 operations. It also supports asynchronous and collective operations using multi-part upload. Additional YAML files with associated metadata are generated during the writing phase, including a local per-rank and a global map. However, this approach adds overhead in metadata management, connection, and data type conversions, whereas PDC-XF keeps metadata in memory, avoiding type conversions, and relies on S3 CRT and Express One Zone to reduce overheads.

Consistency: Most widely-deployed parallel file systems (e.g., Lustre [60], GPFS [6], and BeeGFS [7]) support POSIX consistency semantics. However, as several studies [13], [15] have demonstrated, POSIX consistency is expensive to enforce at scales and HPC applications rarely need it. Many new file systems adopt relaxed consistency models in pursuit of improved performance and scalability. For instance, UnifyFS [16], SymphonyFS [17], and BurstFS [61] feature commit consistency semantics, whereas NFS [62] and Gfarm/BB [18] offer session consistency semantics.

To avoid the prohibitive cost of enforcing POSIX consistency across both HPC and cloud storage, PDC-XF provides flexible transfer APIs. This allows users to select their desired consistency model, ensuring optimal application performance without relying on a single, static approach.

VIII. Conclusion

We explored solutions for seamless and efficient data movement beyond heterogeneous memory and storage devices by supporting multiple HPC and cloud facilities. We demonstrate this capability, performance, and the trade-offs of our design and solution atop an HPC-based object-focused data management runtime system. We studied the scalability and performance of this runtime system using different storage backends and moving data between them using benchmarks and write-intensive and read-intensive I/O kernels. We achieved a median speedup of $45\times$ over a highly optimized HDF5 implementation while accessing data from a state-of-the-art earthquake simulation workflow. We also discussed the impact of relaxed consistency semantics. Our design and lessons learned in managing scientific data in the continuum by relying on object-focused abstractions and transfers provide insights. Future work will focus on enhancing support for DAOS and non-traditional HPC storage (e.g., databases), assessing weaker consistency models with various real-world HPC applications, and optimizing data movement and in-flight feature extraction to enable complex scientific discovery.

Acknowledgment

This effort was supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research (ASCR) under contract number DE-AC02-05CH11231 with LBNL, under an LBNL subcontract to OSU (GR130493), under Contract DE-AC52-07NA27344, and was supported by the LLNL-LDRD Program under Project No. 23-ERD-053.

REFERENCES

[1] A. Brinkmann, K. Mohror, and W. Yu, "Challenges and Opportunities of User-Level File Systems for HPC," *Dagstuhl Seminar 17202*, 2017.

[2] J. L. Bez, A. M. Karimi, A. K. Paul, B. Xie, S. Byna, P. Carns, S. Oral, F. Wang, and J. Hanley, "Access Patterns and Performance Behaviors of Multi-Layer Supercomputer I/O Subsystems under Production Load," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '22, (New York, NY, USA), p. 43–55, Association for Computing Machinery, 2022.

[3] J. L. Bez, S. Byna, and S. Ibrahim, "I/O Access Patterns in HPC Applications: A 360-Degree Survey," *ACM Computing Surveys*, vol. 56, sep 2023.

[4] ASCR IRI Task Force, "Toward a Seamless Integration of Computing, Experimental, and Observational Science Facilities: A Blueprint to Accelerate Discovery," tech. rep., USDOE Office of Science, 3 2021.

[5] P. J. Braam *et al.*, "Lustre Storage Architecture," 2004.

[6] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *1st USENIX Conf. on File and Storage Technologies*, FAST '02, (USA), p. 19–es, USENIX, 2002.

[7] "BeeGFS Parallel Cluster File System." https://www.beegfs.io, 2020.

[8] D. Hildebrand, A. Nisar, and R. Haskin, "pNFS, POSIX, and MPI-IO: A Tale of Three Semantics," in *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pp. 32–36, ACM, 2009.

[9] J. Stender, B. Kolbeck, F. Hupfeld, E. Cesario, E. Focht, M. Hess, J. Malo, and J. Martí, "Striping without Sacrifices: Maintaining POSIX Semantics in a Parallel File System," in *USENIX Workshop on Large-Scale Computing*, LASCO'08, (USA), USENIX, 2008.

[10] The HDF Group, "Hierarchical Data Format, version 5," 1997-2022. https://www.hdfgroup.org/HDF5/.

[11] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel NetCDF: A High-Performance Scientific I/O Interface," in *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, (New York, NY, USA), p. 39, Association for Computing Machinery, 2003.

[12] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchouam, L. Jay, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "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.

[13] C. Wang, K. Mohror, and M. Snir, "File System Semantics Requirements of HPC Applications," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '21, (New York, NY, USA), p. 19–30, ACM, 2021.

[14] S. Yellapragada, C. Wang, and M. Snir, "Verifying IO Synchronization from MPI Traces," in *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*, pp. 41–46, IEEE, 2021.

[15] M. Vilayannur, S. Lang, R. Ross, R. Klundt, L. Ward, *et al.*, "Extending the POSIX I/O Interface: A Parallel File System Perspective," tech. rep., Argonne National Lab.(ANL), Argonne, IL (United States), 2008.

[16] M. J. Brim, A. T. Moody, S.-H. Lim, R. Miller, S. Boehm, C. Stanavige, K. M. Mohror, and S. Oral, "UnifyFS: A User-level Shared File System for Unified Access to Distributed Local Storage," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 290–300, 2023.

[17] S. Oral, S. S. Vazhkudai, F. Wang, C. Zimmer, C. Brumgard, J. Hanley, G. Markomanolis, R. Miller, D. Leverman, S. Atchley, and V. V. Larrea, "End-to-end I/O portfolio for the Summit supercomputing ecosystem," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[18] O. Tatebe, S. Moriwake, and Y. Oyama, "Gfarm/BB—Gfarm File System for Node-Local Burst Buffer," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 61–71, 2020.

[19] A. W. Services, "Amazon Simple Storage Service (Amazon S3)," 2020.

[20] G. Borges, S. Crosby, and L. Boland, "CephFS: A New Generation Storage Platform for Australian High Energy Physics," *Journal of Physics: Conference Series*, vol. 898, p. 062015, oct 2017.

[21] Z. Liang, J. Lombardi, M. Chaarawi, and M. Hennecke, "DAOS: A Scale-Out High Performance Storage Stack for Storage Class Memory," in *Supercomputing Frontiers* (D. K. Panda, ed.), (Cham), pp. 40–54, Springer International Publishing, 2020.

[22] H. Tang, S. Byna, F. Tessier, T. Wang, B. Dong, J. Mu, Q. Koziol, J. Soumagne, V. Vishwanath, J. Liu, and R. Warren, "Toward Scalable and Asynchronous Object-Centric Data Management for HPC," in *CCGrid'18*, p. 113–122, IEEE Press, 2018.

[23] H. Group, "HSDS (Highly Scalable Data Service) - REST-based service for HDF5 data," 2023.

[24] R. Rizun, "S3FS – FUSE-based file system backed by Amazon S3." https://github.com/s3fs-fuse/s3fs-fuse, 2010.

[25] Skoobe, "RioFS." https://github.com/skoobe/riofs, 2010.

[26] D. Poccia, "YAS3FS (Yet Another S3-backed File System)." https://github.com/danilop/yas3fs, 2012.

[27] M. Vrable, S. Savage, and G. M. Voelker, "BlueSky: a Cloud-backed File System for the Enterprise," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, (USA), p. 19, USENIX, 2012.

[28] K. Lillaney, V. Tarasov, D. Pease, and R. Burns, "Agni: An Efficient Dual-access File System over Object Storage," in *ACM Symposium on Cloud Computing*, SoCC '19, (New York, NY, USA), p. 390–402, ACM, 2019.

[29] M.-A. Vef, R. Steiner, R. Salkhordeh, J. Steinkamp, F. Vennetier, J.-F. Smigielski, and A. Brinkmann, "DelveFS - An Event-Driven Semantic File System for Object Stores," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 35–46, 2020.

[30] D. McCallen, A. Petersson, A. Rodgers, A. Pitarka, M. Miah, F. Petrone, B. Sjogreen, N. Abrahamson, and H. Tang, "EQSIM—A multidisciplinary Framework for Fault-to-structure Earthquake Simulations on Exascale Computers part I: Computational models and workflow," *Earthquake Spectra*, vol. 37, no. 2, pp. 707–735, 2021.

[31] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan, "Ultrahigh Performance Three-dimensional Electromagnetic Relativistic Kinetic Plasma Simulations," *Physics of Plasmas*, vol. 15, no. 5, 2008.

[32] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," in *Proceedings. Frontiers '99. Seventh Symposium on the Frontiers of Massively Parallel Computation*, pp. 182–189, 1999.

[33] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol, *et al.*, "Taming Parallel I/O Complexity with Auto-Tuning," in *SC'13: Proceedings of the International Conf. on High Performance Computing, Networking, Storage and Analysis*, SC '13, (Denver, Colorado), pp. 1–12, ACM, 2013.

[34] R. W. Watson and R. A. Coyne, "The parallel I/O architecture of the high-performance storage system (HPSS)," in *Proceedings of IEEE 14th Symposium on Mass Storage Systems*, pp. 27–44, IEEE, 1995.

[35] J. Mu, J. Soumagne, H. Tang, S. Byna, Q. Koziol, and R. Warren, "A Transparent Server-Managed Object Storage System for HPC," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 477–481, 2018.

[36] J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. Ross, "Mercury: Enabling Remote Procedure Call for High-Performance Computing," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–8, 2013.

[37] R. Chard, J. Pruyne, K. McKee, J. Bryan, B. Raumann, R. Ananthakrishnan, K. Chard, and I. T. Foster, "Globus Automation Services: Research Process Automation across the Space–time Continuum," *Future Generation Computer Systems*, vol. 142, pp. 393–409, 2023.

[38] I. MinIO, "MinIO." https://github.com/minio/minio, 2014.

[39] R. B. Roy, T. Patel, and D. Tiwari, "Characterizing and Mitigating the I/O Scalability Challenges for Serverless Applications," in *IEEE Int. Symposium on Workload Characterization (IISWC)*, pp. 74–86, 2021.

[40] Amazon Web Services, "AWS Whitepaper: Big Data Analytics Options on AWS," tech. rep., Amazon, 2024.

[41] D. Kimpe and R. Ross, "Storage Models: Past, Present, and Future," *High Performance Parallel I/O*, pp. 335–345, 2014.

[42] C. Wang, *Parallel File System with Tunable Consistency*. PhD thesis, University of Illinois at Urbana-Champaign, 2022.

[43] C. Wang, K. Mohror, and M. Snir, "Formal Definitions and Performance Comparison of Consistency Models for Parallel File Systems," *IEEE TPDS*, vol. 35, no. 6, pp. 937–951, 2024.

[44] M. M. A. Patwary, S. Byna, N. R. Satish, N. Sundaram, Z. Lukić, V. Roytershteyn, M. J. Anderson, Y. Yao, Prabhat, and P. Dubey, "BD-CATS: Big Data Clustering at Trillion Particle Scale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[45] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, p. 226–231, AAAI Press, 1996.

[46] D. McCallen, F. Petrone, M. Miah, A. Pitarka, A. Rodgers, and N. Abrahamson, "EQSIM—A multidisciplinary Framework for Fault-to-structure Earthquake Simulations on Exascale Computers, part II: Regional simulations of building response," *Earthquake Spectra*, vol. 37, no. 2, pp. 736–761, 2021.

[47] N. A. Petersson, B. Sjogreen, H. Tang, and R. Pankajakshan, "SW4, version 3.0," Sept. 2023.

[48] P. Lindstrom, "Fixed-rate Compressed Floating-point Arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[49] IETF HTTP Working Group, *Hypertext Transfer Protocol (HTTP/1.1): Range Requests*, 6 2014.

[50] B. Enders, D. Bard, C. Snavely, L. Gerhardt, J. Lee, B. Totzke, K. Antypas, S. Byna, R. Cheema, S. Cholia, M. Day, A. Gaur, A. Greiner, T. Groves, M. Kiran, Q. Koziol, K. Rowland, C. Samuel, A. Selvarajan, A. Sim, D. Skinner, R. Thomas, and G. Torok, "Cross-facility science with the Superfacility Project at LBNL," in *2020 IEEE/ACM 2nd Annual Workshop on Extreme-scale Experiment-in-the-Loop Computing (XLOOP)*, pp. 1–7, 2020.

[51] P. Jain, S. Kumar, S. Wooders, S. G. Patil, J. E. Gonzalez, and I. Stoica, "Skyplane: Optimizing transfer cost and throughput using Cloud-Aware overlays," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, (Boston, MA), pp. 1375–1389, USENIX Association, Apr. 2023.

[52] Z. Yang, Z. Wu, M. Luo, W.-L. Chiang, R. Bhardwaj, W. Kwon, S. Zhuang, F. S. Luan, G. Mittal, S. Shenker, and I. Stoica, "SkyPilot: An intercloud broker for sky computing," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, (Boston, MA), pp. 437–455, USENIX Association, Apr. 2023.

[53] S. Ali, S. Calvez, P. Carns, M. Dorier, P. Ding, J. Kowalkowski, R. Latham, A. Norman, M. Paterno, R. Ross, S. Sehrish, S. Snyder, and J. Soumagne, "HEPnOS: a Specialized Data Service for High Energy Physics Analysis," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, (St. Petersburg, FL, USA), pp. 637–646, IEEE, 2023.

[54] H. Tang, S. Byna, S. Bailey, Z. Lukic, J. Liu, Q. Koziol, and B. Dong, "Tuning Object-Centric Data Management Systems for Large Scale Scientific Applications," in *IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 103–112, 2019.

[55] T. J. Skluzacek, R. Souza, M. Coletti, F. Suter, and R. Ferreira da Silva, "Towards Cross-Facility Workflows Orchestration through Distributed Automation," in *Practice and Experience in Advanced Research Computing 2024: Human Powered Computing*, PEARC '24, (New York, NY, USA), Association for Computing Machinery, 2024.

[56] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and Evaluation of IPFS: A Storage Layer for the Decentralized Web," in *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, (New York, NY, USA), p. 739–752, Association for Computing Machinery, 2022.

[57] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*. Univ. of California, Irvine, 2000.

[58] I. Foster, "Globus Online: Accelerating and Democratizing Science through Cloud-Based Services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011.

[59] D. Araújo De Medeiros, S. Markidis, and I. Bo Peng, "LibCOS: Enabling Converged HPC and Cloud Data Stores with MPI," in *Int. Conf. on HPC in Asia-Pacific Region*, pp. 106–116, 2023.

[60] SUN, "High-Performance Storage Architecture and Scalable Cluster File System," tech. rep., Sun Microsystems, Inc, 2007.

[61] T. Wang *et al.*, "BurstFS: A Distributed Burst Buffer File System for Scientific Applications," in *The Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2015.

[62] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network File System (NFS) Version 4 Protocol," 2003.