Computing and Software for Big Science (2025) 9:20
https://doi.org/10.1007/s41781-025-00152-5

BRIEF REPORT

dCache: The Storage System of Choice for Data-Intensive Applications

Tigran Mkrtchyan[1] · Christopher Green[3] · Karen Hoyos[1] · Dmitry Litvintsev[3] · Paul Millar[1] · Lea Morschel[1] · Albert Rossi[3] · Marina Sahakyan[1] · Darren Starr[2,4]

Received: 15 April 2025 / Accepted: 12 November 2025

Abstract

The ever-increasing volumes of data produced by modern scientific facilities like EuXFEL and LHC put significant stress on data management infrastructure operated by laboratories and research centers. The challenges to be addressed span the entire data life cycle, from ingest and efficient data analysis to long-term preservation, typically involving large tape libraries. dCache, a storage system developed in collaboration between the Deutsches Elektronen-Synchrotron (DESY), Fermi National Accelerator Laboratory, and Nordic e-Infrastructure Collaboration (NeIC), is designed to manage a large number of disk servers and to facilitate transparent data migration to and from archival storage. Its multifaceted approach offers a unified method to support a variety of scientific use cases with the same storage infrastructure, including high-throughput data ingest, data sharing over wide area networks, efficient access from HPC clusters, and long-term data preservation on tertiary storage. Initially developed for high energy physics (HEP) experiments, dCache is now used by various scientific communities, including astrophysics, biomedical research, and life sciences, each having specific requirements. This paper presents architecture, deployment strategies, performance and scalability enhancements, and recent advancements in dCache addressing the needs of scientific communities. Finally, we touch on the development and release process, ensuring the software's high quality.

Keywords DCache · Storage · HSM · Data access

Introduction

The dCache project started in 2000 as a collaboration between Deutsches Elektron-Synchrotron (DESY) and Fermi National Accelerator Laboratory. The mission then was to develop a common storage software for the two

Christopher Green, Karen Hoyos, Dmitry Litvintsev, Paul Millar, Lea Morschel, Albert Rossi, Marina Sahakyan and Darren Starr have contributed equally to this work.

✉ Tigran Mkrtchyan
tigran.mkrtchyan@desy.de

Christopher Green
greenc@fnal.gov

Karen Hoyos
karen.hoyos@desy.de

Dmitry Litvintsev
litvinse@fnal.gov

Paul Millar
paul.millar@desy.de

Lea Morschel
lea.morschel@desy.de

Albert Rossi
arossi@fnal.gov

Marina Sahakyan
marina.sahakyan@desy.de

Darren Starr
darren.starr@usit.uio.no

[1] Deutsches Elektronen-Synchrotron DESY, Notkestrasse 85, 22607 Hamburg, Germany

[2] The University of Oslo, Boks 1072, NO-0316 Blindern, Norway

[3] Fermi National Accelerator Laboratory (FNAL), PO Box 500, Batavia 60510-5011, IL, USA

[4] Nordic e-Infrastructure Collaboration (NeIC), c/o NordForsk, Stensberggata 25, NO-0170 Oslo, Norway

Published online: 25 November 2025

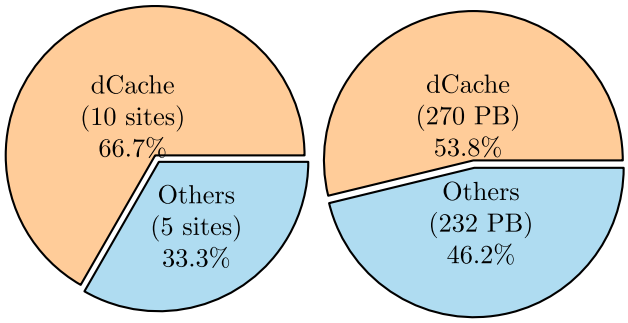

Fig. 1 53% of the total storage for WLCG at Tier-1 centers is provided by dCache instances

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 namespace and clean separation between that namespace and the 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. The focus on network protocols that supported transferring file data directly between clients and the nodes holding that data allowed dCache to scale, matching the storage demands.

At about the same time, CERN's LHC facility began adopting grid technology, resulting in the creation of WLCG—The Worldwide Large Hadron Collider Computing Grid. WLCG is an international collaborative project that brings together computational and storage resources of many research institutes and universities across the world to analyze the data collected by the four LHC experiments. At the time, WLCG followed a strict hierarchical model, known as the MONARC [1]. dCache software was proven popular within WLCG; many laboratories and universities are deploying dCache to manage their data. As of March 2025, as shown in Fig. 1, combined, these dCache storage instances accounted for more than 50% of the overall WLCG storage capacity, excluding CERN.[1] Therefore, dCache played a critical role underpinning the discovery of the Higgs boson [2].

Today, dCache has been used in production for over 20 years and is deployed across the globe [3]. Increasingly, sites are using dCache to support communities with requirements different from those of the LHC experiments. For example, DESY facilities and services now support photon sciences, biology, future accelerators R&D, and others. Over the years, dCache has proven its scalability and performance. The Photon Science instance at DESY and the so-called public dCache at Fermilab host more than 1.5 billion files per instance, demonstrating the scalable design of the dCache namespace service. The EuXFEL instance at DESY combined 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. From its earliest versions on, dCache has been responding to the challenges presented by new user communities and new and emerging workflows and technologies, developing and adopting innovative solutions to satisfy the demands for increased productivity [4].

Despite its maturity, dCache maintains agility and flexibility that allow it to adapt to changing technology and end-user requirements. 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 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.

This paper discusses technical details of the dCache architecture, deployment scenarios, and recent developments.

Technical Design

dCache is a storage system designed for storing and retrieving huge amounts of data distributed among a large number of heterogeneous server nodes under a single virtual filesystem tree with 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 IO libraries, and a single binary package for various operating systems.[2] By its design, dCache follows a scalable distributed storage architecture [5], where metadata and data servers are implemented as different services. It strictly separates the file namespace of its data repository from the actual physical location of the data. File names, attributes, and directory trees are managed in an internal database and are exposed through a namespace component. Every file (and directory)

[1] The numbers are taken from WLCG operation dashboards.

[2] 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.

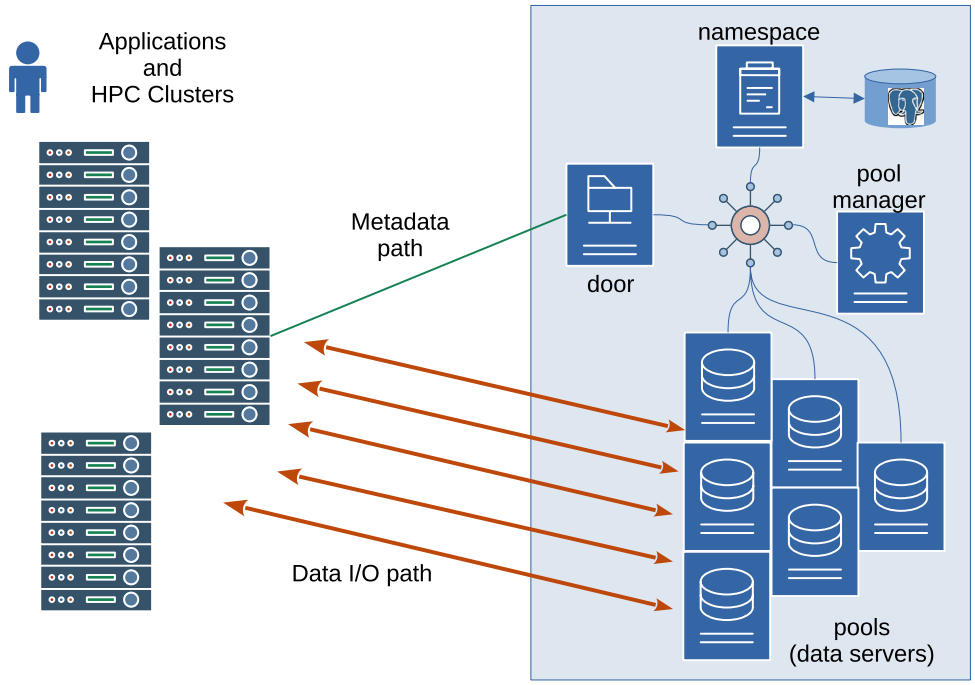

Fig. 2 dCache overview design. The minimal setup consists of four components: protocol-specific user entry point, pool selection unit, metadata server, and data server. The components communicate by sending messages to each other

has a unique identifier independent of its user-exposed name and location. By using this level of indirection, dCache can provide *location transparency*, *location independence*, and *user mobility* to stored data. This means that the name of a file does not depend on its physical location; the name of a file does not change when its physical location changes; users can access the same file by the same full file name from different hosts [6]. Moreover, dCache can accommodate multiple copies of a single file, dynamically add or remove new locations, and make use of external storage like S3 or hierarchical storage management (HSM), typically utilizing magnetic tape. The system scales horizontally with the number of data servers: adding new nodes to the system increases both storage capacity and aggregated data bandwidth.

A simplified dCache architecture is visualized in Fig. 2. 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 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 components communicate by sending or receiving messages over a TCP/IP network. Multiple redundant copies of dCache components can be started in a single deployment for high availability and load balancing. The topology auto-discovery and coordination between multiple instances of dCache services are based on Apache Zookeeper [7]—an open-source, highly reliable coordination service for distributed applications. Data resilience is provided by redundant file replicas and/or a tape copy.

The namespace/metadata component of dCache [8] is composed of several layers (Fig. 3). The topmost layer is called *PnfsManager*,[3] and is responsible for interactions between the rest of dCache and the underlying filesystem backend [10]. The second layer is a filesystem abstraction that supports multiple implementations. Currently, only one implementation exists, called *Chimera* [11], which is built on top of a relational database. At startup, Chimera detects the database flavor and enables database-specific optimizations. A generic SQL driver is used if a database flavor-specific driver is missing. In production deployments, only the PostgreSQL [12] database is used. The development and unit testing rely on the embedded HSQLDB [13] database management system. Such an architecture allows gradual evolutionary changes to refine the namespace component to respond to new requirements or technology changes. An integrated automatic database schema migration applies filesystem structural changes when new versions are installed and enables software downgrades if needed. With the dCache metadata service improvements introduced in version 9.2, dCache demonstrated a file-creation rate of up to 17,500 files per second when running the widely used HPC benchmark *mdtest*, as shown Fig. 4.

[3] The name comes from Perfectly Normal File System [9], the original backend for file metadata in dCache.

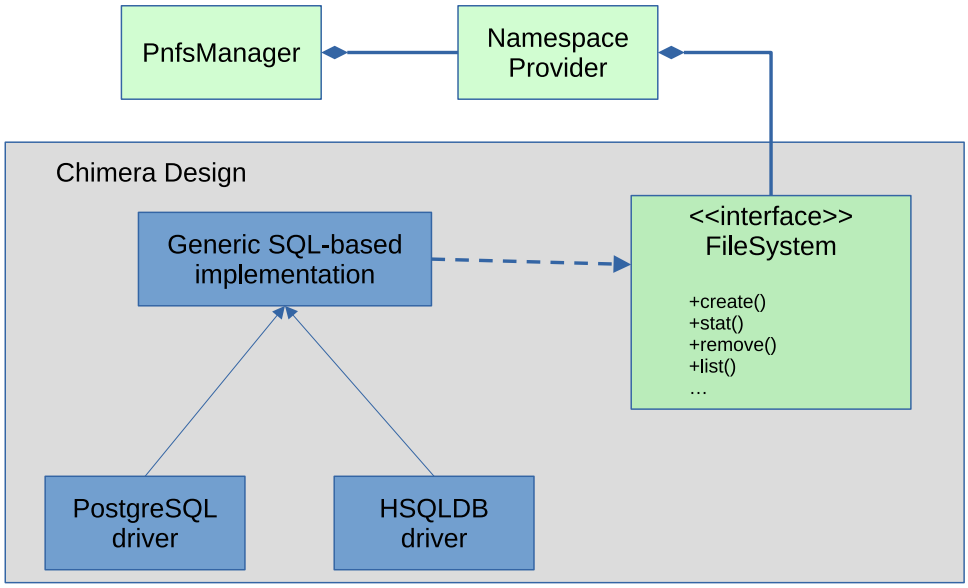

Fig. 3 dCache's namespace implementation diagram. The metadata service uses a filesystem-like interface that has multiple implementations. The RDBMS-based implementation supports database-specific drivers for SQL query optimization

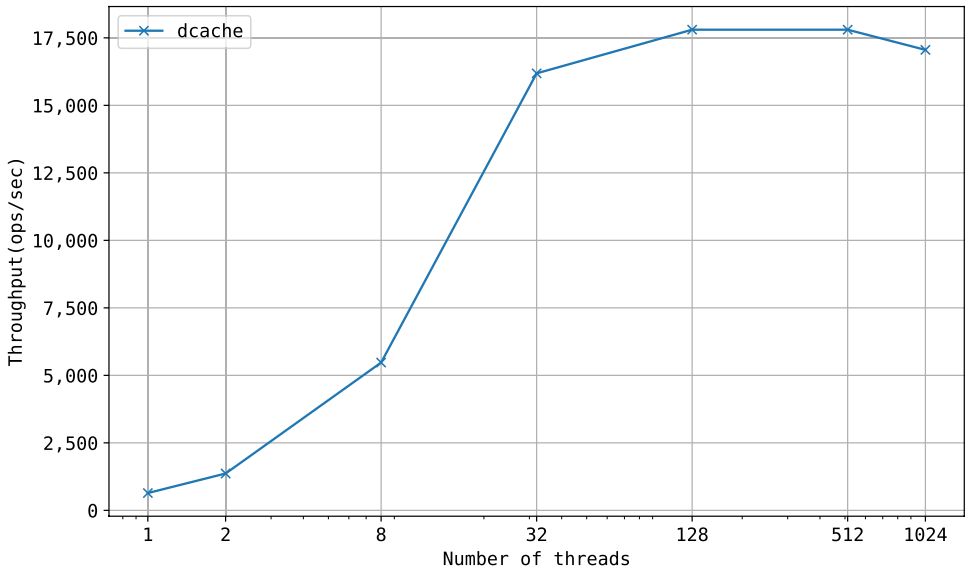

Fig. 4 dCache aggregated file creation rate. The mdtest workload was creating zero-byte files to benchmark only the namespace latency and throughput

A single dCache installation can run multiple copies of the namespace service. Typically, they share the same database. Alternatively, a namespace service might utilize a standby instance of the underlying database for read-only workloads. The replication between primary and secondary databases is performed outside of dCache by database native replication methods, such as by *streaming replication* [14] for PostgreSQL.

The distribution of data in dCache is crucial for resource utilization, hot spot avoidance, and data flows separation. dCache's *PoolManager* service plays a key role in this process. The *PoolManager*, the core of the dCache system, is responsible for deciding which data server will handle the transfer requests when a user is reading or writing files. This decision-making process is based on configurable policies that take into account available capacity, network topology, and data server system load. The system can be logically split into many partitions, with different data placement policies used wherein. dCache supports several placement policies, called partition types:

- *Classic*: Provides a straightforward approach to pool selection based on a combination of the available space and the number of ongoing transfers in regard to the configured limit. Despite its simplicity, this policy works well only if all data server has comparable capacity.

- *LRU* (least recently used): selects the pool that has not been used the longest. It aims to distribute the load more evenly by regularly using all pools.
- *Random*: Randomly selects a pool from the available pool set. While simple, it may lead to uneven load distribution if not carefully managed.
- *WASS* (weighted available space selection): Selects pools randomly weighted by available space, incorporating factors such as time since a file was last accessed, the amount of garbage collectible files, and load information. It is the default partition type designed to provide a balanced approach to load distribution.

If all pools containing the requested file are busy, *PoolManager* will create a new replica of the file on an idle pool. If the file is not available on any online pools, staging from a connected tape system will be triggered.

To scale with the number of clients, dCache redirects the client to the server selected for the given transfer by PoolManager. An example call sequence for file reading is shown in Fig. 5. 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 checking the access rights, the door asks PoolManager for a pool that matches the given transfer. As mentioned earlier, PoolManager selects a pool based on file availability and returns a pool that will service the request. Then, the door contacts the selected pool to authorize user access. On reply, the pool will return a client redirect information, which is then passed 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.

It is worth mentioning that the described flow has a built-in self-consistency correction. If a Pool gets a request for a file replica that it does not host, the metadata server will be notified about such an inconsistency, the stall record in the metadata server will be removed, and the selection process will start over.

Federated Systems

dCache allows different institutes to contribute resources to an aggregated storage system [15] spanning countries. dCache may be configured so that the data is preferentially accepted using local storage when available, falling back to using remote storage if all local storage is either full or off-line. Today, two WLCG sites are operating in such a distributed deployment: Nordic Tier-1, known as NDGF, and ATLAS Great Lake Tier-2, known as AGLT2.

The Nordic Data-Grid Facility (NDGF) Tier-1 was formed by universities from Denmark, Sweden, Norway, Finland, Slovenia, and Switzerland, resulting in a single distributed Tier-1 center for WLCG. It deployed a single dCache instance with storage nodes hosted by participating institutes. The geographical distribution of NDGF resources is shown in Fig. 6. The core services run in Copenhagen in a high-availability setup, while other sites provide disk storage

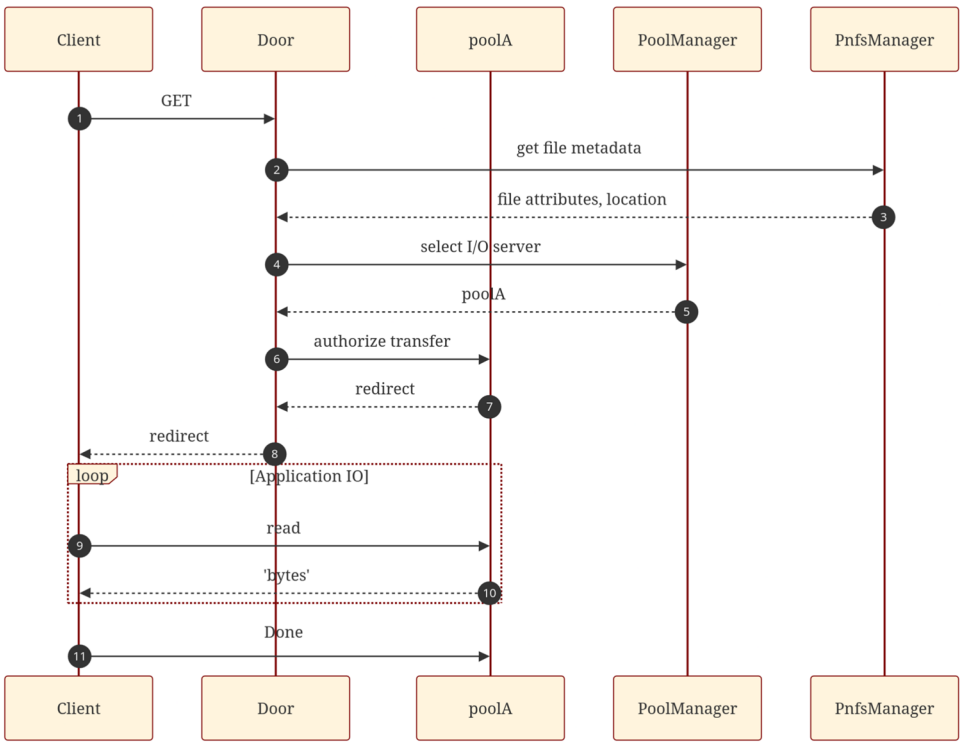

Fig. 5 dCache data access flow. For data I/O client is redirected to the appropriate data server

Fig. 6 NDGF Tier-1—six countries, one dCache instance. The core services run in Copenhagen, Denmark. The pool nodes run in Finland, Sweden, Switzerland, Slovenia, and Norway

that is connected to a tape library. The system ensures that the Tier-1 center can accept data from CERN even if some sites are unavailable.

The AGLT2 is a WLCG distributed Tier-2 site formed as a collaboration between the University of Michigan and Michigan State University. Data collected at any campus is written locally, using storage nodes located at the site. With data servers being part of the single dCache instance, the files are part of the same namespace and can be accessed from any location.

dCache can be configured to maintain multiple copies of certain data. The location of these copies may be constrained, for example, to different geographic locations. This ensures that the data is still available if one of the locations experiences a disaster. The other use case for geographically distributed replication is latency hiding for high data processing efficiency. Such a data placement policy can be specified as a part of the quality of service definition, see sect. Quality of Service.

Although the AGLT2 and NDGF sites have been operating for over a decade, such distributed setups are currently still exceptions. However, to reduce operational overhead and improve data availability, more and more small Tier-2 sites are combining their storage resources together to build so-called data lakes [16]. Clients requesting data stored locally on campus will read the data from those resources. Attempts to read data exclusively located on the remote data servers will trigger an automatic dCache-internal replication of that data to local resources. First and subsequent accesses will use the local copy, which will be available until it is eventually expunged to accept newer replicas when the cache fills to capacity. In addition to on-demand replication, dCache provides mechanisms for manual data placement. The replicas can be cached copies only, i.e., removed when space is needed, or have a guaranteed availability window

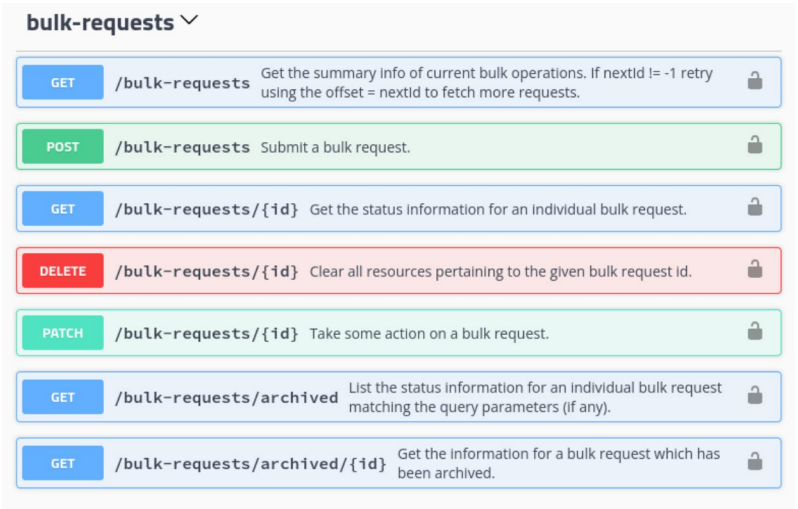

Fig. 7 The RESTful Interface to the Bulk Service

on the remote site to support scheduled data processing activities.

The caching and the data placement have always been at the foundation of the dCache design. Still, to get distributed deployments operational, a special configuration was needed, and the addition and removal of cache nodes on a remote site required manual intervention at the main site. Therefore, the dCache developers have introduced the concept of a *Zone* to reduce the operational cost of distributed deployments. A Zone is a logical group that combines multiple dCache services and is available in data placement and replication rules and for dCache inter-component communication. In high-availability (HA) setups, where redundant copies of critical components are each running in different Zones, the Zone-local instance of a component is preferred to reduce performance penalties due to network latency. For data movement, the standard dCache internal pool-to-pool copy is used, and all data integrity mechanisms, like checksumming or TLS, are available for in-transit data protection. The caching sites can be spawned on demand when data processing needs to be extended by external compute resources without so-called managed storage.

The dCache services, Zones, and caching nodes can be dynamically added and removed in the distributed deployment without configuration changes on other sites. This reduces the operational overhead of distributed deployments and provides a foundation for highly available dynamic storage federation. In addition, combined with a site-local read-only namespace replica, which prefers availability over consistency in network partitioning, full access to the data can be provided even when the connection to the main site is lost without compromising authorization and data integrity. This model can help smaller Tier-2 s to become a more integral part of their associated Tier-1 s, but still benefit from local expertise and deployed technology.

Quality of Service

User communities are looking to maximize the return on their investments in IT infrastructure, computing, and storage while operating on limited budgets. Although the cost-per-terabyte of storage is generally decreasing, there is an increasing demand to store ever more data. One possibility is to provide a tiered storage; that is, some storage capacity is made available at a lower cost with limited IO performance or reliability, while other storage is made available at a higher cost, with enhanced IO performance and reliability. Different technologies and procedures lend themselves naturally to building tiered storage systems, like the use of a combination of spinning disks and SSDs for different workloads or the use of RAID systems and erasure-coded JBODs for fault tolerance and performance efficiency.

In dCache, we have introduced a concept of quality of service (QoS) for storage developed as a part of the INDIGO-DataCloud project [17]. QoS describes how data is stored within dCache and exposed as a REST API. It is an extension of the long-standing support for storing data on disk and tape.

The original stimulus for QoS and, indeed, dCache itself comes from using disks as cache in front of tape for particle physics analysis. However, QoS has more dimensions than just latency and price. The new requirements, such as durability, availability, and technology independence, play an important role in the data life cycle of scientific communities, such as EuXFEL [18].

To address these requirements, the *Resilience* [19] subsystem of dCache, which is responsible for data durability, is evolving into the QoS engine. A significant amount of Resilience's architecture is needed to be refactored to be placed on top of the already implemented functionality.

Bulk Service

To operate on a large number of files, dCache provides the *Bulk* Service. It enables users to perform deletions, data migrations to/from tape, and QoS transitions of files as described in Sect.Quality of Service. The Bulk Service in dCache is built as an asynchronous task execution module that allows users to submit batch requests via RESTful API, as shown in Fig. 7. A bulk request consists of a set of target files or directories, the operation to be performed (delete, pin, qos update, etc.), and additional arguments depending on the operation. Upon submission, the Bulk Service returns a unique request ID that can be used to query the request status or cancel the request if needed.

dCache's Bulk Service API has been adopted by the WLCG storage providers. This adoption is known as the *WLCG Tape API* [20] and has replaced the SRM interface [21] as the standard protocol for interaction with tape storage.

Non-HEP Analysis

A critical issue when dealing with large volumes of data is how to grant access. Many standard protocols lack the features to benefit from distributed data, which is common in multi-petabyte storage systems.

The analysis framework ROOT [22] developed and maintained at CERN supports various network protocols with the ability to add more. This has allowed using ROOT with dCache via HEP-proprietary protocols, such as DCAP or XRootD [23]. With a growing demand for non-HEP software, like Jupyter Notebooks and Apache Spark, POSIX-like data access is expected. Moreover, with the availability of opportunistic HPC resources to HEP experiments, the provided POSIX interface must not require any special software installed on the worker nodes.

To provide access to the distributed data without the need for a driver or application changes, dCache can be accessed as a locally mounted filesystem over NFSv4.1/pNFS protocol [24] 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 of accessing large-scale storage systems [25] in a distributed fashion. For this reason, dCache supports NFSv4.1/pNFS, with an emphasis on pNFS. This allows the storage system to be mounted without requiring any driver or application changes. To our knowledge, dCache is the first publicly available storage system to utilize pNFS in production [26, 27]. With recent developments in the inter-component communication protocol used by dCache [28], the overhead of the internal communication is reduced, which improves HPC job efficiency, where the file's metadata access is as essential as the data access itself.

Another case of non-ROOT-based analysis is using commercial applications, such as MATLAB, under Microsoft Windows OS. To support MS Windows users in storing and accessing data in dCache, the SMB protocol was added. Rather than implementing the SMB protocol support directly in dCache, a protocol translation server runs the open-source SAMBA [29] software while providing access to dCache storage via re-exporting NFS.

In close collaboration with Linux kernel developers, dCache is an early adopter and demonstrator of the evolution of the NFS protocol and gives early access to new developments, such as extended file attributes [30] over NFS or NFS-over-TLS [31].

Data Labeling

Traditional file systems organize data in a hierarchy of directories [32]. The directory structure is built as a hierarchy of containers that group files and directories by a logical unit, e.g., experiment run number, beam time, data type, etc. The data processing application can then be pointed to such a directory to process all data therein. However, sometimes only a subset of data files is required based on one specific attribute. Recent changes in dCache have introduced dynamic grouping of files into *virtual directories* based on a user-provided grouping key– a file label. These labels can be added, removed, and queried via the dCache REST API. A single file can have multiple labels and thus exist in multiple virtual directories simultaneously. To ensure seamless integration with existing workflows, the virtual directories are exposed to the user as regular read-only directories and are hence accessible via all supported protocols, such as NFS and WebDAV. Future developments aim to integrate end-user-provided metadata extraction and automate data labeling.

Tape Interface

While large hard disk-based storage is efficient in terms of cost, space, and volume, magnetic tape still offers the most economical solution for long-term data archival, especially for rarely accessed, so-called cold data. When not used, the tape cartridges do not require electrical power; they can store up to 50 TB [33] of uncompressed data and have an expected lifetime of 15–30 years. Tape libraries with robotic arms are used to organize a large number of tape cartridges.

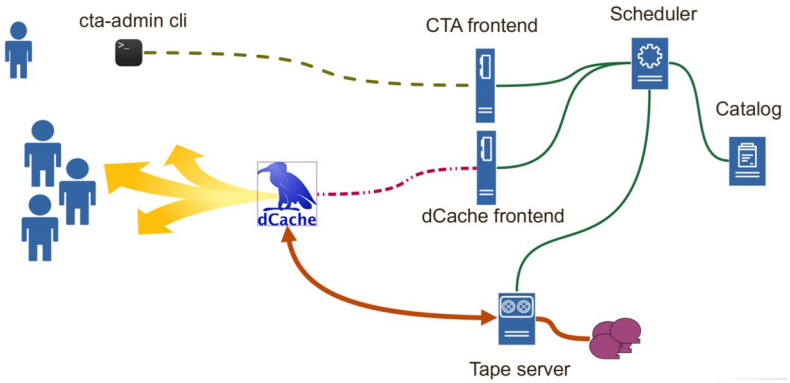

Fig. 8 dCache integration with CTA: a dCache-specific frontend submits the requests into the CTA scheduler queue. The CTA scheduler processes requests the same way as if they were submitted with EOS

There are downsides to tape-based technologies. When streaming, a modern tape drive achieves up to 400 MB/s transfer rate, but this high rate is saddled with a long latency of tape positioning of about 50 s on average. On top of that, the mounting time, the time which is needed by the robotic arm to put the tape into an appropriate tape drive, can reach minutes. Magnetic tapes do not support concurrency, meaning that only one stream, or only one file, can be read or written at any point in time. A high level of integration between hardware and software components is required to achieve maximum efficiency of the available tape resources, i.e., maximize the time during which tape is read or written at the top speed.

Some scientific communities, such as those in photon science, exhibit very wide file size distribution, with an overall tendency toward smaller files. Directly storing these files results in an unacceptably poor efficiency of tertiary storage systems, particularly tape, which is optimized for streaming large files. As dCache bridges the file system view with the underlying storage and manages transitions between media, it is the natural place to solve the pool performance of storing small files on tape. Since 2015, a so-called small file service [34] has been deployed at DESY, which packs small files into containers based on specified aggregation policies for better tape resource utilization and scalability.

dCache has a flexible tape interface that allows it to connect to any tape system. There are two ways a file can be migrated to tape: either by calling a tape system-specific copy command, or by utilizing a tape system-specific driver within dCache called nearline storage provider. The latter has been shown (by the Tier-1 sites NDGF, TRIUMF, and KIT) to provide better resource utilization and efficiency than the former [35].

CERN Tape Archive (CTA) [36] is an open-source tape storage system developed by the CERN IT storage group to replace the legacy CASTOR system that had been used in the 2010s. It is designed to meet the requirements of LHC Run 3 as well as high luminosity LHC (HL-LHC) runs. The CTA project is actively adding flexibility into the system to allow wider adoption by other sites, encouraging contributions from other developers and eliminating CERN-specific dependencies in the provided binary packages [37].

CTA has two key components: the frontend and the tape daemon. The frontend accepts requests such as *Archive*, *Retrieve*, *Delete*, or *Cancel* from the attached disk storage and puts them into a request queue. If needed, the file catalog is then updated. The scheduling logic is embedded into the tape daemons, which are per-tape drive processes that scan for matching tasks in the request queue. Once the desired number of requests is reached, the data are moved between disk and tape media using an embedded XRootD client.

CTA supports two request types to trigger data migration between disk and tape: *Archive* and *Retrieve*. The retrieve operation can be canceled with a *Cancel* request. The *Delete* request is used to cancel the archive operation and/or to remove an existing file from the tape catalog. In the case of the EOS [38] disk frontend, all those operations are tied to EOS filesystem events, like *CREATE*, *CLOSE*, *READ*, and *DELETE*. The requests are encoded with Google's protobuf [39] and sent from the EOS server to the CTA frontend over the XRootD Scalable Service Interface (XrootdSSI). In addition to data migration requests, the frontend supports operations required by the *admin-cli* tool, like *define tape pool* or *enable/disable tape drive*.

The CTA scheduler queue is backed by a CEPH[40] object store instance or by a shared filesystem. This allows hundreds of thousands of requests to be collected for optimal scheduling decisions.

Out of the box, CTA comes with a frontend that communicates with EOS, the disk system deployed at CERN. However, as CTA's queuing system is not EOS-aware, other frontend implementations are possible. For seamless integration with CTA, dCache developers at DESY have implemented a CTA-specific nearline storage provider called dcache-cta[41, 42], and a corresponding CTA frontend component. The communication between dCache and the new frontend is based on Google's gRPC library and is not limited to dCache. Therefore, it can be used by other disk

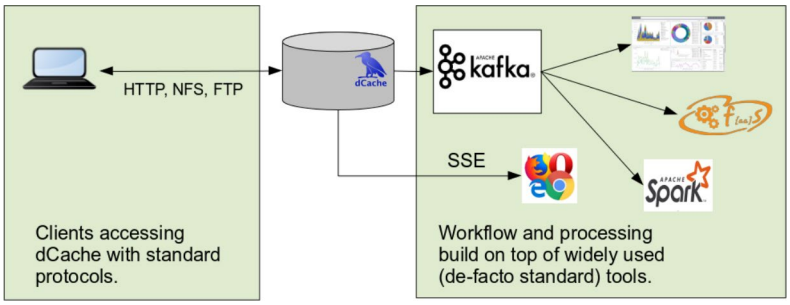

Fig. 9 dCache storage events. The storage events can be integrated into external services to provide monitoring information or to trigger processing in workflow systems

systems using the CTA[4] backend. The dCache-CTA integration is shown in Fig. 8.

As CTA has its own scheduler, the flush and restore request queuing at the dCache pools should be disabled to avoid double-scheduling. In this way, all HSM requests from dCache are directly submitted to the CTA scheduler.

Initially, dCache's tertiary storage connectivity interface was optimized for efficient writing to tapes, and file recall was considered a rare event. The recent ATLAS data carousel [43] activities with large-scale recalls have invalidated this assumption. Unordered requests to tape libraries result in a high number of tape mounts and reposition requests, thus resulting in low tape access efficiency overall. After analyzing the locations of files on tape, file access patterns, and logs provided by participating sites, bulk-recall optimization strategies were identified and implemented [44, 45].

Storage Events

Some research communities, especially those that process images, must populate metadata catalogs before the data or dataset can be processed. Traditionally, this is done by repeatedly querying a storage system to learn if new data is available. This approach does not scale with the number of clients, as each query uses resources to obtain the current status, eventually reaching the limit of the number of concurrent activities a service may sustain.

Storage events are a new way of interacting with storage. In contrast to the traditional request-response pattern, the storage system generates events when something happens, such as the creation of a file, access, removal, or migration to/from tape storage, and asynchronously delivers them to the clients. As storage events do not require polling, the client quickly learns of changes. It also scales well since these subscriptions typically require very few resources.

dCache provides two ways for clients to receive events: via Apache Kafka or by means of the W3C Server-Sent Events (SSE) protocol. The two event delivery mechanisms have different trade-offs with different intended audiences: they complement rather than compete with each other, providing a non-overlapping set of possible events. Some events are only available via Apache Kafka, while others are available via SSE. The Apache Kafka-based events are suitable for integration with IT infrastructure, such as monitoring systems or transfer analytics, while SSE-based events are for web-based end-user applications.

A new development activity within the dCache project tries to extend the namespace component with metadata catalog functionality. Such a catalog should then be populated automatically when new data arrives by utilizing storage events to execute scientific community-specific metadata extraction applications (see Fig. 9).

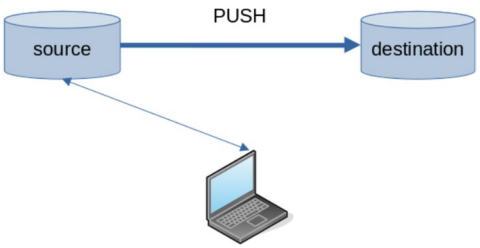

Fig. 10 TPC push mode. The source system pushes data to a destination

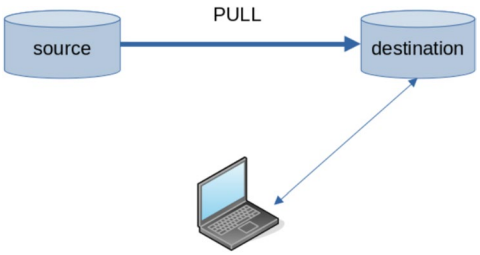

Fig. 11 TPC pull mode. The destination system pulls data from a source

[4] The CTA developers at CERN are in the process of adopting the gRPC-based frontend to the EOS system.

Third-Party Copy

For efficient data movement between sites, so-called third-party copy (TPC) is used. During such a transfer, the client that initiates the data movement might maintain a control connection to one or both systems, but the data is transferred between them without going through this client at the 3^{rd} location. This model allows higher-level management tools to coordinate data movement without directly participating in the transfers. There are two TPC modes—*push*, in which the initiating client connects to the source storage system to push the data to the destination, as demonstrated in Fig. 10, and *pull*, in which the client connects to the destination system to pull the data from a source, as demonstrated in Fig. 11.

In the last two decades, the LHC experiments used GridFTP [46] to perform TPC between the sites. With the announcement of the end-of-life for Globus Toolkit by the Globus Alliance, the WLCG Data Organization Management and Access Working Group (DOMA-TPC) [47] was established to investigate alternatives to the GridFTP protocol for bulk transfers across WLCG sites. This has resulted in WebDAV- and XRootD-based [23] third-party transfer implementations. The Globus Security Infrastructure (GSI)-based authentication has been replaced with standard and widely used OAuth2 authorization and OpenID Connect authentication mechanisms [48].

Authentication and Authorization

Although it is possible to allow anonymous access to dCache, it is usually desirable to authenticate users. The user then has to connect using one of the supported access protocols and log in with credentials that prove their identity. In the Grid world, these credentials are typically X.509 certificates, but dCache also supports other methods like username/password, Kerberos authentication, or OAuth2 tokens. All these mechanisms can be combined in a single deployment to cover the needs of local and remote access to the same data. This means that, independent of the access protocol and security flavor in use, a user will always be mapped to the very same internal identity, and the same permission and access rules will be applied.

To achieve this flexibility, dCache has an authentication framework that allows plug-in modules for different steps of the authentication and mapping process. The system distinguishes four steps in the authentication process: login, user mapping, user account, and session operations. Each module within one of the steps can decide whether the provided information is sufficient to proceed with the module chain, to let the request fail, or to proceed to the next step. At the end of the complete chain, it is expected that one valid user ID and at least one valid group ID are present in the list of principals. Site administrators can configure different modules for each step with corresponding wiring to achieve the desired behavior (see Listing 1).

```
auth      optional   x509
auth      optional   voms
auth      optional   ldap

map       sufficient gridmap
map       sufficient ldap

identity  requisite  ldap
session   sufficient authzdb
```

Listing 1: dCache authentication configuration example

In the example shown in Listing 1, a user-provided X509 certificate is extracted, a virtual organization membership is validated, and finally, the username and password are verified against the LDAP server. All these steps are defined as *optional*, therefore, a missing credential would not fail the login process. The mapping step stops after the first successful mapping using a gridmap file or LDAP.

Development Process and Quality Assurance

From the outset, dCache has been an international software project with developers at DESY, Fermilab, and, later, NeIC. As a critical part of site infrastructure, the dCache project has stringent requirements on software quality and build reproducibility. From early on, the dCache project has adopted semantic versioning[5] and a time-based release policy [49] as a strategy to deliver software packages to the sites. With the growing number of software branches to support and the increasing complexity of testing, the manual build and test steps were replaced with continuous integration (CI) systems that execute automatic build and test procedures triggered by code changes.

Since the fall of 2020, DESY-IT has been offering a central GitLab service to local scientists to host, build, and publish their software projects. Integrated with the Kubernetes [50] container orchestration solution, GitLab provides all the necessary building blocks for a complete software lifecycle. Today, the GitLab service at DESY-IT hosts more than 12,000 projects for on-site scientists and external collaborations, like the Belle-II experiment at KEK in Japan.

GitLab-CI is a build, test, and packaging automation tool integrated into the GitLab source code hosting software. It is based on the Configuration as Code philosophy, where the run pipeline definition becomes integrated into the application source code. In the case of GitLab-CI, the build and test pipeline is described in a YAML[6]-formatted file, typically called *.gitlab-ci.yml*. The pipeline file describes multiple stages that are to be executed sequentially. Each stage consists of one or more jobs that perform a specific action required in that stage. All jobs that are defined in a single stage are executed in parallel. By default, the pipeline execution is aborted if one of the jobs in a stage fails. Listing 2 describes the dCache build, test, and release pipeline stages.

```
stages:
  - build          # build rpm, tar, deb, oci
  - sign           # PGP sign rpms
  - testenv_pre    # prepare k8s env
  - test_infra     # deploy pg, zk, ...
  - test_deploy    # deploy dcache helm
  - testing        # run all tests
  - testenv_post   # collect logs
  - upload         # publish packages
```

Listing 2: dCache build/test/publish CI pipeline stages

The pipeline stages are logically divided into three major groups with the following responsibilities:

1. *Can we build the code?* At this stage, code is compiled, unit tests are run, and target packages (rpm, deb,...) are created.
2. *Can we deploy the code?* This stage validates that the produced packages can be deployed on target platforms and dCache services can be started.
3. *Did we break expected functionality?* This stage runs various tests that typically mimic end-user and application behavior. In contrast to unit tests, this testing requires a fully operational dCache instance and interacts with external services, like Identity Providers (IdP) or Virtual Organization Management Systems (VOMS).

The pipeline relies on Linux containers for the dCache test deployment and the required infrastructure for integration and user tests. The Kubernetes service provided by the DESY-IT Systems group is utilized for container orchestration. All container-based applications and services are deployed using Helm charts. The dCache-specific Helm charts [51] mimic a multi-node setup. Multiple versions of

[5] https://semver.org/spec/v2.0.0.html.

[6] https://yaml.org/spec/1.2.2/

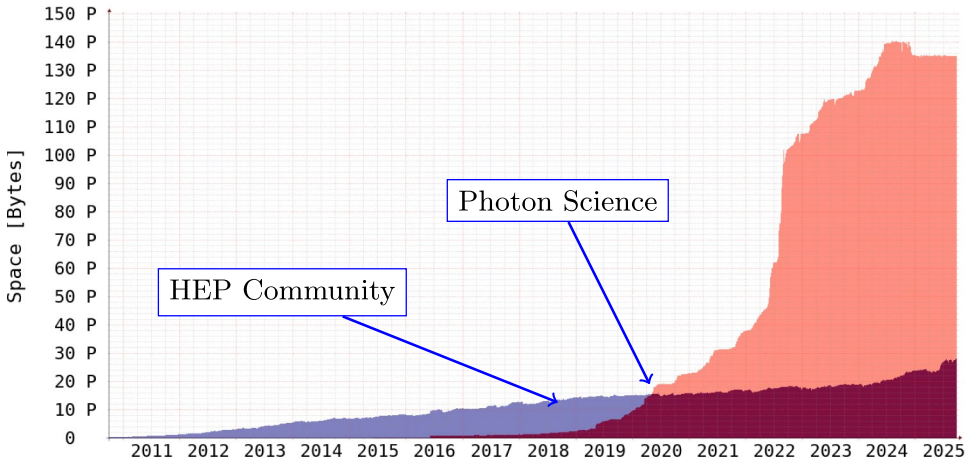

Fig. 12 On disk data per the scientific community at DESY. With the start of EuXFEL and Petra-III, the Photon Science community dominated the storage needs

dCache components can be deployed simultaneously to test backward compatibility and interoperability. The complete build, test, and publish pipeline comprises over thirty jobs covering the entire development life cycle. The pipelines triggered by the release process additionally publish signed packages into the *dcache.org* download area. For each pipeline, a new Kubernetes namespace is created, allowing better resource isolation to avoid running memory-intensive dCache processes alongside building jobs from other pipelines. Before the namespace is destroyed at the end, logs from all containers are collected for debugging purposes.

Although the dCache developers use the GitLab service at DESY for their CI, the main code repository remains in GitHub. This means that all code changes are committed and pushed to GitHub. A dedicated GitHub action [52] is used to synchronize code changes with GitLab at DESY and to trigger the CI pipeline.

Future Work

While dCache is a well-established production-ready solution that addresses many of the challenges of data-intensive science today, there are areas that need further development to make it ready for upcoming challenges expected by the EuXFEL upgrade, HL-LHC, PETRA-IV, and other future experiments. These areas are motivated by both the need to effectively use HPC resources and to reduce operational overhead while supporting system scale to exascale levels.

As mentioned in Sect. Federated Systems, there are already federated dCache that provide services to WLCG. However, such deployments require special, often non-trivial configurations. To address this issue, dCache developers are working on providing an easy-to-deploy container-based solution that can be pre-configured by the main site (the site that either runs the core services or has the main know-how) and distributed to satellite sites, which is expected to be a site with experienced system administrators or even only provide a standard infrastructure, such as a Kubernetes, to remotely deploy services. This kind of deployment is already exercised as part of the dCache CI pipeline, as discussed in Sect. Development Process and Quality Assurance.

Typical service providers need to organize a monitoring and an alarming system for mission-critical services. Usually, sites that run dCache already have some kind of monitoring system in place, but the integration with dCache is done manually. To reduce operational overhead with the growing number of nodes, starting release 11.1[53], dCache comes with an integrated Prometheus [54] exporter, which provides out-of-the-box integration into many monitoring and reporting systems. Today, the exporter provides a very limited number of metrics, which will be extended based on the site's feedback and needs.

Another aspect of monitoring is processing historical data to identify system bottlenecks, hardware misbehavior, and changed user access patterns. Today, such analysis is performed on a regular basis based on the so-called dCache billing log files, log entries corresponding to each transfer in the system. As those log entries might produce up to a gigabyte of data per day, for effective analysis, standard Big Data workflows are used.[7] This analysis performed manually and, often, comes too late; thus, the results are mostly used to guide future deployment configuration. The modern ML methods allow for the detection of such behavior anomalies automatically and adjust the system on the flight, as needed. As a part of a joint project with the University of Applied Science in Hamburg,[8] we have collected the requirements for such a self-adaptive system [55]. The first data access anomaly detection cases were presented as part of the Summer Student Program 2025 at DESY.

[7] CHEP 2024 paper "Monitoring large-scale dCache installations with storage events using Kafka streams" is pending to be published.

[8] https://www.haw-hamburg.de/en/

With the growth of dCache usage by the photon science community (see Fig. 12), the HPC workloads on dCache are growing as well. These workloads require POSIX-compliant, low-latency data access. Enabling *read delegations* in dCache is a natural next step in reducing server-side load and improving client efficiency when a single data files is accessed multiple times during the job runtime. A first working prototype is already available to sites as part of the standard dCache release. However, extensive benchmarking and caching policy optimizations are required for optimal HPC resource utilization.

At the same time, based on the recent adoption of NFS-over-TLS [31], the dCache developers are working on enabling OpenID Connect in the NFSv4.1 protocol to close the gap between WLCG security requirements and local data access at analysis facilities. The project's outcomes will allow short-lived token-based access to the data on a POSIX-like storage system.

Summary and Outlook

The dCache storage system is a scalable, high-performance solution designed to address the demands of data-intensive scientific research. Over the last 20 years, dCache has evolved from a relatively static storage system to a set of dynamic frameworks, providing *out of the box* solutions to many scientific community-specific challenges. Initially developed for High Energy Physics (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 HSM, providing seamless migration between disk and tape storage while ensuring efficient data access, high availability, and fault tolerance.

To efficiently help scientists manage the data life cycle, dCache introduces Quality of Service (QoS) mechanisms, allowing users to specify storage preferences based on latency, reliability, and cost constraints. Dynamic data labeling enables metadata-driven file organization. Storage event notifications allow real-time data-driven scientific workflow automation updates. The system supports third-party transfers for efficient site-to-site data movement, replacing legacy GridFTP with modern authentication and security standards like OAuth2 and OpenID Connect.

A key feature of dCache is its protocol-agnostic design. It supports standard and community-specific file access methods such as NFSv4.1 (pNFS), WebDAV, XRootD, allowing for wide interoperability across scientific computing environments. The system further enhances performance through federated storage deployments, where multiple institutions can pool resources while maintaining a unified namespace and intelligent data replication.

dCache continues to evolve through active development, rigorous automated testing, and a structured release cycle. The system's integration with Kubernetes and CI/CD pipelines ensures robust quality assurance and adaptability to emerging research needs. As scientific computing scales toward exascale data processing, dCache remains a flexible and future-proof storage solution for managing petabyte-scale datasets efficiently. With a focus on standard interfaces, dCache can be seamlessly integrated into the IT landscape, becoming an integral part of a site's infrastructure.

Finally, the dCache continues to develop toward a more adaptive, self-managing, and HPC-scalable storage system, ensuring that it remains capable of supporting the evolving needs of scientific communities in the era of exascale data management.

Acknowledgements This work was 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.

Author contributions All authors contributed equally to this work. These authors contributed equally to this work.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was supported by the project 'An interdisciplinary Digital Twin Engine for science' (interTwin), which received funding from the European Union's Horizon Europe Programme under Grant 101058386.

Data availability No datasets were generated or analyzed during the current study.

Declarations

Competing interests The authors declare no competing interest.

References

1. Aderholz M, Amako K, Auge E, Bagliesi G, Barone L, Battistoni G, Bernardi M, Boschini M, Brunengo A, Bunn J, Butler J, Campanella M, Capiluppi P, Carminati F, D'Amato M, Dameri M, Di Mattia A, Dorokhov A, Erbacci G, Williams D. (2001) Models of networked analysis at regional centres for lhc experiments (monarc). phase 2 report

2. Wengler JC (2012) How grid computing helped CERN hunt the Higgs . https://www.isgtw.org/feature/how-grid-computing-helped-cern-hunt-higgs
3. Fuhrmann P (2006) V.Gülzow: dCache, storage system for the future, 1106–1113 https://doi.org/10.1007/11823285
4. Millar A, Baranova T, Behrmann G, Bernardt C, Fuhrmann P, Litvintsev D, Mkrtchyan T, Petersen A, Rossi A, Schwank K (2012) Dcache, agile adoption of storage technology. J Phys Conf Ser 396(3):32077–087. https://doi.org/10.1088/1742-6596/396/3/032077
5. Gibson GA, Nagle DF, Amiri K, Butler J, Chang FW, Gobioff H, Hardin C, Riedel E, Rochberg D, Zelenka J (1998) A cost-effective, high-bandwidth storage architecture. SIGOPS Oper Syst Rev 32(5):92–103. https://doi.org/10.1145/384265.291029
6. Levy E, Silberschatz A (1990) Distributed file systems: concepts and examples. ACM Comput Surv 22(4):321–374. https://doi.org/10.1145/98163.98169
7. Apache Software Foundation: Apache ZooKeeper. https://zookeeper.apache.org
8. Mkrtchyan T, Chitrapu K, Garonne V, Litvintsev D, Meyer S, Millar P, Morschel L, Rossi A, Sahakyan M (2021) Dcache: interdisciplinary storage system. EPJ Web Conf 251:02010. https://doi.org/10.1051/epjconf/202125102010
9. Fuhrmann P (1997) A perfectly normal namespace for the desy open storage manager. http://www-zeuthen.desy.de/CHEP97/paper/409.ps
10. Mkrtchyan T, Chitrapu K, Litvintsev D, Meyer S, Millar AP, Morschel L, Rossi A, Sahakyan M (2024) DB back-ended filesystem for science. In: Störl, U. (ed.) Proceedings of the 35th GI-Workshop Grundlagen Von Datenbanken, Herdecke, Germany, May 22-24, 2024. CEUR Workshop Proceedings, vol. 3710, pp. 58–63. CEUR-WS.org, Germany https://ceur-ws.org/Vol-3710/paper9.pdf
11. Gasthuber M, Mkrtchyan T, Fuhrmann P (2005) Chimera - a new, fast, extensible and Grid enabled namespace service. Comput High Energy Phys Nucl Phys 2004. https://doi.org/10.5170/CERN-2005-002.1180
12. The PostgreSQL Global Development Group: PostreSQL. https://www.postgresql.org/
13. The HSQL Development Group: HSQLDB. https://hsqldb.org/
14. The PostgreSQL Global Development Group: PostgreSQL 16.3 Documentation, (2024). Chap. 20.6. https://www.postgresql.org/docs/16/runtime-config-replication.html
15. Behrmann G, Fuhrmann P, Gronager M, Kleist J (2008) A distributed storage system with dcache. J Phys Conf Ser 119(6):062014. https://doi.org/10.1088/1742-6596/119/6/062014
16. Bird I, Campana S, Girone M, Espinal X, McCance G, Schovancova J. Architecture and prototype of a WLCG data lake for HL-LHC. EPJ Web of Conferences **214**, 04024 https://doi.org/10.1051/epjconf/201921404024
17. Salomoni D, Campos I, Gaido L, Donvito G, Antonacci M, Fuhrman P, Marco J, Lopez-Garcia A, Orviz P, Blanquer I (2016) Indigo-datacloud: foundations and architectural description of a platform as a service oriented to scientific computing. arXiv preprint arXiv:1603.09536, https://arxiv.org/abs/1603.09536
18. Malka J (2023) *et al.*: Data Management Infrastructure for European XFEL. JACoW ICALEPCS2023, 1–02 https://doi.org/10.18429/JACoW-ICALEPCS2023-WE1BCO02
19. Rossi AL (2017) Data resilience in the dCache storage system. J Phys Conf Ser. https://doi.org/10.1088/1742-6596/898/6/062024
20. Afonso J, Caffy C, Patrascoiu M, Leduc J, Davis M, Murray S, Cortes P (2024) An http rest api for tape-backed storage. EPJ Web of Conf 295:01008. https://doi.org/10.1051/epjconf/202429501008
21. Österle H, Becker J, Frank U, Hess T, Karagiannis D, Krcmar H, Loos P, Mertens P, Oberweis A, Sinz EJ (2011) Memorandum on design-oriented information systems research. Eur J Inf Syst 20(1):7–10. https://doi.org/10.1057/ejis.2010.55
22. Brun R, Rademakers F, Canal P, Naumann A, Couet O, Moneta L, Vassilev V, Linev S, Piparo D, GANIS G, Bellenot B, Guiraud E, Amadio G, wverkerke Mato P, Timur P, Tadel M, wlav Tejedor E, Blomer J, Gheata A, Hageboeck S, Roiser S, marsupial Wunsch S, Shadura O, Bose A, CristinaCristescu Valls X, Isemann R. Root-project/root: V6.18/02. https://doi.org/10.5281/zenodo.3895860
23. XRootD: The XRootD Software Framework. https://xrootd.org/
24. Shepler S, Eisler M, Noveck DB (2010) Network file system (NFS) version 4 minor version 1 protocol. RFC 5661, 1–617 https://doi.org/10.17487/RFC5661
25. Hildebrand D, Honeyman P (2005) Exporting storage systems in a scalable manner with pnfs. In: 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005), pp. 18–27 https://doi.org/10.1109/MSST.2005.14
26. Elmsheuser J, Fuhrmann P, Kemp Y, Mkrtchyan T, Ozerov D, Stadie H (2011) Lhc data analysis using nfsv4.1 (pnfs): a detailed evaluation. J Phys Conf Ser 331(5):052010. https://doi.org/10.1088/1742-6596/331/5/052010
27. Fuhrmann P, Gasthuber M, Kemp Y, Ozerov D (2012) Experience with hep analysis on mounted filesystems. J Phys Conf Ser 396(4):042020. https://doi.org/10.1088/1742-6596/396/4/042020
28. Morschel L, Adeyemi O, Garonne V, Litvintsev D, Millar P, Mkrtchyan T, Rossi A, Sahakyan M, Starek J, Yasar S (2020) Dcache - efficient message encoding for inter-service communication in dCache: evaluation of existing serialization protocols as a replacement for Java object serialization. EPJ Web Conf 245:05017. https://doi.org/10.1051/epjconf/202024505017
29. Samba Contributors: Samba. Online; last access on 07.01.2025. https://samba.org
30. Naik M, Eshel M (2017) File system extended attributes in nfsv4. RFC 8276:1–28. https://doi.org/10.17487/RFC8276
31. Myklebust T, Lever C (2022) Towards remote procedure call encryption by default. RFC 9289:1–21. https://doi.org/10.17487/RFC9289
32. Tanenbaum AS (2008) Modern Operating Systems, 3, ed. Pearson Prentice Hall, Upper Saddle River, NJ
33. Spectra Logic Corporation: IBM TS1170 Tape Drive Technology. Online; 2023. https://spectralogic.com/wp-content/uploads/Datasheet-TS1170.pdf. Accessed 7 Jan 2025
34. Schwank K (2015) Transparent handling of small files with dCache to optimize tape access. J Phys Conf Ser 664:042048. https://doi.org/10.1088/1742-6596/664/4/042048
35. Musheghyan H, Petzold A, Heiss A, Ressmann D, Martin Beitzinger M (2020) The gridka tape storage: various performance test results and current improvements. EPJ Web Conf 245:04026. https://doi.org/10.1051/epjconf/202024504026
36. Davis M, Bahýl V, Cancio G, Cano E, Leduc J, Murray S (2019) Cern tape archive – from development to production deployment. EPJ Web Conf 214:04015. https://doi.org/10.1051/epjconf/201921404015
37. Davis M, Afonso J, Bachmann R, Bahýl V, Vera J, Leduc J, Cortés P, Rademakers F, Wardenær L, Yurchenko V (2024) The cern tape archive beyond cern an open source data archival system for hep. EPJ Web of Conferences 295. https://doi.org/10.1051/epjconf/202429501048
38. Sindrilaru E, Peters A, Adde G, Duellmann D (2017) Eos developments. J Phys Conf Ser 898:062032. https://doi.org/10.1088/1742-6596/898/6/062032
39. Google Developers: Protocol Buffers. https://www.developers.google.com/protocolbuffers
40. Weil SA, Brandt SA, Miller EL, Long DD, Maltzahn C (2006) Ceph: A scalable, high-performance distributed file system. Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), 307–320

41. Mkrtchyan T, Chodak J, Karimi M, Lueken R, Meyer S, Suchowski P, Voss C (2024) dcache integration with cern tape archive. EPJ Web of Conferences 295 https://doi.org/10.1051/epjconf/202429501016
42. The dCache Collaboration: dCache/dcache-cta: V0.14.0. https://doi.org/10.5281/zenodo.14421668
43. Barisits M (2020) ATLAS data carousel. EPJ Web Conf 245:04035. https://doi.org/10.1051/epjconf/202024504035
44. Morschel L (2020) Improving tape restore request scheduling in the storage system dCache. Bachelor Thesis. https://doi.org/10.3204/PUBDB-2020-01394
45. Morschel L, Chitrapu K, Garonne V, Litvintsev D, Meyer S, Millar P, Mkrtchyan T, Rossi A, Marina Sahakyan M (2021) Improving performance of tape restore request scheduling in the storage system dcache. EPJ Web Conf 251:02016. https://doi.org/10.1051/epjconf/202125102016
46. Allcock W. GridFTP: Protocol Extensions to FTP for the Grid (2003). https://ogf.org/documents/GFD.20.pdf
47. The WLCG DOMA Third Party Copy (TPC) working group: Third Party Copy. Onlinehttps://twiki.cern.ch/twiki/bin/view/LCG/ThirdPartyCopy. Accessed 7 Jan 2025
48. Ceccanti A, Vianello E, Caberletti M, Giacomini F (2019) Beyond x.509: token-based authentication and authorization for hep. EPJ Web Conf. 214, 09002 https://doi.org/10.1051/epjconf/201921409002
49. Raymond ES (1998) The cathedral and the bazaar. First Monday 3(3) https://doi.org/10.5210/FM.V3I2.578
50. The Kubernetes Authors: Kubernetes. https://kubernetes.io/
51. dCache developers: dCache Helm Cart. https://github.com/dCache/dcache-helm
52. GitHub, Inc.: GitHub Actions Documentation. https://docs.github.com/en/actions
53. The dCache collaboration: dCache 11.1.0. https://doi.org/10.5281/zenodo.17098358
54. Prometheus - Monitoring system & time series database. https://prometheus.io/
55. Schwarz K, Mkrtchyan T, Voss C, Sudeikat J, Koehler-Bussmeier M (2024) Self-Adaptive dCache. Technical report https://bib-pubdb1.desy.de/record/607523