Fermilab

Label-based Virtual Directories In dCache

FERMILAB-CONF-25-0940-CSAID

Label-based Virtual Directories In dCache

Marina Sahakyan[1,*], *Christopher* Green[2], *Dmitry* Litvintsev[2], *Krishnaveni* Chitrapu[3], *Lea* Morschel[1], *Tigran* Mkrtchyan[1], and *Svenja* Meyer[1]

[1]Deutsches Elektronen-Synchrotron DESY, Notkestraße 85, 22607 Hamburg, Germany
[2]Fermilab, PO Box 500, Batavia IL 60510-5011, USA
[3]National Supercomputer Centre in Sweden

Abstract. Traditional filesystems organize data in directories. These directories are typically a collection of files whose grouping is based on a single criterion, e.g., the starting date of an experiment, experiment name, beamline ID, measurement device, or instrument. However, each file in a directory can belong to several logical groups, such as a special event type, experiment condition, or a part of a selected dataset. dCache is a storage system developed to store large amounts of scientific data, used by many HEP and Photon Science experiments. With recent developments in dCache, we have introduced a concept of file tagging, which dynamically groups files with the same label into virtual directories. The file labels can be added, removed, renamed, and deleted through the admin interface or via REST API. The files in virtual directories are exposed through all protocols supported by dCache. This contribution will describe the details of the implementation for file tagging in dCache and present our future development plans on automatic metadata extractions, a feature that will significantly simplify data management. Additionally, we are exploring the future use of virtual directories as a way to translate scientific data catalogs into filesystem views for direct data analysis.

1 Introduction

The dCache project began in 2000 as a collaboration between Deutsches Elektron-Synchrotron (DESY) and Fermi National Accelerator Laboratory. Its goal was to develop a unified storage software solution for these laboratories, utilizing commodity heterogeneous disk servers as a caching layer in front of tape storage. Over time, dCache has gained significant popularity within the Worldwide LHC Computing Grid (WLCG), with numerous laboratories and universities adopting it. Collectively, these deployments account for approximately 50% of WLCG's total storage capacity.

Having been in production for over two decades, dCache is now widely used across the world [1]. More recently, it has been increasingly adopted by sites supporting scientific communities with requirements beyond those of LHC experiments. For instance, DESY now leverages dCache for applications in photon sciences, biology, future accelerators R&D, and other fields. From its inception, dCache has continuously evolved to meet the needs of emerging user communities and new workflows, implementing innovative solutions to enhance productivity [2].

*e-mail: marina.sahakyan@desy.de

Despite its maturity, dCache remains adaptable to evolving technologies and user demands. Beyond its role as a distributed storage system, it offers multiple access and authentication protocols tailored to diverse communities. It supports third-party copy for efficient data exchange between sites, provides access through both standard and HEP-specific protocols, and operates in heterogeneous environments, granting flexibility in hardware and OS selection. Its scalable architecture allows deployment on anything from a single node to hundreds, enabling seamless growth as needed.

dCache separates the file namespace of its data repository from the actual physical location of the datasets. The file names, attributes, and filesystem tree are managed in an internal database and exposed through a namespace component. By splitting a file's metadata and data, dCache uses a unique identifier for each file which is independent of the file's name and location.

The namespace/metadata component of dCache is built out of several layers. The top one called PnfsManager, is responsible for interactions between the rest of dCache and the underlaying filesystem backend. It communicates with the filesystem abstraction layer Chimera [3], which is built on top of a relational database using a thin Java layer.

Clients can then use NFS to mount the namespace locally, which allows the use of OS-level tools such as *ls*, *mkdir*, *mv* for Chimera. Direct I/O operations such as *cp* and *cat* are possible with the NFSv4.1 [4] door.

2 Motivation

Scientific data is not only the data itself, which is stored on disk, but also the associated metadata that describes the data. It may include the starting date of the experiment, the experiment name, beamline ID, measurement device, or instrument. Traditional file systems organize data in directories. Directories are typically a collection of files whose grouping can be based on some of the above mentioned criteria. However, each file in a directory can belong to several logical groups, such as a special event type or experiment condition. The same logical groups are then stored in two different directories. When they are required for the analysis, it might be difficult to search for files with a specific criterion, the more intuitive way in this case would be first tag the files based on metadata and later dynamically group files with the same label into virtual directories.

As it is shown in Figure 1 experimental data first is coming into a file system. Later on a researcher will need to search for files based on a certain experiment condition or outcome to perform his analysis. In this example, they want to analyze images captured of large blue birds that they collected during their experiment.

To find these specific images among all bird pictures they have to search through all directories, which are ordered by date, not the attributes of interest.

To address these challenges, dCache introduces a dynamic labeling mechanism that enhances metadata-driven file organization. By allowing users to attach labels to files, researchers can categorize data in a way that aligns with their workflows, creating virtual directories based on shared attributes rather than fixed locations. This approach enables efficient searching, filtering, and retrieval of relevant datasets without requiring predefined hierarchical structures.

3 Implementation

To implement the above mentioned functionality in dCache we have introduced user metadata handling. Any type of string label describing raw data, calibrated data or detector telemetry

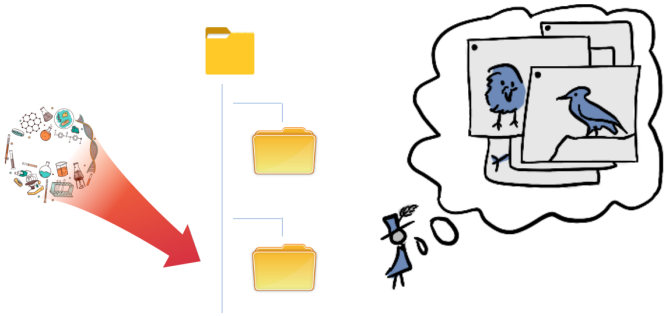

Figure 1. Experimental data is stored in file system and researcher is looking for blue-bird images.

can be attached to a file via the RESTful API, or via the graphical dCache-view interface [5]. Labels can be added to, renamed or removed from a file. It is also possible to query for all labels that a particular file has.

To implement the labeling functionality we have introduced two tables, t_labels and t_labels_ref to the database (Figure 3). The t_labels table has two fields, a unique, auto-incrementing label_id and a unique labelname, each describing unique id of the label and it's string value respectivly. The t_labels_ref table links label_id with inumber field. inumber (or inode number) is a unique identifier assigned to each file and directory within the filesystem. It is used to map relationships between labels and other entities in the database, for example, file's parent path or its corresponding metadata.

On the left side of Figure 2 researcher would like to categorize images stored in the file system by colors, for example *blue* and *brown*, as well as by the size of the image *large* and *small*.

A user may choose between the web interface and the REST API to add any label to a file. As illustrated in the center of Figure 2, where the researcher tags the file (/path-to-file/image.jpg) with the label *blue-bird* via our dCache view web interface. The same could be done using our REST API (the listing 1). Both methods are interchangeable and key advantage of this approach is flexibility. Labels can be added, removed, renamed, or deleted via an admin interface or a REST API.

On the right side of Figure 2, the file is now stored in the file system together with the tags. New entries for the newly added labels *large* and *blue-bird* will be created in tables t_labels and t_lables_ref. The manually added tag is now immediately visible as a virtual directory *blue-bird*.

Listing 1. Example of REST API call for adding tag to a file.

```
curl -X POST
https://host:portnumber/namespace/path-to-file/image.jpg
{
"action" : "set-label",
"label" : "blue-bird"
 }'
```

New entries were added to both tables t_labels and t_labels_ref, respectively, as illustrated in the tables Figure 3. t_labels maps the label_id field to a labelname filed (e.g., 1 -> large, 2 -> blue-bird). The labels are now assigned to specific files referenced by inumber (e.g., 1 ->

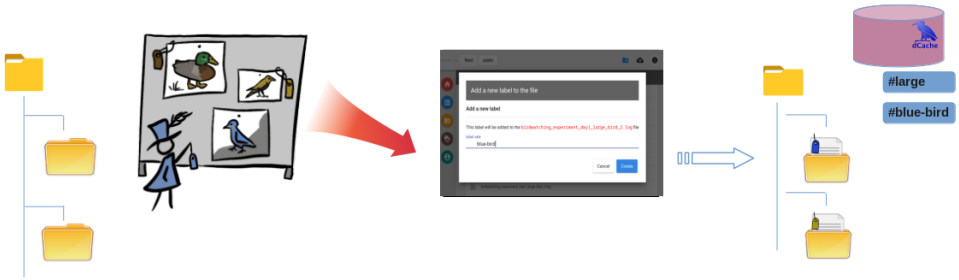

Figure 2. Manually added tags are immediately visible in as a virtual directory.

327, 2 -> 356), corresponding to file's identifier in other tables. For our use case, this is the table t_dirs, which stores hierarchical file relationships. As shown in Figure 3 the inumber field (356) of the table t_labels_ref is mapped to the t_dirs's ichild field (356), which is the unique identifier of the stored object *image.jpg*. The t_dirs table links directories inumber field (355) with ichild and iname elements, where iname is files's name, defining a directory structure. Using the ichild field it is possible to recursively trace back to the parent directories and for each found entry, prepend the directory name to the path with a preceding slash. For example, the path of the *image.jpg* file will be mapped by iname -> ichild, ichild -> inumber, inumber -> iname (image,jpg -> 356, 356 -> 355, 355-> path-to-file). The ichild element (355) in t_dirs connects to the inumber (356) in t_nodes table, which stores all file system objects with attributes. From the t_nodes table we get the unique id (ipnfsid -> 000092) of the file and its metadata, for example, creation time, user id or group id.

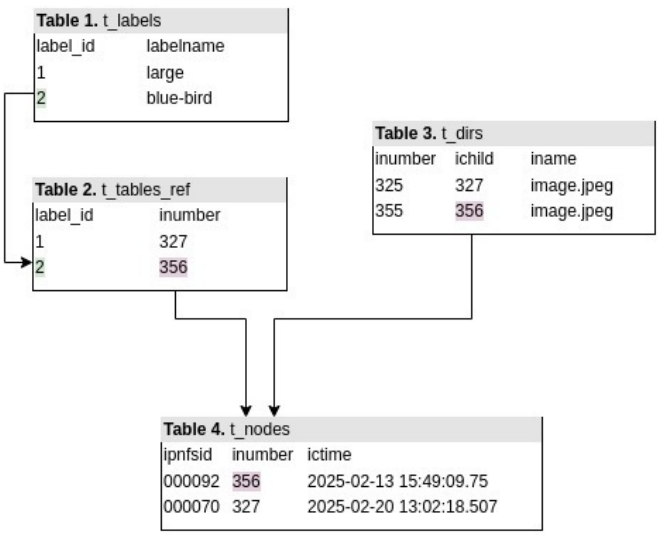

Figure 3. t_labels and t_lables_ref relationship.

It is worth noticing, that in the cases when the same label is attached to two different files with the same name as it is shown in table Figure 3, the correct path will still be selected because t_dirs links directories inumber with child elements (ichild, iname) (355 -> 327, image.jpg), defining a directory structure.

The traditional file system [6] and the new label-based view are both visible at the same time. The user now has the possibility to dynamically group files with the same label into virtual directories independent of the actual file location. These virtual read-only directories are dynamically populated with files as soon as the files get the corresponding label. Those virtual directories are exposed through all protocols supported by dCache.

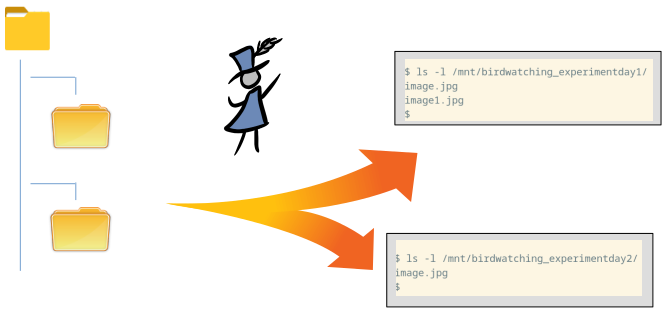

Figure 4. Listing contents of a traditional file system using *ls*, based on hierarchical directory organization.

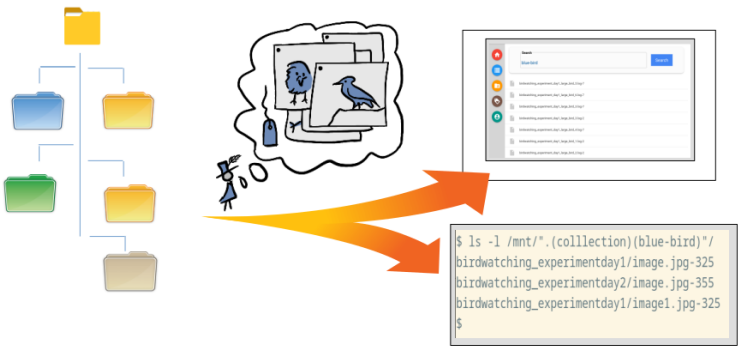

Figure 5. Listing contents of a virtual label-based directory using *ls*, showing dynamic label-driven organization.

Figures 4 and 5 illustrate the difference between traditional hierarchical file organization and a dynamic metadata-based file retrieval system. In Figure 4, multiple directory structures represent conventional file storage, where files are categorized into nested folders based on predefined attributes. This approach makes it difficult for users to locate specific files without precise knowledge of their location.

As Figure 5 shows, instead of organizing files strictly within folders, the system now allows users to retrieve files dynamically based on metadata tags. The users can now search

for all files tagged with *blue-bird* or *large* (blue and green folders respectively) instead of manually navigating directories.

In Figures 4 and 5, we can see how the researcher envisions a more flexible approach to data retrieval, where files are grouped dynamically based on their attributes rather than static folder locations. The right side of the Figure 5 showcases two alternative directory listing mechanisms enabled by a metadata-driven system. The first is a graphical dCache web interface that allows users to query files based on specific tags such as *blue-bird*, making retrieval more intuitive and efficient. The second is a command-line search using structured queries, demonstrating how users can locate relevant files programmatically based on metadata attributes.

This approach significantly enhances usability, as files can be retrieved through multiple perspectives without duplicating data. By leveraging metadata tagging instead of strict directory hierarchies, researchers can improve data accessibility, streamline workflows, and enhance collaboration across different domains.

As mentioned above, a user can now look up tagged files. For example, via WebDAV by simply entering the tag *blue-bird* in the dCache view web interface or via NFS using the following command : *ls -l /mnt/".(collection)blue-bird"/.* As a result, the user receives a list of all files labeled with the *blue-bird* tag, regardless of their location in the file system. For example, *image.jpg-355* and *image.jpg-325*, where 355 and 325 are the parent directory IDs. This will link to the correct paths of each file when the user wants to download it. Additionally, this ensures data integrity when deleting a file or label.

4 Conclusion and Future Works

Traditional file systems organize data in fixed directories, making it difficult to retrieve files based on specific metadata attributes. To address this, dCache now allows users to attach labels to files, enabling dynamic grouping into virtual directories. These labels can be added, removed, and queried via the dCache REST API or graphical interface, significantly improving data accessibility.

The implementation relies on a structured database approach, linking labels to files through relational tables. This method allows researchers to search and retrieve data based on experimental conditions or attributes rather than rigid directory structures. The system supports multiple access protocols, such as NFS and WebDAV, ensuring seamless integration with existing workflows.

Future developments aim to automate metadata extraction and enhance catalog integration, further streamlining scientific metadata management.

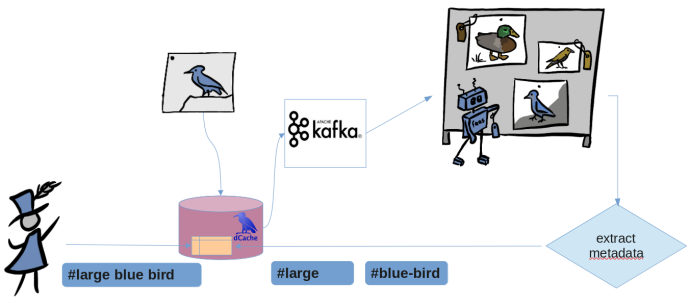

Figure 6. Automatic Metadata Population.

Metadata can be added or updated either manually or by an experiment's automation process. dCache events can trigger a so-called serverless function to execute community-specific applications that extract metadata and populate catalogs.

Figure 6 demonstrates an automated metadata extraction and cataloging workflow for data stored in a dCache. The process begins with data ingestion into dCache, which subsequently triggers an event in Apache Kafka [7] upon detecting new data. This event is processed by a Function-as-a-Service (FaaS) [8] system, which extracts relevant metadata. The extracted metadata can be used to automatically label that data in dCache and then then incorporated into a metadata catalog, allowing for efficient organization and retrieval. The workflow enables an automated update mechanism, ensuring that metadata is consistently synchronized with incoming data. This approach enhances data discoverability and facilitates streamlined data management in large-scale storage environments.

dCache events can trigger a so-called serverless function to execute community-specific applications that extract metadata and populate catalogs.

References

[1] P. Fuhrmann, V. Gülzow, dCache, Storage System for the Future, in *European Conference on Parallel Processing (Euro-Par 2006)* (Springer, 2006), Vol. 4128 of *Lecture Notes in Computer Science*, pp. 1106–1113

[2] A.P. Millar, T. Baranova, G. Behrmann, C. Bernardt, P. Fuhrmann, D. Litvintsev, T. Mkrtchyan, A. Petersen, A. Rossi, K. Schwank, dcache, agile adoption of storage technology, Journal of Physics: Conference Series **396**, 032077 (2012). 10.1088/1742-6596/396/3/032077

[3] T. Mkrtchyan, K. Chitrapu, D. Litvintsev, S. Meyer, A.P. Millar, L. Morschel, A. Rossi, M. Sahakyan, DB Back-ended Filesystem for Science, in *Proceedings of the 35th GI-Workshop Grundlagen von Datenbanken, Herdecke, Germany, May 22-24, 2024*, edited by U. Störl (CEUR-WS.org, 2024), Vol. 3710 of *CEUR Workshop Proceedings*, pp. 58–63, https://ceur-ws.org/Vol-3710/paper9.pdf

[4] J. Elmsheuser, P. Fuhrmann, Y. Kemp, T. Mkrtchyan, D. Ozerov, H. Stadie, LHC data analysis using NFSv4.1 (pNFS): A detailed evaluation, Journal of Physics: Conference Series **331**, 052010 (2011). 10.1088/1742-6596/331/5/052010

[5] dCache Project, dcache view repository, https://github.com/dCache/dcache-view, accessed: 2025-02-27

[6] A.S. Tanenbaum, Modern Operating Systems, 3rd edn. (Pearson Prentice Hall, Upper Saddle River, NJ, 2008), ISBN 9780136006633

[7] J. Kreps, N. Narkhede, J. Rao, Kafka: A distributed messaging system for log processing (2011), available: https://api.semanticscholar.org/CorpusID:18534081.

[8] M. Shahrad, J. Balkind, D. Wentzlaff, Architectural implications of function-as-a-service computing, Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (2019).