Bridging Grid and HPC Computing for Data-Intensive Science: Scaling dCache for Modern Workflows

Tigran Mkrtchyan[1,5], Anastasiia Chub[1], Christopher Green[2], Karen Hoyos[1], Dmitry Litvintsev[2], Marina Sahakyan[1], and Darren Starr[3,4]

[1] Deutsches Elektronen-Synchrotron DESY, Notkestrasse 85, 22607 Hamburg, Germany
[2] Fermi National Accelerator Laboratory (FNAL), PO Box 500, Batavia, 60510-5011 IL, USA
[3] The University of Oslo, Boks 1072, Blindern, NO-0316, Norway
[4] Nordic e-Infrastructure Collaboration (NeIC), c/o NordForsk, Stensberggata 25, Oslo, NO-0170, Norway
[5] FernUniversität in Hagen, Universitätsstraße 47, 58097 Hagen, Germany

Abstract. The increasing integration of high-performance computing (HPC) resources into data-intensive scientific workflows places new demands on storage systems traditionally developed for grid and distributed computing environments. dCache, a mature, exascale storage system jointly developed by Deutsches Elektronen-Synchrotron (DESY), Fermi National Accelerator Laboratory (FNAL), and the Nordic e-Infrastructure Collaboration (NeIC), has evolved to support a broad range of scientific communities beyond its origins in High-Energy Physics, including astrophysics, Photon science, and AI training. These communities increasingly rely on HPC systems for large-scale data analysis, exposing scalability and performance challenges at the metadata and data access layers.
This paper presents recent development efforts to make dCache more HPC-friendly in interdisciplinary scientific environments. By aligning dCache's architecture and development practices more closely with HPC requirements, we enable transparent access to shared data infrastructures from HPC clusters while preserving dCache's strengths in data management, federation, and long-term preservation. We focus on optimizing metadata access to improve scalability and reduce latency under highly parallel workloads, as well as on substantial enhancements to dCache's NFSv4.1/pNFS implementation. These include improved pNFS layout handling, read delegation, and zero-copy data paths to reduce CPU overhead and memory copies, resulting in significantly improved I/O performance for HPC applications. We evaluate these enhancements using representative HPC workloads at DESY and FNAL, assessing their impact on throughput, latency, and overall system scalability. The benchmark results demonstrate that the proposed changes can significantly improve application performance, depending on the workflow in use.

Keywords: HPC · HTC · Distributed systems · Storage Systems · dCache · Data access.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

1 Introduction

The dCache [20] project started in 2000 as a collaboration between Deutsches Elektron-Synchrotron (DESY) and Fermi National Accelerator Laboratory (FNAL). The mission then was to develop a common storage software for the two laboratories that combined commodity heterogeneous disk servers as a caching layer in front of tape storage. In contrast to earlier approaches, the software would provide an experiment-agnostic solution, allowing different groups of particle physicists to use a shared infrastructure. A POSIX-compliant [1] namespace and clean separation between that namespace and the physical location of the file's data meant that various operator interventions were possible without requiring downtime and that a failure of storage nodes resulted in unavailability of the data stored exclusively on these nodes without affecting access to the bulk of the data. On top of being a distributed storage, dCache provides various access and authentication protocols that are used by different communities; it supports third-party copy to move data between sites; the data stored in dCache can be accessed by several standard and HEP-specific protocols to satisfy the demands of the different scientific community; dCache can run in heterogeneous environments, giving sites flexibility in hardware and OS selection. Owing to its scalable architecture, dCache can run on a single node or on hundreds of nodes, allowing sites to grow as needed. All this made the dCache software popular within the Worldwide Large Hadron Collider Computing Grid (WLCG)[6]. As of March 2025, combined, these dCache storage instances accounted for more than 50% of the WLCG storage capacity excluding CERN[7].

Though initially, dCache was developed for High Energy Physics Experiments (HEP), sites increasingly started to use dCache to support other communities with different requirements. For example, DESY facilities and services now support photon science, biology, future accelerator R&D, and other areas. The EuXFEL instance at DESY combines more than 400 physical hosts, providing 120PB of total capacity. The XFEL and FLASH accelerators write the telemetry data directly to dCache, effectively integrating dCache as an essential part of the accelerator DAQ system and, therefore, relying on dCache I/O bandwidth and low latency.

Despite its maturity, dCache maintains agility and flexibility that enable it to adapt to evolving technologies and end-user requirements. The shift from grid-centric computing models, often referred to as high-throughput computing (HTC), toward tightly coupled high-performance computing (HPC) and data-analysis platforms has exposed limitations in the dCache storage system, driving the adoption of more HPC-oriented design and optimization strategies.

In this paper, we present the evolution of the dCache distributed storage system to support modern scientific workloads and AI-based analysis. We identify two dominant bottlenecks: (1) metadata serialization caused by strict POSIX semantics under highly parallel file creation, and (2) extra network overhead in-

[6] https://wlcg.cern.ch
[7] The numbers are taken from WLCG operation dashboards.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

duced by repeated open–read–close access patterns in machine learning pipelines. To address these challenges, we introduce configurable directory consistency modes that relax parent metadata updates while preserving application-visible correctness, and we implement an adaptive read delegation mechanism combined with NFSv4.1 FLEX_FILES layouts and zero-copy data transfers.

The remainder of this paper is structured as follows. Section 2 reviews related work. Sections 3 and 4 introduce the dCache distributed storage system and its NFSv4.1/pNFS implementation. Section 5 presents the proposed metadata consistency modes and adaptive read-delegation mechanism, followed by a performance evaluation using standard benchmarks. Finally, Section 6 summarizes the main findings and outlines directions for future research.

2 Related works

The filesystem semantics required by POSIX specifications create the so-called hot inode issue. Every *create*, *link*, *mkdir*, *rename*, and *unlink* operation in a directory must update the directory's mtime and i_version attributes. There is a significant development effort in the filesystem developers' community to eliminate the performance bottleneck introduced by i_version attribute handling from a trivial spin-lock to CAS-based lock-free atomic increment [18,7]. IBM developers are making significant efforts to improve how GPFS handles the hot-inode issue [27,16].

Wang et al. [28] presented a detailed analysis of the I/O requirements of 17 representative HPC applications. They have demonstrated that all tested applications, except one, will benefit from relaxed POSIX semantics to improve data access. Although metadata handling was not part of the study, they collected data on which metadata operations HPC applications typically use. Only two applications perform directory listing or use file and directory access time attributes. Therefore, consistency in parent directory attributes between them is not important. The evaluations performed by Oeste et al. [22] indicate that semantics enforced by the POSIX I/O standard might not be necessary for many HPC applications and significantly contributed to performance bottlenecks in the parallel file system.

Recent research explores the use of databases for filesystem metadata management. HopsFS [21] replaces HDFS's in-memory metadata service with a distributed NewSQL-based alternative, achieving 16–37× higher throughput but lacking hot inode mitigation. In comparison to dCache, HopsFS positions itself as a drop-in replacement for HDFS metadata server, requires a proprietary MySQL NDB distribution, and has a limited number of installations[8].

Chen et al. [5] performed a deep comparison of different versions of the NFS protocols. The researchers concluded that NFSv4.1 is more verbose than version 3. However, with proper implementation of read- and write-delegations and

[8] We couldn't find any publicly available information on the number of HopsFS deployments.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

caching, NFSv4.1 can be 172× faster. Moreover, NFSv4.1 allows multiple requests to be sent in parallel; thus, multiple I/O requests can be performed concurrently, which is a significant benefit, especially on high-latency networks. The research focuses on the NFSv4.1 protocol without the pNFS extension. Our work confirms the findings reported by researchers and extends them to distributed deployments and real-world applications.

Although machine learning (ML) and artificial intelligence (AI) training workloads are becoming a common use case for HPC resources [24], they exhibit different I/O patterns from classic HPC jobs [17] and, at scale, introduce unique performance challenges. This work focuses on addressing these observed I/O characteristics within the dCache storage system.

Previous work on dCache has emphasized scalability, federation, and long-term data management for data-intensive science [20]. This paper contributes by focusing on HPC and AI workloads, demonstrating how modern pNFS features, metadata consistency tuning, and zero-copy data paths can transform a mature HTC-oriented storage system into an efficient backend for contemporary HPC and AI platforms.

3 dCache design overview

dCache is a storage system designed to store and retrieve large volumes of data distributed across heterogeneous server nodes within a single virtual filesystem tree, supporting a variety of standard access methods. dCache is written in the Java programming language and makes extensive use of the Java ecosystem, such as the built-in profiler, concurrency model, high-performance I/O libraries, and a single binary package for various operating systems[9]. By design, dCache follows a scalable distributed storage architecture [11]; it strictly separates file metadata from the physical location of data. File names, attributes, and directory trees are managed in an internal database and are exposed through a namespace component. Moreover, dCache can accommodate multiple copies of a single file, dynamically add or remove locations, and use external storage such as S3 or hierarchical storage management (HSM), typically using magnetic tape. The system scales horizontally with the number of data servers: adding new nodes increases both storage capacity and aggregate data bandwidth.

A simplified dCache architecture is visualized in Fig. 1. There are four main components: *doors*, the user entry points that implement the supported access protocols, each door implementing a specific protocol; *pools*, the data servers that store the data and use plugin-based architecture to implement all supported protocols; *pool manager*, the component that is responsible for the data placement, i.e., selecting which pool should be used for a given transfer; and the *namespace*, a component that stores file metadata, exposes a hierarchical file system view and enforces POSIX semantics on file system operations. All

[9] As a Java application, dCache can run on any OS that supports the desired Java version. Today, all known dCache deployments run on various flavors of Linux-based systems.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

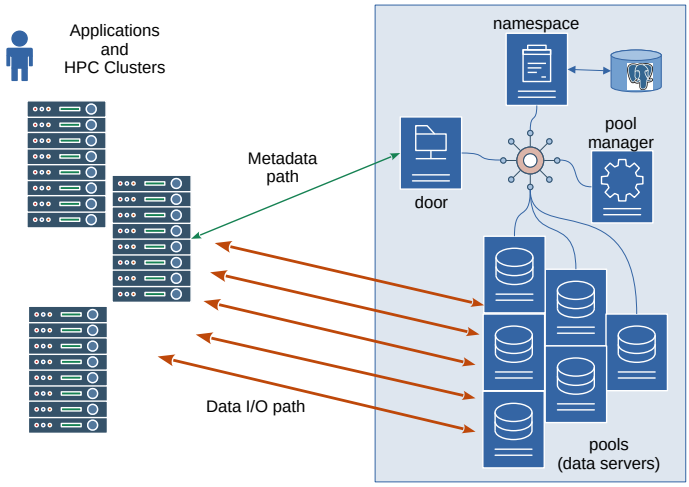

Fig. 1: dCache overview design. The minimal setup comprises four components: a protocol-specific door, a pool manager, a namespace, and data servers. The components communicate by exchanging messages.

components communicate by sending or receiving messages over a TCP/IP network. Multiple redundant copies of dCache components can be started within a single deployment to ensure high availability and load balancing. The topology auto-discovery and coordination between multiple instances of dCache services are based on Apache Zookeeper [2] - an open-source, highly reliable coordination service for distributed applications. Data resilience is provided by redundant file replicas and/or a tape copy.

To scale with the number of clients, dCache redirects the client to the server selected for the given transfer by PoolManager. First, the client contacts a door and requests access to a file. The door queries the metadata server for file attributes and location within the system. After verifying the access rights, the door requests a pool from PoolManager that matches the specified transfer. As mentioned earlier, PoolManager selects a pool based on file availability and returns a pool that will service the request. To reduce the number of requests required to access a file, each door maintains its own copy of the PoolManager, which replicates the central PoolManager's state. The door contacts the selected pool to authorize the transfer and then passes access-protocol-specific redirect information to the client. Finally, the client connects to the pool and performs the I/O requests. The details of the flow may differ depending on the access protocol. In general, every access to a file in dCache requires only two internal message exchanges.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

It is worth mentioning that the described flow has a built-in self-consistency correction. If a Pool receives a request for a file replica that it doesn't host, the metadata server will be notified of the inconsistency, and the incorrect record pointing to that data server will be removed. Then, the selection process will restart.

The namespace/metadata service of dCache is built on top of a PostgreSQL [26] database. An integrated automatic database schema migration applies filesystem structural changes if needed. The dCache's namespace relies heavily on database ACID semantics for filesystem consistency and integrity: any file system change is performed atomically within a single transaction; transaction isolation ensures that end users and applications always see a consistent filesystem state. Almost all database records are retrieved by keys, and query execution times are in sub-millisecond ranges. Table 1 shows query execution times reported by PostgreSQL 17.2, running on Dell PowerEdge R660, with 64 Cores and 256GB RAM. The dCache namespace hosts 200.000.000 filesystem objects.

Operation	**Mean query execution time, ms**
lookup file by name	0.0073
read file attributes	0.0800
update attributes	0.0145
create file	0.1515
list directory	7.0444
delete file	0.0111

Table 1: Mean database query execution times in milliseconds as reported by PostgreSQL statistics on the dCache system with 200.000.000 filesystem objects. The average number of entries per directory for the list operation is 286, max 326381.

4 NFSv4.1/pNFS implementation

To provide access to distributed data without requiring a driver or application changes, dCache can be accessed as a locally mounted filesystem via NFSv4.1/pNFS protocol [25], using standard clients such as the Linux kernel. In versions 4.0 and earlier of the NFS protocol, all data access goes through a single node, limiting overall performance. With the pNFS extension, NFSv4.1 has split the metadata and data access paths and has, therefore, become a practical way to access large-scale storage systems [14] in a distributed fashion. Fig. 2 demonstrates a pNFS-enabled distributed server architecture. Although NFSv4.1/pNFS is not the most popular data access protocol in HPC environments, it is currently the only open standard that provides POSIX access to distributed data. Moreover, recent developments in the Linux NFS kernel client have demonstrated that

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

pNFS can deliver the performance required by HPC workloads[6,13,23]. For this reason, dCache supports NFSv4.1/pNFS, with an emphasis on pNFS. To our knowledge, dCache was the first publicly available storage system to use pNFS in production [8,10]. We not only play the role of early adopters but also actively participate in the IETF[10] protocol definition process and in regular cross-vendor compatibility testing events.

The NFSv4.1/pNFS protocol is seamlessly integrated into the dCache architecture. The pNFS metadata server (MDS) is implemented as yet another dCache door, while data servers (DS) are implemented in dCache pools (see Section 3). The protocol specifies only the interactions among the client, MDS, and DSes; the so-called control path between MDS and DS is unspecified and may be defined by implementations.

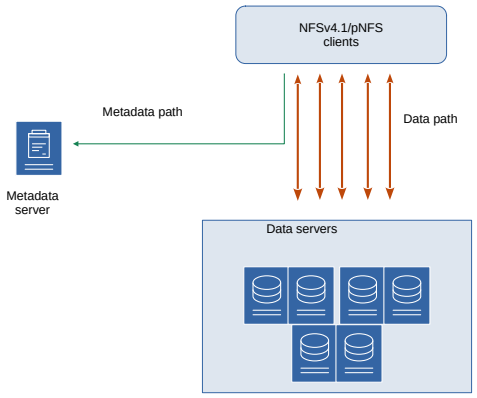

Fig. 2: NFSv4.1/pNFS distributed architecture. The storage bandwidth grows with the number of data servers.

The pNFS standard specifies four layout types that define how data is distributed across multiple data servers and how clients access those servers. Initially, dCache supported only LAYOUT4_NFSV4_1_FILES, which uses a limited subset of the NFSv4.1 protocol to access DSes (only establishment of the NFSv4.1 session, READ, and WRITE operations are allowed). With the introduction of the LAYOUT4_FLEX_FILES type in RFC8435 [12], dCache has updated its implementation to take advantage of the new layout type's benefits, such as error reporting and pNFS I/O statistics provided by clients. RFC8435 introduces the concepts of tightly- and weakly-coupled DSes, where, in the tightly-coupled model, communication between MDS and DS uses a specifically designed protocol, and NFSv3 [4] as a data path protocol. As of now, dCache is the only pNFS server implementation with a tightly-coupled DS model and NFSv4.1 as the data path protocol. This not only enriches the NFSv4.1/pNFS landscape but

[10] https://datatracker.ietf.org/wg/nfsv4/about/

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

also proves that the protocol is not tied to a single vendor implementation. The NFS protocol handling in dCache is provided as a separate LGPLv2.0-licensed [9] library and can be used by others to implement their own pNFS clusters.

The recent changes to the NFS implementation in dCache version 11.2 have introduced zero-copy data transfers, in which dCache pool nodes can send data to clients using the copy_file_range system call, thereby avoiding additional copying from disk to the dCache internal I/O buffer. This optimization has improved I/O bandwidth by 20% and reduced CPU load on data servers during performance benchmarking.

5 Improving HPC workloads

As dCache has started to serve HPC workloads, we have identified limitations in the current dCache implementation. Firstly, some applications that use checkpoints experienced a hot-inode issue (or hot directory), in which multiple clients concurrently attempt to create files in a single directory. Secondly, due to the large number of small reads, dCache's internal communication significantly contributed to file open latency.

5.1 Metadata service optimization

As mentioned in Section 3, dCache uses PostgreSQL as a backend for its metadata service. Although PostgreSQL can handle more than 100k database INSERTs per second, UPDATEs of a single record by multiple concurrent transactions are serialized. PostgreSQL (and many other databases) uses *Multiversion Concurrency Control* (MVCC) for transaction isolation, which creates multiple versions of the records and controls the visibility by transaction id: the row version is visible to a transaction if it was committed before the transaction started, i.e., has a lower transaction id. On UPDATE, if the target row is being updated by another concurrent transaction, the updater will wait for the first transaction to commit or roll back. This behavior is guaranteed by row-level locking, which is implicitly applied to any DELETE or UPDATE operations. Such low-level locking doesn't apply to INSERT; therefore, INSERT-only workloads provide better throughput and are limited only by hardware constraints.

This behavior of the database degrades throughput when multiple clients create files in a single directory, because each create operation requires updating the parent directory's mtime. Fig. 3 shows the file creation rates using the mdtest [15] benchmark running two different workloads. In one case, files are created in a single directory shared by all tasks, whereas in the other, files are created in a directory per task.

With respect to the filesystem object creation procedure, we identify two distinct suboperations with different responsibilities. The first one is responsible for the filesystem structure, i.e., allocating and inserting a new inode into the filesystem tree. Filesystem integrity must be guaranteed at this stage: no orphan inodes or dangling file names should exist when the operation completes or fails.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

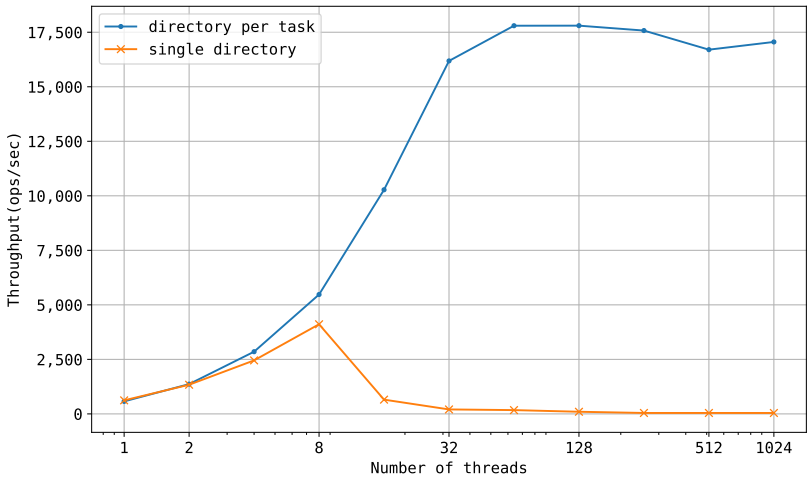

Fig. 3: File creation per second over NFS. The creation rate drops with a growing number of concurrent clients when all jobs create files in the same directory.

The second one is responsible for updating the parent directory's attributes to reflect the filesystem change so that clients can discover directory modification events, typically to check the client's local cache validity and, if needed, re-fetch the directory listing. Typically, those caches have a validity period, and clients will detect directory changes after some time.[11] The consistency offered by both suboperations serves two fundamentally different functions: one ensures filesystem integrity, and the other makes the filesystem modification (actually, only attribute updates) discoverable by the clients. While the first function, from a database perspective, requires strong consistency, the second can tolerate an inconsistent state, i.e., rely on eventual consistency. Moreover, the filesystem object creation operations are not idempotent; a second attempt fails with the *File exists* error. However, updating the parent directory attributes is an idempotent operation and can be applied multiple times. When multiple concurrent create or remove operations are processed, the highest (latest) value of the parent directory time is desired.

To improve file creation rate and match the application needs dCache metadata server has introduced three parent directory attribute consistency guarantees:

strong Creation of a filesystem object will update the parent directory's mtime, ctime, nlink and generation attributes right away. This behavior strictly mimics POSIX constraints.

[11] For Linux NFS client, the default cache validity is 30 seconds, which can be changed by the *acdirmin* mount option.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

weak Creating a filesystem object will eventually update the parent directory's mtime, ctime, nlink, and generation attributes. The clients may observe outdated parent directory attributes.

soft The behavior is similar to weak consistency with a key difference: directory attribute reads will reflect pending updates. In other words, it ensures eventual consistency for directory attributes at the database level while providing readers with the most current information. This is achieved by introducing additional latency through an extra lookup to account for those updates.

This functionality was introduced in dCache version 9.2.0 and has demonstrated the ability to create up to 17500 files per second, matching the file-creation rate achieved with a dedicated directory per task. The benchmark results are shown in Fig. 4.

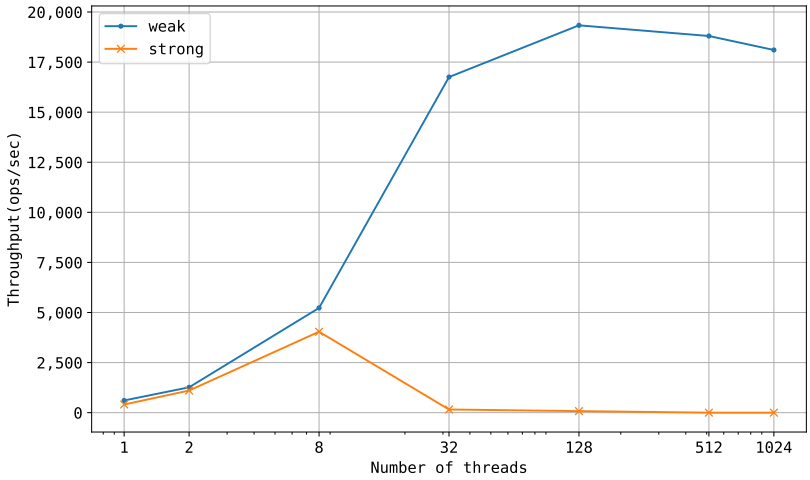

Fig. 4: File creation rates depending on consistency level enabled in the metadata server.

5.2 NFS read delegations

Though the pNFS protocol provides native access to distributed data, it introduces overhead. For each open-read-close on the application level, multiple NFS requests must be exchanged between the client and the server. The client will send the first OPEN request to obtain the file handle, then issue a LAYOUTGET operation to determine which DSes should be accessed, and subsequently send READ requests to the appropriate DS. Upon close, the client will issue

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

the CLOSE operation to release server resources. For applications that open a file once and stream large amounts of data, this overhead is negligible. However, many modern data analysis and ML-training frameworks repeatedly open and close the same file, often reading only a small portion or a single frame of a dataset each time. In a loop processing thousands of frames, this results in thousands of OPEN and CLOSE operations.

From the analysis application developer's perspective, this pattern is natural and idiomatic. From the storage system's perspective, it leads to pathological behavior: the OPEN-CLOSE overhead dominates the I/O.

NFS delegations allow a server to decouple application-level open-close cycles from protocol-level open-close cycles. The server delegates file access to the client. When a client holds a delegation, it may perform open-close operations locally without contacting the server. For read delegations, the server assures the client that no other client will modify the file during the delegation period. As a result, the client can cache data and suppress repetitive OPEN and CLOSE operations. If a conflicting OPEN request is sent to the server, then the server will recall the delegation, forcing the client to fall back to a standard OPEN-CLOSE cycle.

One of the challenges is deciding when to grant a delegation. Granting delegations too aggressively can increase recall traffic and impose additional load on the server, whereas being too conservative limits performance gains. dCache therefore implements an adaptive, per-client heuristic based on observed access patterns. For each client, the server maintains two queues: an eviction queue and an active queue. When a file is accessed for the first time, it is placed in the eviction queue. A second access within the specified time window moves it to the active queue, and the server then delegates the file. Files in the active queue continue to benefit from delegation. If a file in the active queue is not accessed for a configurable idle period, it is demoted back to the eviction queue. Both queues are capacity-limited and managed using least-recently-used (LRU) eviction policies. This heuristic captures the intuition that files accessed repeatedly within a short time window are good candidates for delegation, while one-off accesses should not consume delegation resources.

Despite the availability of delegations in the NFSv4.1 protocol, dCache pNFS implementation based on LAYOUT4_NFSV4_1_FILES layout type could not use it; the I/O on DSes is associated with OPEN-stateid, while the client is free to send OPEN-, DELEGATION-, or LOCK-stateids. The introduction of the LAYOUT4_FLEX_FILES layout, standardized in 2018, provided a turning point. Flex_files layout explicitly specifies which stateid should be used for I/O. As a consequence, dCache dropped support for LAYOUT4_NFSV4_1_FILES and older clients, such as those based on RHEL6, that could not support the required semantics.

To evaluate the impact of read delegations, we compared dCache version 11.0.x, which did not include the adaptive delegation mechanism, with dCache 11.1.x and later. The test workload with standard FIO [3] simulated a scientific analysis, by reading 128 KB frames from an HDF5 file over NFS. Every frame reads as an open-close cycle. The test runs in a loop and reads at a different

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

offset on each iteration. In total, 4 GB of data out of 22 GB file was read by the benchmark. The benchmark execution command is shown in listing 1.1. The test run replicates the real application by running a deep-learning pipeline based on the widely used CLAM [19] framework for whole-slide image (WSI) classification. In dCache 11.0.x, the benchmark ran 168 seconds, corresponding to a per-frame access time of about 5 milliseconds. With read delegations enabled in dCache 11.1.x, the total processing time dropped to under 12 seconds, with access of approximately 0.37 milliseconds per frame. The results are summarized in Table 2.

```
fio --name open-in-loop \
      --size=128k  --bs=128k --loops=32000 \
      --rw=randread --filename=test-data.h5
```

Listing 1.1: Opening file in the loop with FIO.

	Without delegation	**With read delegation**
Number of RPC requests	128023	32006
Benchmark execution time, seconds	168	12
Average per frame read time, ms	5	0.37

Table 2: Number of OPEN-CLOSE cycles on the server, mean number of bytes read per cycle, and total number of bytes read per job with and without read-delegation.

6 Conclusions and Future work

dCache is a mature, production-grade storage system that has successfully supported data-intensive scientific workflows for over two decades. Initially developed for HEP experiments, dCache has evolved into a universal solution supporting various scientific domains, including astrophysics, biomedical research, and life sciences. Its architecture integrates distributed disk storage with an HSM, enabling seamless migration between disk and tape while ensuring efficient data access, high availability, and fault tolerance. With the shift from grid clusters toward HPC resources and non-HEP workloads, design limitations in previous versions of dCache were exposed by new clients' data access patterns.

In this paper, we have demonstrated that these challenges can be addressed without abandoning standardized interfaces or dCache's core design principles. By analyzing representative HPC and AI access patterns, we identified key bottlenecks and introduced targeted optimizations at both the metadata and data-access layers. On the metadata side, we showed that selectively relaxing parent directory attribute consistency significantly improves file-creation scalability under highly parallel workloads while preserving filesystem integrity. This approach

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

aligns with prior observations that many HPC and AI applications do not depend on strict POSIX metadata semantics.

At the data-access level, we presented substantial enhancements to dCache's NFSv4.1/pNFS implementation. We showed that poor performance observed by users was primarily due to excessive OPEN and CLOSE operations triggered by inefficiencies in popular ML and AI training frameworks and applications. By introducing an adaptive delegation heuristic and leveraging modern pNFS layouts, dCache eliminates this overhead and delivers order-of-magnitude performance improvements. Additionally, the switch to zero-copy data paths on the data server side reduces CPU utilization and improves throughput, further increasing the efficiency of HPC applications.

Together, these results demonstrate that NFSv4.1/pNFS, when combined with modern client implementations and carefully engineered server-side optimizations, is a viable and effective solution for HPC and AI storage. By relying on a standardized, POSIX-compliant protocol, dCache enables transparent access from heterogeneous environments without proprietary clients, simplifying operations while maintaining competitive performance.

A limitation of the current work is the absence of application-level I/O studies for HEP workloads, which we identify as an important direction for future research to better understand data access patterns and optimize end-to-end performance.

Future work will focus on further reducing inter-component messaging latency, which remains a critical factor for small-file-access–based applications. In parallel, delegation heuristics will be extended to support mixed read/write workloads. A key direction for future research is a deeper analysis of I/O requirements for ML and AI applications on HPC systems. Insights from such analyses will serve as a foundation for further development, ensuring that dCache continues to evolve to meet the data management needs of data-intensive science.

Acknowledgments. Tigran Mkrtchyan, Anastasiia Chub, Karen Hoyos, and Marina Sahakyan acknowledge support from DESY (Hamburg, Germany), a member of the Helmholtz Association HGF.

This work is supported by FermiForward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

We also thank Jennifer Ahrens from the Center for Molecular Neurobiology (ZMNH), University Medical Center Hamburg-Eppendorf (UKE), for running analysis jobs during the performance optimization period.

Code availability

The source code of dCache is hosted at https://github.com/dCache/dcache. The NFSv4.1/pNFS protocol components are available at https://github.com/dCache/nfs4j

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

References

1. IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7. IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008) pp. 1–3951 (2018). https://doi.org/10.1109/IEEESTD.2018.8277153
2. Apache Software Foundation: Apache ZooKeeper. https://zookeeper.apache.org (2026)
3. Axboe, J.: Flexible I/O Tester (2022), https://github.com/axboe/fio
4. Callaghan, B., Pawlowski, B., Staubach, P.: RFC1813: NFS Version 3 Protocol Specification (1995)
5. Chen, M., Hildebrand, D., Kuenning, G., Shankaranarayana, S., Singh, B., Zadok, E.: Newer is sometimes better: An evaluation of nfsv4.1. In: Lin, B., Xu, J.J., Sengupta, S., Shah, D. (eds.) Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Portland, OR, USA, June 15-19, 2015. pp. 165–176. ACM (2015). https://doi.org/10.1145/2745844.2745845
6. Christofferson, F.: Optimizing AI and HPC Workloads with Parallel NFS 4.2: A Practical Overview. Tech. rep., Hammerspace, Inc (2024), https://hammerspace.com/optimizing-ai-and-hpc-workloads-with-parallel-nfs-4-2-a-practical-overview/
7. Edge, J.: Improving i_version. https://lwn.net/Articles/937247/ (2026), online; posted on July 5, 2023
8. Elmsheuser, J., Fuhrmann, P., Kemp, Y., Mkrtchyan, T., Ozerov, D., Stadie, H.: LHC Data Analysis Using NFSv4.1 (pNFS): A Detailed Evaluation. Journal of Physics: Conference Series **331**(5), 052010 (dec 2011). https://doi.org/10.1088/1742-6596/331/5/052010
9. Free Software Foundation, Inc.: GNU LIBRARY GENERAL PUBLIC LICENSE (1991), https://www.gnu.org/licenses/old-licenses/lgpl-2.0.txt, version 2, June 1991
10. Fuhrmann, P., Gasthuber, M., Kemp, Y., Ozerov, D.: Experience with hep analysis on mounted filesystems. Journal of Physics: Conference Series **396**(4), 042020 (Dec 2012). https://doi.org/10.1088/1742-6596/396/4/042020
11. Gibson, G.A., Nagle, D.F., Amiri, K., Butler, J., Chang, F.W., Gobioff, H., Hardin, C., Riedel, E., Rochberg, D., Zelenka, J.: A cost-effective, high-bandwidth storage architecture. SIGOPS Oper. Syst. Rev. **32**(5), 92–103 (Oct 1998). https://doi.org/10.1145/384265.291029
12. Halevy, B., Haynes, T.: Parallel NFS (pNFS) Flexible File Layout. RFC 8435 (Aug 2018). https://doi.org/10.17487/RFC8435
13. Hammerspace, Inc: Hammerspace unveils reference architecture for large language model training. Tech. rep., Hammerspace, Inc (2024), https://hammerspace.com/hammerspace-unveils-reference-architecture-for-large-language-model-training/
14. Hildebrand, D., Honeyman, P.: Exporting storage systems in a scalable manner with pnfs. In: Mass Storage Systems and Technologies, Proceedings. 22nd IEEE/13th NASA Goddard Conference on. pp. 18–27,. IEEE
15. IOR, H.: IOR and mdtest. https://github.com/hpc/ior (2026)
16. Lewars, J.: IBM Spectrum Scale: Performance Update (2020), https://www.spectrumscaleug.org/wp-content/uploads/2020/09/004-spectrum-scale-performance-update.pdf, introducing SSUG::Digital

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40

17. Lewis, N., Bez, J.L., Byna, S.: I/O in machine learning applications on HPC systems: A 360-degree survey. ACM Comput. Surv. **57**(10), 256:1–256:41 (2025). https://doi.org/10.1145/3722215
18. Linux Kernel Authors: vfs: Add 64 bit i_version support. https://github.com/torvalds/linux/
19. Lu, M.Y., Williamson, D.F., Chen, T.Y., Chen, R.J., Barbieri, M., Mahmood, F.: Data-efficient and weakly supervised computational pathology on whole-slide images. Nature Biomedical Engineering **5**(6), 555–570 (2021). https://doi.org/10.1038/s41551-020-00682-w
20. Mkrtchyan, T., Green, C., Hoyos, K., Litvintsev, D., Millar, A.P., Morschel, L., Rossi, A., Sahakyan, M., Starr, D.: dcache: The storage system of choice for data-intensive applications. Comput. Softw. Big Sci. **9**(1), 20 (2025). https://doi.org/10.1007/S41781-025-00152-5
21. Niazi, S., Ismail, M., Haridi, S., Dowling, J., Grohsschmiedt, S., Ronström, M.: HopsFS: Scaling hierarchical file system metadata using NewSQL databases. In: 15th USENIX Conference on File and Storage Technologies (FAST 17). pp. 89–104. USENIX Association, Santa Clara, CA (Feb 2017), https://www.usenix.org/conference/fast17/technical-sessions/presentation/niazi
22. Oeste, S., Kluge, M., Tschüter, R., Nagel, W.E.: Analyzing parallel applications for unnecessary I/O semantics that inhibit file system performance. In: Bienz, A., Weiland, M., Baboulin, M., Kruse, C. (eds.) High Performance Computing - ISC High Performance 2023 International Workshops, Hamburg, Germany, May 21-25, 2023, Revised Selected Papers. pp. 161–176. Lecture Notes in Computer Science, Springer (2023). https://doi.org/10.1007/978-3-031-40843-4_13
23. Palencia, J., Nadkarni, A.: High-Performance NFS Storage for HPC-AI. Tech. rep., VAST Data Inc. (2023), https://assets.ctfassets.net/2f3meiv6rg5s/I8ZSlykavxe3LFrlbnOgr/f6816cda4c6933fd623fb104c3d57064/idc-high-performance-nfs-storage-for-hpc-ai.pdf
24. Paul, A.K., Karimi, A.M., Wang, F.: Characterizing machine learning I/O workloads on leadership scale HPC systems. In: 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2021, Houston, TX, USA, November 3-5, 2021. pp. 1–8. IEEE (2021). https://doi.org/10.1109/MASCOTS53633.2021.9614303
25. Shepler, S., Eisler, M., Noveck, D.B.: Network file system (NFS) version 4 minor version 1 protocol. RFC **5661**, 1–617 (2010). https://doi.org/10.17487/RFC5661, https://doi.org/10.17487/RFC5661
26. The PostgreSQL Global Development Group: PostreSQL. https://www.postgresql.org/ (2026)
27. Vef, M.A.: Analyzing File Create Performance in IBM Spectrum Scale. Master's thesis, Johannes Gutenberg University Mainz (2016), https://www.staff.uni-mainz.de/vef/pubs/vef2016thesis.pdf
28. Wang, C., Mohror, K., Snir, M.: File System Semantics Requirements of HPC Applications. In: Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing. p. 19–30. HPDC '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3431379.3460637

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29912-3_40