UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Traballo de Fin de Grao que presenta

D. Hugo González Labrador

para a obtención do Título de Graduado en Enxeñaría Informática

CERNBox:
Petabyte-Scale Cloud Synchronisation and Sharing Platform

Xuño, 2016

Traballo de Fin de Grao Nº: EI15/16-029

Titor/a: Arno Formella
Área de coñecemento: Linguaxes e Sistemas Informáticos
Departamento: Informática

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Leslie Lamport

Acknowledgements

I would like to thank all of the people who supported and helped me with the thesis and with the development of CERNBox.

First of all, I'd like to thank my supervisor PhD. Arno Formella for his guidance and his valuable feedback on my work. I would also like to thank him for giving me the freedom that I had with regard to the topic of the thesis and also for giving me the opportunity to work in his research group since three years ago. Furthermore, I want to thank him for his extraordinary observation of small details and also for having his door always open whenever I ran into problems. Second, I'd like to thank my CERN's supervisor, PhD. Jakub T. Mościcki, for his incredible support during my stay at CERN and afterwards. I would aso like to thank him for giving me the opportunity to present my work at international events and for his precious feedback on this thesis.

With regard to the development of CERNBox, I'd like to thank the whole CERNBox team, specially Kuba, Massimo and Luca, who have supported me since the first day and I really have no words to describe what they have meant to me. I would also like to thank the EOS team, specially Mr. EOS for his unlimited patience and hard work (and also for his English German-based lessons on how to pronounce 'Sssssssspanish'). Also, I would like to say a warm thank you to Steve, for making every day at CERN an adventure and 31/2-003 the best place to work (even with ancient furnitures, old blackboard, wasp nets, green lights, noisy pipes and small fans).
With regard to university, I would like to thank LIA2's group, specially Arno, Pili and Nany for their support since I joined university. I would also like to thank you the people of 36 and 34 (Santi, Juan, Loxo, Diego and Castor) for making every day at university an enjoyable experience full of anecdotes.
Furthermore, I'd like to say thank you to my friends, specially the ones of 'Los buitres', who have always supported me during my studies.

Last, but not least, a warm thank you to my family and girlfriend who have always being there for me: Naty, Pili and Maikel.

Contents

List of Figures

List of Tables

1 Background

At CERN (Organisation européenne pour la recherche nucléaire), physicists and engineers are probing the fundamental structure of the universe. They use the world's largest and most complex scientific instruments to study the basic constituents of matter — the fundamental particles. The particles are made to collide together at close to the speed of light. The process gives the physicists clues about how the particles interact, and provides insights into the fundamental laws of nature.

The instruments used at CERN are purpose-built particle accelerators and detectors, for instance the Large Hadron Collider (LHC) is the biggest. Accelerators boost beams of particles to high energies before the beams are made to collide with each other or with stationary targets. Detectors observe and record the results of these collisions. Founded in 1954, the CERN laboratory sits astride the Franco-Swiss border near Geneva. It was one of Europe's first joint ventures and now has 21 member states.

CERN is also the birthplace of the World Wide Web. The main site at Meyrin has a large computer facility containing powerful data processing facilities, primarily for experimental data analysis; because of the need to make these facilities available to researchers elsewhere, it has historically been a major wide area networking hub

The LHC experiments produce over 30 petabytes of data per year. Archiving vast quantities of data is an essential function at CERN. Magnetic tapes are used as the main long-term storage medium. Often people wonder why CERN still use tape, as it is an old-fashioned technology. Tape is actually a very sophisticated storage material and it can store huge volumes of data. For example, the data which was stored on thousands of reels for the 1990's OPAL[1] experiment now fit on one of today's cartridges. Tape is inexpensive, compact, doesn't consume much electricity, and is durable for long-term storage. With the data tsunami from the LHC, being able to quickly retrieve petabytes of stored data is essential for physicists to make groundbreaking discoveries. CERN has more than 130 Petabytes of stored data (the equivalent of 700 years of full HD-quality movies).

Besides tape infrastructure, CERN has a parallel data storage system called EOS, a disk-based service providing a low latency storage infrastructure for physics users. The main target area for the service are physics data analysis with use cases often characterised by many concurrent users.

CERN IT-DSS (renamed to IT-ST in 2016) is the group that ensures a coherent development and operation of the storage services at CERN for all aspects of physics data. The group is divided into three sections: IT-DSS-TAB (Tapes, Archives and Backups Section), IT-DSS-FDO (File systems and Disk Operations Section) and IT-DSS-DT (Design and Transitions Section).

The FDO section operates and supports the storage and file system services for physics. During the year 2015 I was working inside this section as a Technical Student under the supervision of PhD. Jakub T. Mościcki for the CERNBox project, core of this thesis.

During this period at CERN, I was involved into the whole lifecycle of the CERNBox service, from requirements, design, implementation, deployment, and user support. My main role was the integration of an open source component called ownCloud with EOS, the disk storage for the LHC, to provide synchronisation and share capabilities for scientific and engineering use-cases. Besides pure development, I have also participated in various events: I was invited by ownCloud to their development event [10] to give a presentation to their development teams about scalability problems found on ownCloud, I have spoken at CERN IT Auditorium a couple of times [9, 11, 12] to present the CERNBox service and I collaborated in other international

[1] OPAL was one of the major experiments at CERN before LHC era.

conferences [15]. As parallel tasks to the main job, I acted as a technical advisor to ownCloud , as an active member on the Géant group dedicated to the development of the OpenCloudMesh[2] project and as an active member on the IETF Internet Storage synchronisation mailing list[3]. I was also co-author of a couple of technical publications [14, 13] about CERNBox and CERN storage during this period and I also helped to organise the first Workshop for Cloud Synchronisation and Sharing Services held at CERN on November 2014.

2 Introduction

Cloud computing introduces a different way to communicate to computer services, from email communications to the usage of social networks. Within this paradigm, the heavy and hard job has been moved from the client to remote servers, converting the client into a thin client for promoting mobility and network pervasiveness.

Cloud storage is a special type of cloud service with the primary goal of providing a pervasive and coherent access to data from a variety of devices. The usage of cloud storage services like DropBox has been very successful because they provide a convenient way for people to synchronize and share their data across different devices. However, the adoption of these proprietary solutions has some potential problems:

- Off-premise deployments: data is outside the control of your organisation with somewhat unclear business and security policies.
- Closed source code: the application logic is hidden, without the possibility to customize or optimize the code base.
- Unclear service level: the data confidentiality and jurisdiction may be at risk.

To handle these problems, in the last five years, some open source solutions came to play the synchronisation and share game. The most popular products are ownCloud, SeaFile, PowerFolder and Pydio, which provide open source and on-premise solutions to online cloud storage. These products offer the synchronisation and share layer on top of different storage backends like local storages, network storages like NFS or object storages like Amazon S3.

A problem that has not yet been solved by any open or closed source project is the possibility to give the end user a performant and integrated access to existing data repositories without having to migrate the data or replacing the user work flows.

In order to clarify this problem to the reader Figure 1 shows the architecture found on most synchronisation platforms of today like DropBox or ownCloud.

The usual work flow in this scenario is the following:

1. The user accesses her data interacting with the server using a wide range of available devices like smart phones, desktop synchronisation clients, or a browser.
2. The data is saved to the storage for future use.

[2] https://oc.owncloud.com/opencloudmesh.html
[3] https://datatracker.ietf.org/wg/iss/charter/

Figure 2 shows an advanced architecture, where the storage can be accessed by the user and also by third party services.

In this scenario, the previous use case is still there (see green circles), but also a new use case that is explained below using the infrastructure common at CERN.

1. Other service unrelated to the synchronisation and share server, in this example the data pumping service of the Atlas detector, writes data to the storage.
2. The end user, a physicist, uses analysis tools to interpret the raw data kept in the storage to create new results.
3. The synchronisation and share server connects to the same storage used by the detector.
4. The results of the analysis will appear automatically in the synchronisation clients of the user. Moreover, now the user can share its data with his colleges.

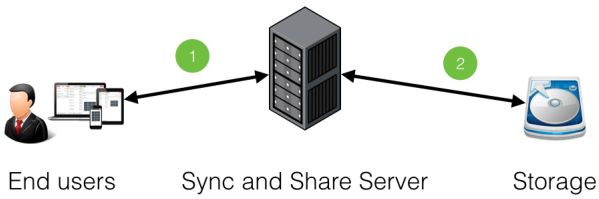


Figure 1: Common architecture of a synchronisation and Share Platform

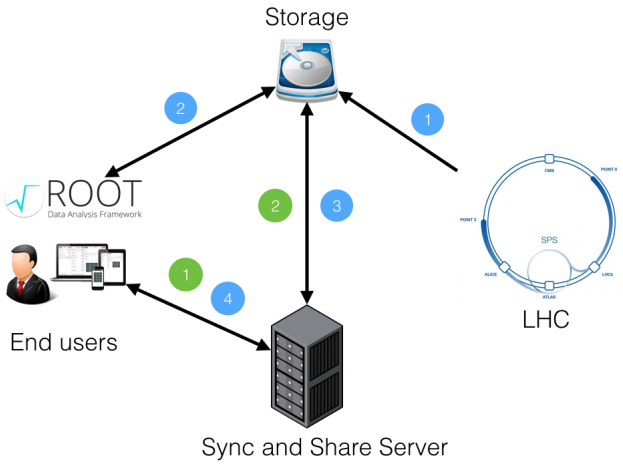


Figure 2: Advanced architecture: CERNBox

The next sections will focus on the challenges that need to be addressed in order to move from the basic architecture of Figure 1 to the advanced architecture found on Figure 2.

3 Related work

3.1 ownCloud

At the time of the discussion to chose the FSS platform (2014) for the CERNBox project, the following open source projects were analysed: ownCloud, SyncAny, Pydio, SeaFile and PowerFolder.

Pidyo was discarded due to the lack of synchronisation and mobile clients at that time, SyncAny did not have a web application and the project looked like dead at that time, SeaFile's git-based synchronisation algorithm was too complex to implement in EOS and PowerFolder was only partial open source. ownCloud was the most promising option as it had desktop synchronisation clients for all major operative systems, a web application and an active and big open source community.

Another reason for this selection was due to the feedback received from a survey done by SWITCH[4] among its users to choose their favourite open source alternative to DropBox, and ownCloud was the winner.

Yet another factor that contributed to the election of ownCloud was the small investment needed for having enterprise support and licenses for up to 10000 users. The cheap cost for the enterprise version of the software is due to the agreement between ownCloud and GÉANT[5] in order

[4]SWITCH is the swiss national research and education network organisation, SWITCH also manages the educational networks among Swiss universities and research facilities, and the links to other (non Swiss) university networks.

[5]GÉANT is the pan-European data network for the research and education community.

to boost research facilities and universities that rely on ownCloud as their on-premise file synchronisation and share solution with the ability to protect sensitive data or research and to manage data at scale.

3.2 Synchronisation

The number of File synchronisation and Share (FSS) services has grown over the last years and the product catalog is now wider and more diverse than ever. For such reason the synchronisation with these cloud providers is done with multiple synchronisation clients. For example, to synchronise data with DropBox one has to use the DropBox synchronisation client, to Google Drive with the Drive client and to ownCloud with ownCloud clients and so on. Each software vendor has its own synchronisation protocol, thus making it very difficult to use one synchronisation client to talk to multiple cloud providers with the same underlying synchronisation protocol. Over recent years there have been studies that analysed these synchronisation protocols [4, 2, 3, 5, 16, 1].

The ownCloud Synchronisation Protocol (OSP) is a good candidate for cloud file synchronisation because it relies on the well known WebDAV standard [21, 8]. It is documented[6] and compliant with the protocol so it can be tested with the SmashBox tool[7].

The OSP is a state-based synchronisation protocol with a cycle of three clearly defined phases [7] that are triggered periodically:

- Discovery phase: its objective is to collect metadata on the remote server to build a tree of remote changes to be compared with the local tree that represents the state of the synchronisation client.
- Reconciliation phase: given the remote and local trees, the ownCloud synchronisation Client has to decide what to do with each file based on the collected metadata. The result of this phase is a propagation list that specifies what to do to make the two sets the same.
- Propagation phase: it is the how-to-do-it phase. This phase executes the jobs to achieve the required changes specified in the propagation list.

OSP uses WebDAV as the underlying protocol for synchronisation but it has been extended with new attributes to improve efficiency. Theses extensions are as follows:

- File ID: it is an unique identifier that is attached to a resource (the concept is similar to an inode number — it points to an structure that represents a filesystem object) and it does not change when resources are moved to another location. This identifier is needed to track remote moves in the synchronisation clients to perform a local rename instead expensive delete and download operations.

It interconnects national research and education networks (NRENs) across Europe, enabling collaboration on projects ranging from biological science to earth observation and arts and culture.

[6] OSP, Jakub T. Mościcki and Klass Freitag, https://github.com/cernbox/smashbox/blob/master/protocol/protocol.md

[7] Smashbox is a framework for end-to-end testing of the core storage functionality of an ownCloud-based service installation through the ownCloud Synchronisation Protocol.

- ETag: it is an opaque unique identifier that is given to all resources to identity versions of the resource. As a consequence, when a resource ever changes, it is assigned a new ETag that has not been previously used. Whenever an ETag changes, all the ancestor directories up to the home directory of the user update their own ETag to reflect the path of changes. This propagation allows clients to construct a partial tree with just the changed branches.

- Mtime: the modification time of a resource. This attribute is also propagated like the ETag along the path to solve a corner case when a synchronisation client looses its synchronisation journal and has to decide which file to kept in case of conflict.

3.3 Sharing

The other dimension of a FSS platform is Sharing. These platforms usually offer two ways of sharing data: by a public share (also known as link share) or by internal sharing.

Public sharing means creating a link to a particular resource (either a file or a directory) that is sent to other people usually not registered in the FSS platform. This way of sharing is very useful to export internal data to third parties without granting them full access to the FSS platform. On the other hand, internal sharing only allows share to users of the FSS. This latter form of sharing usually allows to share to groups, increasing the benefits of collaboration between departments or work groups. Both forms of sharing allow to share a resource, but what happens when the resource is moved or renamed? This question highlights the decision that FSS platforms must deal with, and there are two possibilities to address it:

- DropBox allows you to share a file or folder, but as soon as you move or rename it the share is lost[8], consequently having to re-share the moved resource with previous users. This approach reduces user usability but increases the scalability of the system as it does not have to update share information after a rename or move.

- Google Drive and ownCloud use a permanent file identifier for the resources, therefore the resource is referenced by an id and not by path like in the previous case, thus renames or moves do not lose the share. This approach increases usability but decreases performance, as the system has to update the share information in an atomic way when the resource is moved or renamed.

4 CERNBox

CERNBox is a cloud synchronisation service for end-users: it allows syncing and sharing files on all major mobile and desktop platforms (Linux, Windows, MacOSX, Android, iOS) aiming to provide offline availability to any data stored in the CERN EOS infrastructure. There is a high demand in the community for an easily accessible cloud storage solution such as CERNBox. Integration of the CERNBox service with the EOS storage back-end is the next step towards providing 'synchronisation and sharing' capabilities for scientific and engineering use-cases. CERNBox achieves the integration of 'synchronisation and sharing' capabilities with the LHC data analysis tools and transfer services. IT-DSS introduces CERNBox, a synchronisation and sharing service, to promote a new integrated way of accessing data for scientific research.

[8] Why did my shared link stop working?, https://www.dropbox.com/help/45

The core of CERNBox is a synchronisation and sharing system based on components provided by the ownCloud open software stack. CERNBox provides a service layer on top of storage systems already provided by IT-DSS. This allows coherent data access policies to be applied and for service levels compatible with CERN standards to be defined. CERNBox offers various modern ways to access cloud storage:

- FUSE access[9]: the storage can be mounted as a filesystem. This is very important for computers inside the computer centre that do not have graphical interfaces but do a lot of batch job processing.
- WebDAV[10]: the storage is accessible through WebDAV enabled clients like Finder for OSX, Windows Network Drives for Windows or third-party tools like CyberDuck.
- XROOTD[11]: the storage can be accessed through high optimised data transfer protocols for doing efficient data analysis.
- synchronisation client: this is a component, integrated in the user desktop environment, which keeps one (or more) local folders in synchronisation with the central storage server. Users can work on their devices without connectivity and the synchronisation client reconciles changes when the network connectivity is restored. This component is the one that enables the access to offline data with eventual consistency[12].
- Direct web access: via any web browser the user can upload/download/remove files and share them with other users. Some extensions (available as application plugins in the ownCloud server) provide added functionality such as easing the access to certain types of files (e.g. image viewers).
- Mobile devices: given the fact that most web traffic comes from mobile devices, access to data from these devices is a must. CERNBox provides mobile and tablet applications for Android and IOS.

In 2014 a pilot service was deployed (NFS storage as backend) which was operated with minimal operational effort. This service demonstrated a growing demand for file synchronisation and sharing, and reinforced the assumption on the potential popularity of such a service. It was observed a constant increase of the number of users of the pilot service (several tens of new users weekly) both from the physics community and from other communities such as the general laboratory services and the accelerator departments. The type of usage was consequently broad, ranging from the synchronisation of data files (ROOT analysis files), arbitrary work files to office documents shared with colleagues.

[9] Filesystem in Userspace is an operating system mechanism for Unix-like computer operating systems that non-privileged users create their own filesystems without editing kernel code. This is achieved by running filesystem code in user space while the FUSE module provides only a bridge to the actual kernel interfaces.

[10] Web Distributed Authoring and Versioning (WebDAV) is an extension of the Hypertext Transfer Protocol (HTTP) that allows clients to perform remote Web content authoring operations.

[11] XRootD software framework is a fully generic suite for fast, low latency and scalable data access, which can serve natively any kind of data, organised as a hierarchical filesystem-like namespace, based on the concept of directory

[12] Eventual consistency is a consistency model used in distributed computing to achieve high availability that ,informally, guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

The next step is to federate CERNBox with CERN disk storage system, EOS. This integration effort is the core of this thesis. The current usable capacity of EOS is about 70 PiB with proven capabilities in terms of performance (designed to support LHC analysis) and reliability (geographically distributed replicas). CERNBox could become the main entry point for the entire CERN data store: users will selectively access or synchronise data with their devices while their entire data storage is accessible online by the batch jobs for massive data processing or for the creation of large data archives. The choice of EOS allows for the distribution of large user quotas (presently 1 TiB per user). A preliminary survey of use cases suggests that users at CERN will benefit (on top of the standard synchronisation and sharing) from the seamless access of data at different levels (home directories, group spaces) in a shared way. The "disconnected operations" provided by the synchronisation client is an important feature. In the future other activities may benefit from a synchronisation client becoming a one-way synchronisation agent between the data acquisition and the central services. For example small experiments need to develop ad-hoc scripts and procedures to upload data from their acquisition system to the CERN central data store: a one-way synchronisation client will be a fundamental simplification. One should note that the current client synchronisation behaviour does not yet support this mode (removal of files from the client will remove the same files in the central repository). These are general use cases and are more widely applicable than in the High-Energy Physics workflows. CERNBox can satisfy them in a pretty straightforward way: pictures from surveys in the field, readings of devices or upload of data from a remote machine (sensors, etc).

The ownCloud software provides the possibility of using any filesystem (and more recently some object store systems) for hosting user data. Files are stored in a straightforward directory structure. Metadata (in particular, the one controlling the synchronisation) are stored in a relational database which ultimately controls the behaviour of the synchronisation clients. In its core implementation ownCloud assumes that the backend filesystem is visible only via the ownCloud servers: this is clearly incompatible with allowing users to directly access their data (via POSIX commands like cp, mv and rm). ownCloud also allows an 'External Storage Access', unfortunately this solution does not scale well: the whole directory tree on the external storage must be scanned periodically for updates and integration of independent storage systems in the way attempted by ownCloud is operationally complex. The need of a relational database is also incompatible with a scale-out system of the size that is needed at CERN.

However, because of the state-based nature of ownCloud synchronisation algorithm and straightforward filesystem organisation in the ownCloud backend server, it is possible to adapt EOS to conform to the ownCloud synchronisation protocol.

The solution to the database problem is simply to remove it and delegate its functionality to EOS itself, since EOS is a shared filesystem with a hierarchical namespace. The key piece of information for the ownCloud synchronisation algorithm is a unique tag (called ETAG) to identify file updates and detect conflicting changes. This is now coherently handled and provided as an extended attribute by the EOS namespace (as a combination of inode number and the content checksum). It is important to highlight that the synchronisation client is left completely unchanged to not break the compatibility with standard clients developed by ownCloud for several platforms. The ownCloud web server has been adapted using a set of plugins to integrate it with EOS and the performance has been increased thanks to modifications to the core. These modifications will be explained in detail in next sections as they are the fundamental sections of this thesis: design of a petabyte-scale synchronisation and share platform.

EOS presents a WebDAV interface (with ownCloud protocol extensions) to the synchronisation clients. The data streams (file upload and download) are redirected to EOS storage nodes while the metadata operations (file removal, directory rename, etc) are handled by the EOS head node. This 'out-of-band' handling of traffic solves the potential data transfer bottleneck of having all transfers channeled through the ownCloud server. The ownCloud server is used to manage sharing and web access exclusively.

Last but not least, CERNBox may help a variety of different profiles at CERN:

- Administration: people working in non-technical areas like financial, human resources and legal departments. These people will get the most out of CERNBox thanks to the sharing capabilities in order to have a document collaboration service and inter-department document exchanges.
- Engineers: people working in technical areas like civil, mechanical, or technology departments. These people are abroad most of the time, so they will benefit from the offline capabilities of the service, allowing them to use their data without the need of network connectivity.
- Physicists: people working on theoretical or applied physics. It will contribute to the collaboration within different physic groups but also to make easy to obtain results directly from the experiments, enhancing the productivity via novel workflows thanks to the combined capabilities of ownCloud and EOS. Integration of CERNBox with physics workflows will be covered in Section 5.3.

4.1 Problems To Solve

This section focuses on explaining in detail the problems that shall be addressed when moving from a common synchronisation and share architecture to an advanced one that exports the underlying storage to the end user.

Although CERNBox uses ownCloud, the design problems presented here are related to any synchronisation and sharing platform that uses a database for keeping the namespace, i.e; that uses an architecture similar to the one shown in Figure 3.

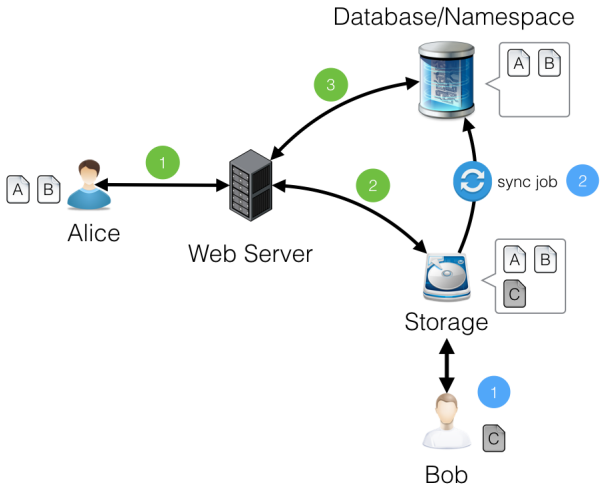


Figure 3: Basic architecture in detail showing the lacking of direct access to the storage

Inside this architecture there are three main components:

- A SQL database: the primary goal is to be used as a filesystem namespace.[13] When the storage is a local file system then the database is used as a cache, thus needing a synchronisation job that is explained later. On the other hand, when the underlying storage is an object store, it becomes a critical component, because if the database is lost, the user looses all his files.

- A Web Application Server: is the piece of the system that deals with synchronisation and sharing. It has a direct connection both to the storage and to the database, thus every data and metadata operation is sent to it. Furthermore, it enforces access control lists alongside the database to perform sharing.

- Storage: is the part of the system that keeps the user files, the data. Modern synchronisation and share platform can use a variety of different storages ranging from local filesystems to object storages. This flexibility if achieved using an abstract virtual filesystem interface[14].

The goal with CERNBox is to offer novel functionalities to the scientific community and to export the underlying storage to the user is a key challenge to achieve it.

Using out-of-the-box synchronisation and share platform this is not possible due to two main constraints inherent in these systems:

- Lack of direct access to the storage: if the storage is modified by external actors, then the database that contains the namespace will not be in synchronisation with the data kept in the storage. As a consequence of this, users will be presented with cached information, therefore time-dependant jobs cannot be done in this system.

  A workaround solution to limit this problem is to kept a background synchronisation process that aims to maintain both sets synchronised. As a result of this, the performance of the system will be degraded as the number of users and files start to increase. This workaround was not viable for CERNBox, a system that could reach up to 10000 users and handles petabytes of data.

- Missing Access Control List consistency: the web server and the database are responsible for enforcing the access to shared data, that is, to guarantee a security policy independently of the access path.

  When operations are done against the web server, the security policy is guaranteed. On the other hand, when the storage is accessed behind the web server, this policy is not fulfilled, creating an insecure system.

Figure 3 highlights the problem of lacking direct access to the storage through two use cases (green and blue circles).

[13] A filesystem namespace provides a level of indirection to route filenames with the same name to the correct storage, thus making it possible to distinguish between identical identifiers.

[14] A virtual filesystem is an abstraction layer on top of a more concrete filesystem. The purpose of a VFS is to allow client applications to access different types of concrete filesystems in a uniform way. A VFS can, for example, be used to access local and network storage devices transparently without the client application noticing the difference.

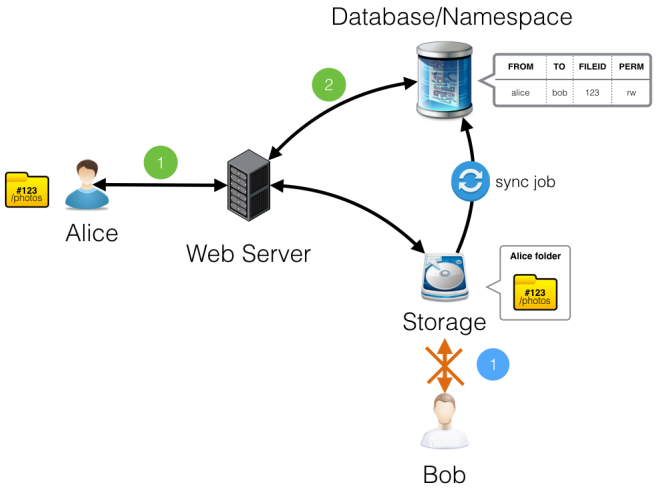


Figure 4: Basic architecture in detail showing ACL inconsistency problem.

The first use case (green circles) shows Alice uploading two files (A and B) through the web server. The web server will save these two files into the storage and then upload the information about the files (path, size, mtime ...) in the database. Notice here the non-atomic operation. After the operation finishes, files are stored both in the namespace and in the storage.

In the second use case (blue circles), Bob, a user able to access the storage by other means, puts a file C into the storage. After this file is put, Bob will not be able to see this file through the web server because a synchronisation job has to be triggered to insert it into the database at some point in the future. The synchronisation job is very inefficient as it has to scan all user storage and compare it with the state kept in the database to reconcile both sets of information.

The problem of inconsistent access control list is shown in Figure 4.

Alice (green circles) shares a folder ”/photos” with a fileID of 123 with Bob and the share information is kept in the database. Bob is only able to access Alice’s shared data through the web browser, as the storage has no notion of sharing.

A possible solution to these problems is presented in the next section.

4.2 A Solution

Figure 5 shows an alternative architecture to solve the lack of direct access to the storage.

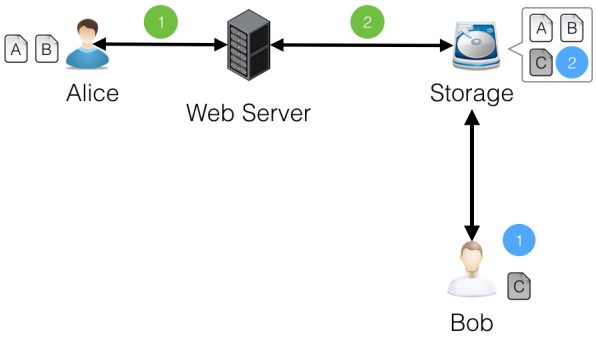


Figure 5: Advanced architecture in detail showing how the direct access to the storage problem is solved relying on the storage as the only namespace.

The main change presented in this architecture is the removal of the database that contains the namespace. As a consequence, the background synchronisation job is avoided. This produces a better experience to end users, as they will not see cached data anymore. Regarding Figure 5, Alice (green circles) uploads two files (A and B) through the web server. These files are saved just into the storage, that now is also a namespace. Now if Bob (blue circles) puts file C directly into the storage, then this file can be seen from the web server without having to wait for a synchronisation job.

The other problem to overcome is the access control inconsistency. Figure 6 shows the way this problem has been solved.

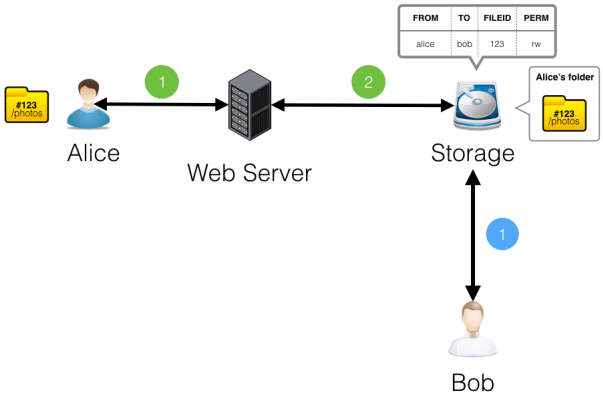


Figure 6: Advanced architecture in detail showing how the ACL inconsistency problem is solved using the storage as the security policy engine.

Alice (green circles) shares the folder ”photos” with fileID 123 with Bob. This information is keep in the storage. Bob (blue circles) is now able to access Alice’s folder from the storage. This is possible, because the storage is enforcing the security policy, thus allowing a homogeneous access to share data independently of the access path.

4.3 Architecture

CERNBox implements the advanced architecture shown in Figure 5 and Figure 6 using EOS and ownCloud.

Figure 7 shows a complete view of the architecture implemented in CERNBox.

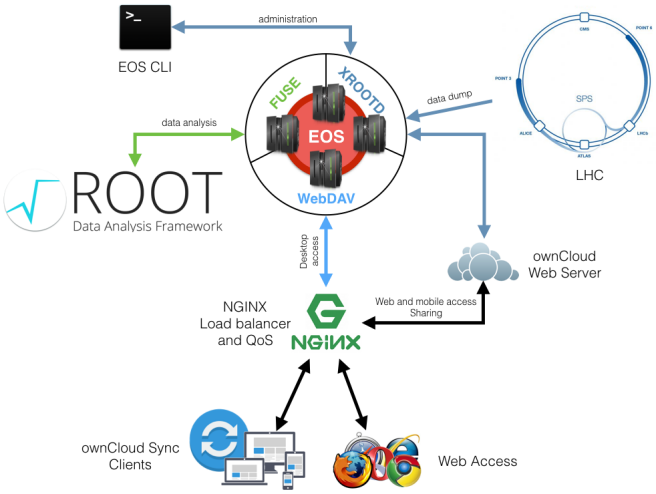


Figure 7: CERNBox architecture

This architecture implements the advanced model using the following components:

- EOS, the LHC Disk Storage: EOS is used as the storage for the CERNBox service. It provides a replicated data store with different ways to access the data: through a FUSE mount, through XROOTD protocol and through the WebDAV protocol. To have the same view from the synchronisation clients, from the web access and from FUSE or XROOTD clients, EOS storage extends the standard WebDAV protocol to include the synchronisation semantics used by ownCloud (see Section 3.2), therefore a change made using the EOS command line interface, the data dumped by the LHC or the results of an analysis will be synchronised automatically to an end user computer through ownCloud synchronisation clients.

- NGINX load balancer and Quality of Service Server: NGINX is used to apply some QoS criteria to synchronisation clients, web access, and WebDAV clients. The desktop synchronisation clients and WebDAV pure clients requests are redirected by NGINX directly to EOS to obtain the best performance for data synchronisation. Mobile devices, the web application and Share operations use the ownCloud REST API. As a consequence, these types of requests are redirected to the ownCloud server.

- ownCloud Web Server: the ownCloud stack is used to implement the web server proposed in the advanced model for a synchronisation and share platform. The ownCloud Web Server has a direct connection to EOS thanks to the development of a storage plugin that uses the EOS library to perform operations to EOS in an efficient way. The development of this plugin is explained in Section 4.4.

Besides the core components, Figure 7 also shows different clients making use of EOS as a shared storage system. The clients shown in the picture are the following:

- ownCloud Desktop Synchronisation Clients: these are the components needed to synchronise automatically the data kept on EOS to the end user local machine. They also

allow sharing of local files without having to share through the web application in latest versions. As these components are a critical part of the system extensive testing and validation of their behaviour have been carried out using the SmashBox Testing Framework developed as part of the CERNBox project.

- ownCloud Mobile Clients: these are the components that allow to access EOS's data from mobiles devices like smartphones and tablets. They do not synchronise data automatically as the storage capacity of these devices is less than the one expected for desktop machines.
- Web Application: the ownCloud server offers a web application to allow universal access to the data from any device anywhere to give an alternative to desktop or mobile applications. The web application also offers a set of highly useful applications like a text editor or an image viewer. As an example of this, a user modifying a text file in the browser will see these changes synchronised to all the clients he has.
- Pure WebDAV Clients: EOS offers a WebDAV interface extended with ownCloud semantics. Therefore, multiple existing DAV clients can be connected to EOS to access the data in a straightforward way. Commonly used clients are the ones already built in the Operative Systems (Finder for MacOSX and Network Drives for Windows) or third-party products like CyberDuck[15].
- EOS CLI: EOS provides a command line tool to interact with the data and access extra features like trash bin and version support among a vast range of other features. Usually, EOS administrators make heavy use of this tool to control EOS' behaviour: configuring replicas, master-slave layouts, setting user quotas or adding more space to the whole cluster.
- LHC: the LHC has a lot of services to deal with data dumping when collisions happen in one of the four main detectors (Alice, CMS, Atlas and LHCb). Some of these services dump data directly to EOS to save data at very high speeds. Usually, the LHC dumping services make use of high performance data transfer protocols like XROOTD to write data to EOS.
- FUSE access: EOS provides a FUSE interface, so users could mount a particular remote directory in EOS as a local folder in their workstation. The fact of having remote resources in a local folder allows the use of tools for data analysis only work in local filesystems. As a consequence, physicists could use analysis frameworks like ROOT to analyse the raw data that have been dumped from the collisions. This is one of the innovative use cases that CERNBox provides to the scientific community, the ability to interact with high valuable data in a non-disruptive and automatic way.

This section provides a high level view of the CERNBox architecture. Next sections will focus on addressing in detail the developments done to have the direct access to the storage and homogeneous access to shared data features.

4.4 Direct Access to Underlying Storage

The main development effort is integrating ownCloud, which provides synchronisation and sharing layers, with EOS, the storage backend. To have a direct access to the storage, EOS and

[15]Cyberduck is an open source client for FTP and SFTP, WebDAV, OpenStack Swift, and Amazon S3, available for Mac OS X and Windows (as of version 4.0) licensed under the GPL

ownCloud have been modified to support the use cases of a FSS platform. In the case of EOS, it has been modified to implement the ownCloud synchronisation protocol. See Section 3.2 and Appendix B to obtain more information about these modifications.

ownCloud has to be modified and extended in order to use EOS as the underlying filesystem. ownCloud was developed with the assumption of having always access to a local filesystem, but this is not always the case. On June 2014, ownCloud added support for remote object storages like OpenStack Swift and Amazon S3. EOS is neither a local filesystem nor an object storage, therefore using it out-of-the-box was not possible and ownCloud has to be modified to use it.

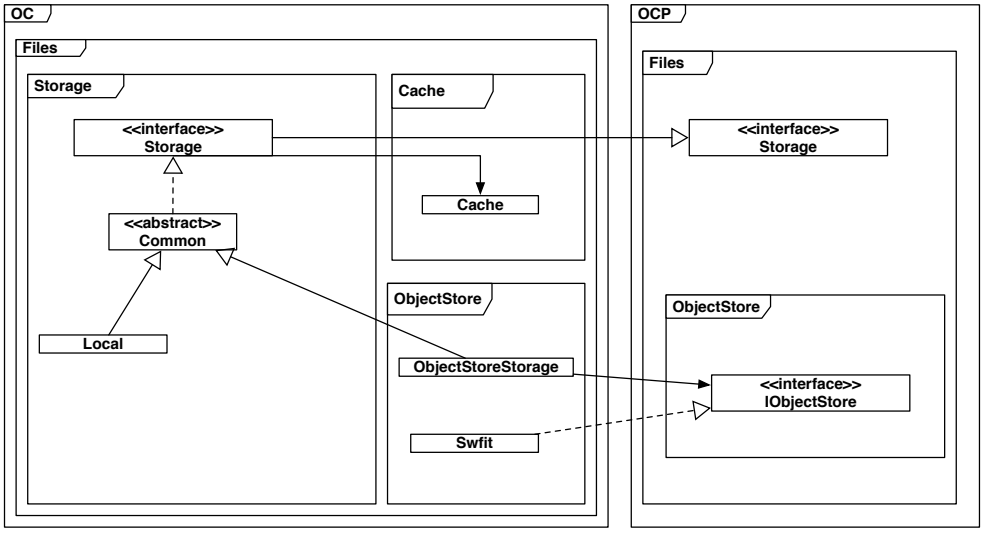


Figure 8: OC class diagram showing the classes involved when dealing with the storage

Figure 8 shows the different classes and interfaces involved in the ownCloud system regarding to the storage ('OC' refers to ownCloud private namespace and 'OCP' refers to public ownCloud namespace). The starting point in this diagram is the OCP\Files\Storage interface. This interface defines a series of methods to implement a filesystem (see Figure 41 for more details). The drawback of this interface is that it is based on the assumption of implementing only local filesystems, thus it specifies methods like 'fopen' and 'opendir', unconceivable when dealing with remote storages. This interface is then extended in the private \OC\Files \Storage \Storage interface class (see Figure 42). This interface mainly introduces the "getCache" method to retrieve the \OC\Files\Cache\Cache component that deals with the namespace in order to perform metadata operations. Notice that the Cache component is not implementing an interface neither living in the public namespace, therefore making it impossible to write custom Caches without patching the system. This Cache (see Figure 44 for more details) is designed to talk exclusively to SQL databases, making it more difficult to adapt it to other databases. The latter Storage interface is implemented partially in the abstract class \OC\Files\Storage\Common. This is the core class for the storage, as local filesystems or object store implementations are extensions of it. The class \OC\Files\Storage\Local implements a local filesystem and the class \OC\Files\ObjectStore\ObjectStoreStorage an object storage. Given the fact that OC\Files\Storage\Common implements a local filesystem, ownCloud decision was to implement an object storage on top of local filesystems. This is radically wrong as remote object storages do not have the extended capabilities of local filesystems. This

decision lead ownCloud having to adapt local filesystem methods like 'fopen' and 'opendir' to remote operations through inefficient hacks like staging files in a temporary storage.

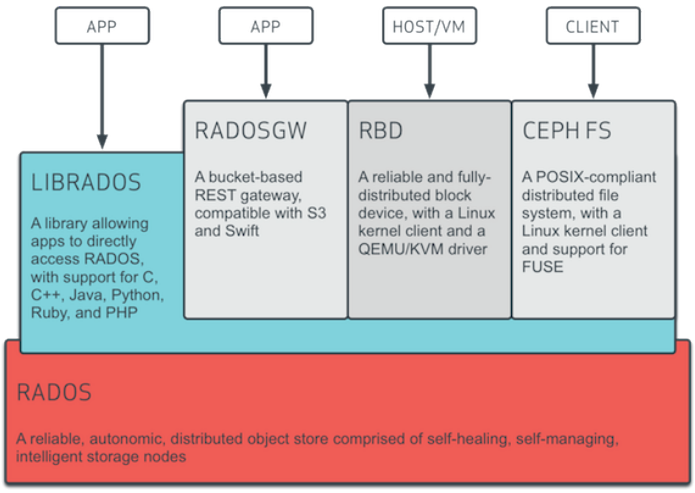


Figure 9: Ceph stack

An alternative architecture that ownCloud could use in future releases is the one used by Ceph[16] as shown in Figure 9. In this drawing the low level service is an object store with very basic operations like "readObject", "writeObject" and "listObjects". On top of this basic service S3 and Swift RESTFul APIs are created along with a POSIX compliant local filesystem.

Returning to the ownCloud storage discussion, the \OC\Files\ObjectStore\ObjectStoreStorage class uses an implementation of the \OCP\Files\ObjectStore\IObjectStore (see Figure 43 for extra details). Based on the design shown, EOS cannot be plugged in out-of-the-box implementing an interface or extending an existing class. The first problem is that EOS is not a local filesystem, therefore extending from \OC\Files\Storage\Common or \OC\Files\Storage\Local is worthless. The remaining alternative is to use the \OC\Files\ObjectStore\ObjectStoreStorage with a custom implementation of the \OCP\Files\ObjectStore\IObjectStore. From a preliminary point of view this looks like a posible solution, but there are two constraints with these classes:

1. Although the OC\Files\ObjectStore\ObjectStoreStorage class adapts local filesystem methods to remote ones using inefficient operations, it still relies on the \OC\Files\Cache\Cache class to perform metadata operations, thus imposing the system to use a SQL database to maintain the namespace.To achieve the direct access to the storage the database is removed from the whole architecture as shown in Figure 5.

2. The \OC\Files\ObjectStore\ObjectStoreStorage implements the "mkdir" operation talking just to the Cache without notifying the implementation of the \OCP\Files\ObjectStore\IObjectStore. This is okay for a pure object store but not for EOS, as EOS is a hierarchical remote shared storage.

[16]Ceph is an object storage based free software storage platform that stores data on a single distributed computer cluster, and provides interfaces for object-, block- and file-level storage. Ceph aims primarily to be completely distributed without a single point of failure, scalable to the exabyte level, and freely available.

To overcome these problems a series of patches are implemented to have direct access to EOS.

In the first place, the OC\Files\ObjectStore\ObjectStoreStorage has been modified to propagate methods like 'mkdir' to the \OCP\Files\ObjectStore\IObjectStore. In the second place, the latter interface has been extended with the 'mkdir' method and a custom implementation of the IObjectStore interface, \OC\Files\ObjectStore\EOS, has been created. Finally, the 'getCache' method of the ObjectStoreStorage class has been overwritten to a use custom cache, the \OC\Files\Cache\EosCache class. This class provides all the metadata needed by ownCloud from EOS (Figure 40 shows the metadata kept by ownCloud in the database). EOS can replace the ownCloud SQL namespace because it addresses the two requirements for an ownCloud namespace:

- Unique resource ID for all resources within the namespace: For being able to scale every resource (either file or folder) is identified by an unique resource ID. This ID is required to handle remote moves in the synchronisation client to avoid expensive delete and download operations. EOS can be configured to have unique IDs in the namespace (combination of inode and checksum), thus addressing the remote move problem.
- Propagation of resource modification bottom-up in the namespace: When a resource is modified, its modification must be propagated bottom-up to the top level directory to help the synchronisation client to scale (avoiding the recursive traversal of all the namespace for discovering changes). Usually, the attributes used for reflecting the modification of a resource are the mtime (modification time) or ETag, but a custom one can be used also. EOS can be configured to propagate modifications in the namespace.

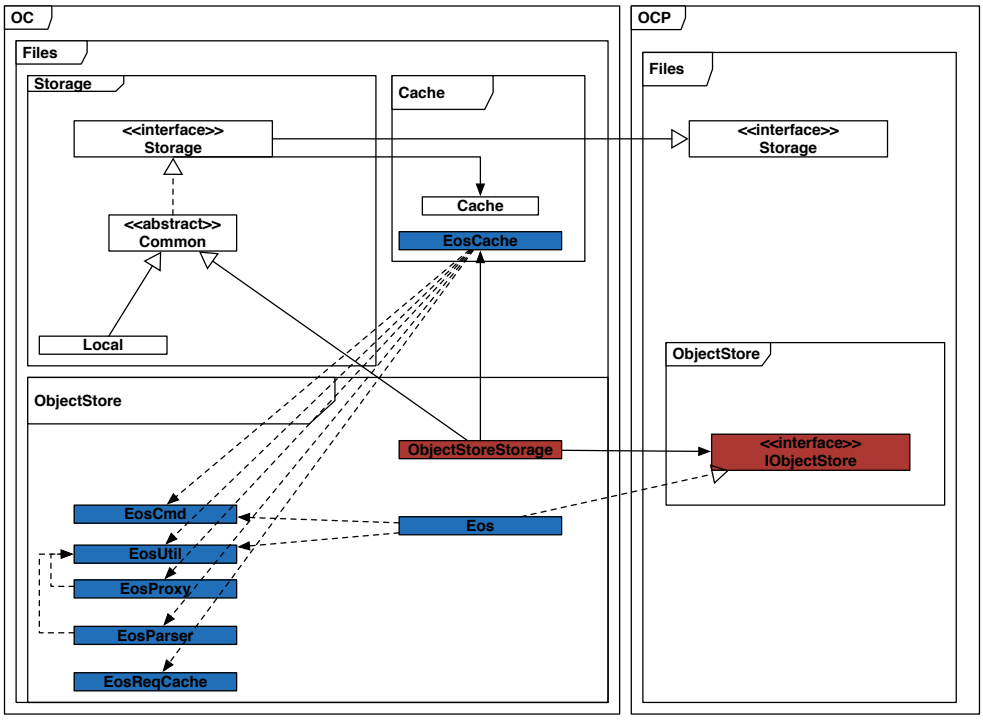


Figure 10: CERNBox class diagram showing the classes involved when dealing with the EOS storage

The design behind CERNBox storage model is shown in Figure 10, where blue boxes represent new classes and red boxes represent ownCloud classes that must be modified to use EOS. This diagram has several new classes:

1. \OC\Files\Cache\EosCache (see Figure 46 for further details): this new class is responsible for dealing with metadata operations against EOS instead of connecting to a SQL database. The connection to EOS MGM is done using system calls to the EOS CLI.

2. \OC\Files\ObjectStore\Eos (see Figure 45 for further details)): this class implements the modified
\OCP\Files\ObjectStore\IObjectStore interface to make data transfers in and out to EOS. It uses system calls to connect to the 'xrootd' binaries to start data transfers.

3. \OC\Files\ObjectStore\EosParser (see Figure 47 for further details): this class is responsible for parsing the stdout and stderr of system calls to EOS to marshal returned data into PHP data types.

4. \OC\Files\ObjectStore\EosProxy (see Figure 51 for further details): this class is responsible for translating ownCloud's paths like 'files/photos' to EOS's paths like '/eos/users/l/labrador/photos' and viceversa.

5. \OC\Files\ObjectStore\EosCmd (see Figure 48 for further details): this class is responsible for making system calls to the EOS CLI binary.

6. \OC\Files\ObjectStore\EosReqCache (see Figure 49 for further details): this class implements a cache that is valid for one request. This class caches expensive calls to EOS to avoid network round trips.

7. \OC\Files\ObjectStore\EosUtil (see Figure 50 for further details): this class contains helper functions to support the EOS integration.

   Figures 11, 12, and 13 show the differences of performing namespace operations using the ownCloud SQL namespace and CERNBox EOS namespace from a high level perspective.

ownCloud SQL Namespace	CERNBox EOS Namespace
SELECT `fileid`, `storage`, `path`, `parent`, `name`, `mimetype`, `mimepart`,`size`, `mtime`,`storage_mtime`, `encrypted`, `unencrypted_size`, `etag`, `permissions` FROM `*PREFIX*filecache` WHERE `storage` = storageOf(hugo) AND `path_hash` = hashOf(/hugo/photos)	eos -r hugo it file info /eos/example/h/hugo/photos -m

Figure 11: Differences between ownCloud and CERNBox on getting metadata by path

ownCloud SQL Namespace	CERNBox EOS Namespace
SELECT `fileid`, `storage`, `path`, `parent`, `name`, `mimetype`, `mimepart`, `size`, `mtime`,`storage_mtime`, `encrypted`, `unencrypted_size`, `etag`, `permissions` FROM `*PREFIX*filecache` WHERE `fileid` = 123	eos -r hugo it file info inode:123 -m

Figure 12: Differences between ownCloud and CERNBox on getting metadata by ID

ownCloud SQL Namespace	CERNBox EOS Namespace
SELECT `fileid`, `storage`, `path`, `parent`, `name`, `mimetype`, `mimepart`, `size`, `mtime`, `storage_mtime`, `encrypted`, `unencrypted_size`, `etag`, `permissions` FROM `*PREFIX*filecache` WHERE `parent` = idOf(photos) ORDER BY `name` ASC'	eos -r hugo it find /eos/example/h/hugo/photos --fileinfo --maxdepth 1

Figure 13: Differences between ownCloud and CERNBox on getting contents of a folder

4.5 Homogeneous Access to Shared Data

To achieve the architecture presented in Figure 6, ownCloud has to be modified to propagate share information to EOS to have an homogeneous access to shared data.

ownCloud keeps share information in an SQL table with the schema shown in Figure 14. The meaning of each field is as follows:

- id: it is an unique identifier to reference the share.
- share_type: specifies if the share is a link share or internal share (either individual or group share).
- uid_owner: the user who shared the resource.
- parent: ownCloud supports re-sharing, thus this identifies the parent of the share.
- item_type: specifies the type of resource like file or directory.
- item_source: references the file ID of the storage. ownCloud uses a permanent ID for shares (see 3.3).
- item_target: the name users receiving the share will see.
- file_source: idem to item_source [17].
- file_target: idem to item_taget.
- permissions: specifies reading and writing permissions for the share.
- stime: specifies the time the share was created.
- accepted: indicates if the share was accepted or not [18].
- expiration: specifies the due data for link shares.
- token: specifies the secure unique token for a link share.
- mail_send: indicates if an email was sent after sharing a resource.

[17] ownCloud allows to share calendars and other abstract resources, but these are not considered in CERNBox.

[18] ownCloud has a special share type called Federated Cloud share, that is a link share between two ownCloud instances, hence the requirement to accept or deny a share in the remote instance.

oc_share		
PK	id	INTEGER(11)
	share_type	SMALLINT(6)
	share_with	VARCHAR(255)
	uid_owner	VARCHAR(64)
	parent	INTEGER(11)
	item_type	VARCHAR(64)
	item_source	VARCHAR(255)
	item_target	VARCHAR(255)
	file_source	INTEGER(11)
	file_target	VARCHAR(512)
	permissions	SMALLINT(6)
	stime	BIGINT(20)
	accepted	SMALLINT(6)
	expiration	DATETIME
	token	VARCHAR(32)
	mail_send	SMALLINT(6)

Figure 14: ownCloud share table SQL schema

CERNBox uses this database table with a subtle change; because the data type used in EOS to keep track of the file IDs is an unsigned integer of 64 bits. The current ownCloud share schema has a data type of signed 32 bit integer for the file_source and item_source columns, so it would be truncated. To avoid this problem, the schema is modified to have an unsigned big integer for the file_source and item_source columns.

Besides storing the share information in these SQL tables, CERNBox propagates it to EOS to have a consistent view of shared resources independently of the access path (see Figure 6 to obtain an overview of the design). Unfortunately, ownCloud does not provide a plug-in mechanism to control share behaviour, thus having to modify the OC\Share\Share core class to achieve it. Figure 15 shows the core classes that were modified to propagate the information to EOS. Red boxes represent modified private core classes and blue boxes represent new classes (see Figure 50, 51 and 47 to get a detailed specification of EosUtil, EosProxy and EosParser respectively).

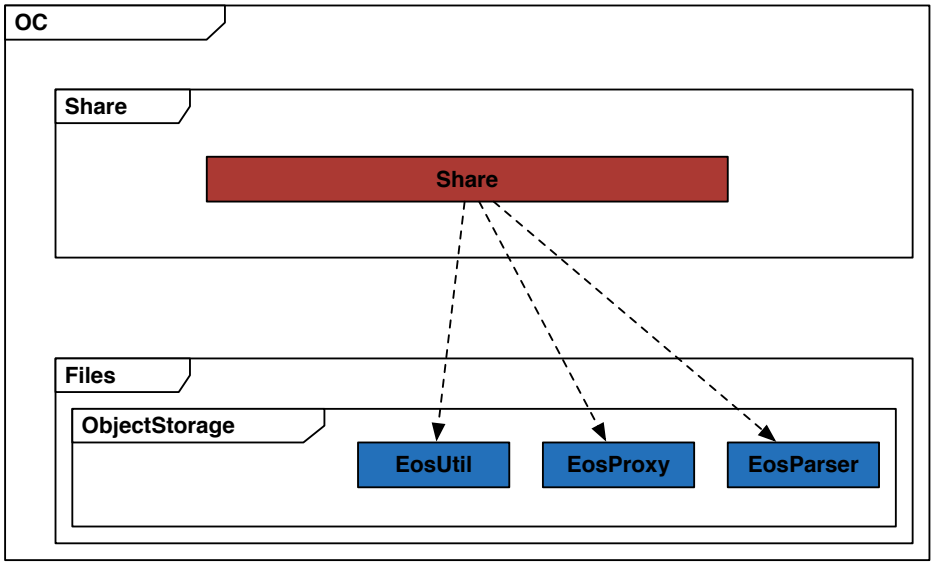


Figure 15: ownCloud share class diagram

The OC\Share\Share class has been modified to trigger share permissions kept in the SQL table to EOS ACL attributes. EOS ACls are defined only on the directory level via extended attributes[19]. An example ACLs could be 'sys.acl="u:hugo:rw, egroup:cernbox-team:r'. This ACl grants read and write permissions to user after each new share, update, or deletion EOS ACLs are updated accordingly through EOS commands ('eos attr set sys.acl="..." /eos/example/l/labrador/photos'). An important aspect of this design is that permissions are not read from the database, but from EOS, i.e 'eos attr ls /eos/example/l/labrador/photos'. This makes it possible to modify ACLs directly from EOS and see the results from CERNBox clients and the web application immediately. Figures 16 and 17 show how share ACLs are set and retrieved from EOS.

eos attr set sys.acl

```
EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> attr -r set sys.acl=u:labrador:rw /eos/eoslia/users/l/labrador/photos
success: set attribute sys.acl="u:labrador:rw" in directory /eos/eoslia/users/l/labrador/photos/
```

Figure 16: Setting share ACLs in EOS

eos attr ls

```
EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> attr ls photos
sys.acl="u:labrador:rw"
sys.forced.checksum="adler"
sys.forced.maxsize="10000000000"
sys.recycle="/eos/lia/proc/recycle/"
```

Figure 17: Obtaining share ACLs in EOS

[19] CERNBox only allows to share folders via internal sharing as EOS does not support ACLs on individual files.

4.6 Data Ownership

In a typical web application the user running the web server (usually apache or www-data) is the one that owns the files created through the web server. ownCloud runs on top of common web servers like Apache and Nginx, thus the ownership of files created in ownCloud are owned by the user running the web server. To increase security and give users full control of their data CERNBox files are owned by the logged-in user. Figure 18 highlights this difference.

ownCloud file ownership	CERNBox file ownership
root@e5ce26450a82:/var/www/html/data/test/files# ls -l drwxr-xr-x 2 www-data www-data 4096 Aug 16 12:31 Documents -rw-r--r-- 1 www-data www-data 2242192 Aug 16 12:31 manual.pdf	EOS Console [root://eosexample.cern.ch/] l/eos/user/o/ourense/> ls -l -rw-r--r-- ourense it 4096 Aug 16 12:31 Documents -rw-r--r-- 1 ourense it 2242192 Aug 16 12:31 manual.pdf

Figure 18: ownCloud and CERNBox differences in data ownership.

4.7 Native File Versioning

ownCloud provides an application for handling versions of files. Using this application, every time a file is updated, a version is created. The versioning application expires old versions automatically to make sure that the user does not run out of space. This pattern is used to delete old versions[20]:

- For the first 10 seconds ownCloud keeps one version every 2 seconds
- For the first minute ownCloud keeps one version every 10 seconds
- For the first hour ownCloud keeps one version every minute
- For the first 24 hours ownCloud keeps one version every hour
- For the first 30 days ownCloud keeps one version every day
- After the first 30 days ownCloud keeps one version every week

The versions are adjusted along this pattern every time a new version gets created. The version application never uses more that fifty percent of the user's currently available free space. If the stored versions exceed this limit, ownCloud deletes the oldest versions until it meets the disk space limit again.

ownCloud versions application does not take into consideration the underlying filesystem, therefore if the storage already provides file versioning, there would be two versioning processes running for the same purpose. EOS provides file versioning so CERNBox takes advantage of this instead using the generic Versioning App. To use EOS versioning instead ownCloud's, a new CERNBox Versioning application is developed using ownCloud's Application Framework[21].

Figures 19 and 20 show a high level overview of EOS versioning features.

[20] https://doc.owncloud.org/server/8.2/admin_manual/configuration_files/file_versioning.html

[21] https://doc.owncloud.org/server/8.0/developer_manual/app/index.html

eos versions list

```
EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> ls -la
drwxrwsr-+  1 root    root              0 Apr 25 13:24 .
drwxrwsr-+  1 root    root              0 Apr 25 13:24 ..
drwxrwsr-x  1 root    root              0 Apr 25 13:35 .sys.v#.passwd
-rw-------  1    root    root         2046 Apr 25 13:35 passwd

EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> file versions passwd
-rw-r-----  1 root    root         2046 Apr 25 13:29 1461583758.00000008
-rw-------  1 root    root         2046 Apr 25 13:35 1461584120.0000000f
-rw-------  1 root    root         2046 Apr 25 13:35 1461584153.00000010
-rw-------  1 root    root         2046 Apr 25 13:51 1461585099.00000019
-rw-------  1 root    root         2046 Apr 25 13:51 1461585101.0000001a
-rw-------  1 root    root         2046 Apr 25 13:51 1461585103.0000001b
-rw-------  1 root    root         2046 Apr 25 13:51 1461585104.0000001c
```

Figure 19: Listing versions of files using EOS versions commands

eos versions restore

```
EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> file versions passwd 1461583758.00000008
success: staged '/eos/eoslia/users/l/labrador/.sys.v#.passwd/1461583758.00000008' back to '/eos/eoslia/users/l/labrador/passwd' - the previous file is
now '/eos/eoslia/users/l/labrador/.sys.v#.passwd/1461584153.00000010
```

Figure 20: Restoring previous versions of a file using EOS versions commands

4.8 Native Recycle Bin

ownCloud provides another application for handling deleted files. Using this application, every time a file is deleted, it is moved to a special directory that acts like a trash bin[22]. To ensure that users do not run over their storage quotas, the Deleted Files application allocates a maximum of fifty percent of users currently available free space to deleted files. If deleted files exceed this limit, ownCloud deletes the oldest files (files with the oldest timestamps from when they were deleted) until it meets the storage usage limit again. ownCloud checks the age of deleted files every time new files are added to the deleted files. By default, deleted files stay in the trash bin for 180 days. The ownCloud server administrator can adjust this value in the config.php file by setting the 'trashbin_retention_obligation' value. Files older than the 'trashbin_retention_-obligation' value will be deleted permanently. Additionally, ownCloud calculates the maximum available space every time a new file is added. If the deleted files exceed the new maximum allowed space ownCloud will expire old deleted files until the limit is met once again.

Again, ownCloud Deleted Files application does not take into consideration the capabilities of the underlying filesystem. Modern filesystems (either local or object storages) offer some kind of recycle bin functionality, and EOS is not an exception. EOS provides first-class support for recycling files and Figures 21, 22 and 23 show a high level overview of its recycling features. To use EOS recycling features instead ownCloud's, a new CERNBox Deleted application is developed using ownCloud's Application Framework.

[22] https://doc.owncloud.org/server/8.2/user_manual/files/deleted_file_management.html

eos recycle ls

```
EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> recycle ls
# Deletion Time          UID     GID     SIZE        TYPE        RESTORE-KEY     RESTORE-PATH
# ================================================================================================
Mon Apr 25 13:27:24 2016   root     root     2046        file        0000000000000008 /eos/eoslia/users/l/labrador/passwd
```

Figure 21: Listing deleted files using EOS recycling commands

eos recycle restore

```
EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> recycle restore 0000000000000008
success: restored path=/eos/eoslia/users/l/labrador/passwd
```

Figure 22: Restoring deleted files using EOS recycling commands

eos recycle purge

```
EOS Console [root://localhost] l/eos/eoslia/users/l/labrador/> recycle purge
success: purged 0 bulk deletions and 1 individual files from the recycle bin!
```

Figure 23: Permanently deleting of deleted files using EOS recycling commands

4.9 Scalability Issues

ownCloud itself has problems related to scalability due to some design flaws. This section focuses on two major scalability problems found while developing the CERNBox.

4.9.1 Redundant Expensive Calls

ownCloud does not rely on system call errors to handle the logic of an operation. If someone ask you to perform the removal of a file named '/tmp/deleteme.txt', the common approach is to run 'rm /tmp/deleteme.txt' and expect a 0 error code meaning success. If the error code is not 0, then maybe the file is not found (error code = 2) or you do not have enough permissions to perform such operation (error code = 3). ownCloud way to removing that file is the following:

- Does the file exist? ownCloud performs this operation to see if the file is there.
- Do we have permission to delete it? ownCloud performs this operation to see if we can remove the file.
- Delete it

The problem with this approach is that the first two operations talk to the namespace, thus requiring two network round trips to check the status of the file and one to really remove the file. In order to mitigate this problem, CERBox uses a request cache (see Figure 49 for more details) that caches an operation if it occurs more that once per request.

4.9.2 File Target Creation

As explained in Section 4.5, the values of the file_target fields inside the SQL database is the name the receiver of the share will be in his user tree. ownCloud behaviour is to have unique file names across different storages. When sharing a resource with somebody, to create the file target (like a symbolic link) in the target user virtual file system, all the storages registered for the target user are scanned to check if there is already a file with the given file name. This check is really costly if the user has lot of files or if she has received lots of shares. Furthermore, there is even a worse scenario, sharing to a group as the scan must be triggered for all the members inside the group. This causes an explosion of expensive scanning operations that limit the scalability of the product.

CERNBox solves this problem creating a unique file name across different storages. For such purpose, the already existing file ID of a resource is appended to the target file name, for example: imagine a scenario where user A has a folder called 'photos' with a file ID of '123'. When user A shares this folder with user B, this file will appear in the shared storage of user B as 'photos (#123)'. This approach also helps to avoid having duplicated shared data, as the file ID is unique for all the resources.

4.9.3 Inefficient Handling of Requests

ownCloud's internal logic flow for handling requests always triggers the mounting of storages for the logged in user, independent of the operation being done. For example: the web application triggers periodically a hearth beat operation to check if the server is still alive. This operation does not need to use any storage, but storages are mounted anyway, thus consuming resources without need. There is not any clean solution at the time of this writing for this problem without having to modify the internal request flow. ownCloud will improve the request flow using the Dependency Injection pattern to just use what is required for every request.

5 CERN Services Integration

CERNBox does not just offer the same use cases as DropBox, it goes a step further to provide integration with already existing services and workflows inside the organisation.

5.1 SSO

Signing-on to a service such as e-mail involves entering a username and password: the credentials. What happens behind the scenes when credentials are entered is called authentication: the computing service in question establishes who is the owner of a valid account and gives access to it. Managing who gets access to which computing services, and with what privileges, is called authorisation. Typically, the service manager authorises the access and privileges of individuals and groups. For authorisation to work, information about individuals has to be associated with the groups they belong to and roles they play in the organisation – a process known as identity management. This ultimately involves information managed by the organisation's human resources service. This situation is summarised in Figure 24.

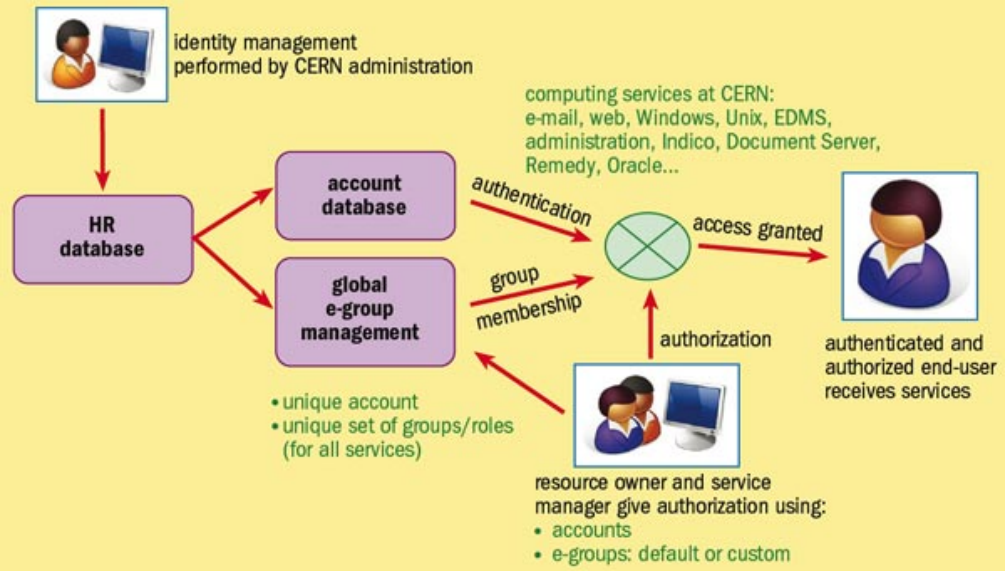


Figure 24: An overview of the identity management system at CERN, and the players involved.

Several years ago in the course of an average working day a typical CERN user would have to authenticate many times using many different credentials. First a login and password were required to unlock the Windows desktop, then the user had to type in another login and password to unlock the Linux desktop or session. Credentials were also needed to read e-mails, use administrative applications, and submit a CHEP presentation on Indico[23], for example.

The aim of the CERN authentication service is to provide a single sign-on for CERN applications, using a unique central account database.

The CERN Single Sign On (SSO) infrastructure provides different authentication methods, with different security levels:

- Classic web forms, where users type in the login and password. Depending on the browser features, the login and password can be saved locally for later use.
- Windows integrated authentication. The current Windows session token is reused to authenticate the user, without having to retype credentials. The security of Windows desktop sessions has also been increased by deploying a strong screen-lock policy: screen savers with password locking forced on all Windows desktops with a short time out, to avoid users leaving their screens unlocked for long periods of time.
- Certificates. Authentication can be made using a certificate provided by the CERN Certification Authority (CA) or any Grid-trusted CA member. Depending on the security level required, SmartCard tokens with pin codes can also be used.

The calling application can select all or some of the authentication methods. For example, to ensure maximum verification, experiment controls can request operators to authenticate only with certificates on SmartCards.

[23] The Indico tool allows to manage complex conferences, workshops and meetings.

The single-sign-on infrastructure can also provide information about the user needed for the calling applications. All account information associated with the user is returned to the calling application, such as name, e-mail address and building. Membership of groups and mailing lists is also returned, so that the calling application can rely on central group membership to handle access control. Instead of a ”per application dedicated role system” we now have a centrally managed group management on which all applications can rely.

CERNBox will use the SSO service to authenticate users. The technology used to do the SSO process will be Shibboleth as it is the only supported by Apache that can interact with CERN SSO.

Shibboleth is an open-source project that provides Single Sign-On capabilities and allows sites to make informed authorisation decisions for individual access of protected online resources in a privacy-preserving manner.

Shibboleth is a standards based, open source software package for web single sign-on across or within organisational boundaries. It allows sites to make informed authorisation decisions for individual access of protected online resources in a privacy-preserving manner.

The Shibboleth software implements widely used federated identity standards, principally the OASIS Security Assertion Markup Language (SAML), to provide a federated single sign-on and attribute exchange framework. A user authenticates with his or her organisational credentials, and the organisation (or identity provider) passes the minimal identity information necessary to the service provider to enable an authorisation decision. Shibboleth also provides extended privacy functionality allowing a user and their home site to control the attributes released to each application.

At its core Shibboleth works in the same way as every other web-based Single Sign-on (SSO) system. What distinguishes Shibboleth from other products in this field is its adherence to standards and its ability to provide SSO support to services outside of a user’s organisation while still protecting their privacy.

The main elements of a web-based SSO system are:

- Web Browser - represents the user within the SSO process.
- Resource - contains restricted access content that the user wants.
- Identity Provider (IdP) - authenticates the user.
- Service Provider (SP) - performs the SSO process for the resource.

In CERNBox the ‘Resource’ to grant access is the CERNBox Server, the Identity Provider is the CERN SSO service and the Service Provider is Shibboleth.

These are the Single Sign-on Steps:

- Step 1: User accesses the Resource

  The user starts by attempting to access the protected resource. The resource monitor determines if the user has an active session and, discovering that they do not, directs them to the service provider in order to start the SSO process.

- Step 2: Service Provider issues Authentication Request

  The user arrives at the Service Provider which prepares an authentication request and sends it to the Identity Provider. The Service Provider software is generally installed on the same server as the resource.

- Step 3: User Authenticated at Identity Provider

  When the user arrives at the Identity Provider, it checks to see if the user has an existing session. If they do, they proceed to the next step. If not, the Identity Provider authenticates them (e.g. by prompting for, and checking, a username and password) and the user proceeds to the next step.

- Step 4: Identity Provider issues Authentication Response

  After identifying the user, the Identity Provider prepares an authentication response and sends it and the user back to the Service Provider.

- Step 5: Service Provider checks Authentication Response

  When the user arrives with the response from the Identity Provider, the Service Provider will validate the response, create a session for the user, and make some information retrieved from the response (e.g. the user's identifier) available to the protected resource. After this, the user is redirected to the resource.

- Step 6: Resource returns Content

  As in Step 1, the user is now trying again to access the protected resource, but this time the user has a session and the resource knows who they are. With this information the resource will service the user's request and send back the requested data.

With this integration, CERNBox service will offer users the CERN standard authentication strategy.

5.2 E-Groups

The use of groups in a FSS platform is a really good feature, as it allows to share a resource with a group, not just with individual users, thus achieving a new level of collaboration. ownCloud offers the possibility to create user groups or retrieve them from an LDAP/AD service, but the way ownCloud handles groups is very inefficient due to their design of a shared namespace. Imagine a scenario where there is an group called cern-staff that has 10 000 members inside. When somebody wants to share some folder with this group, ownCloud queries the LDAP service for the list of members inside this e-group and then inserts a row in the SQL database per member of the group, thus having 10 000 new rows in the database.

To mitigate this problem CERNBox relies on e-groups. E-groups is the interface to manage groups at CERN. An e-group is managed by an owner and administrators. They have the same level of permissions. The owner or administrators add members, and then the e-group can be used as a mailing list or to grant access to different CERN resources (web sites, DFS folders, INDICO events, DFS folders etc). There are couple of ways to access e-groups, but the common way is to query them through LDAP/AD queries. To avoid the problem of inserting as much rows as members in the group, CERNBox just keeps the group name in the database (1 row) and group resolution is made on the fly when the user logs in the service.

5.3 JSROOT: Visualisation of physics data

ROOT is a framework for data processing, born at CERN, at the heart of the research on high-energy physics. Every day, thousands of physicists use ROOT applications to analyse their data or to perform simulations. With ROOT you can save user's data (and any C++ object) in a compressed binary form in a ROOT file. The object format is also saved in the same file: the

ROOT files are self-descriptive. Even in the case the source files describing the data model are not available, the information contained in a ROOT file is always readable. ROOT provides a data structure, the tree, that is extremely powerful for fast access of huge amounts of data – orders of magnitude faster than accessing a normal file. Data can also be generated following any statistical distribution and model, making it possible to simulate complex systems. Results can be displayed with histograms, scatter plots, fitting functions, etc.

The ROOT team has developed a Javascript tool called JSROOT to visualise the content of ROOT files in the browser. JavaScript ROOT aims to provide interactive ROOT-like graphics in web browsers. Object data can be read from binary ROOT files (offline) or received from ROOT applications running THttpServer (online). It is successor of JSRootIO project.

As EOS is the primary disk storage for physics data at CERN, CERNBox is the right place to embed the JSROOT tool and offer a new way of accessing physics data. ownCloud offers a flexible framework to develop custom applications. CERNBox takes advantage of this framework to integrate the JSROOT tool directly into the ownCloud user interface as shown in Figures 25 and 26.

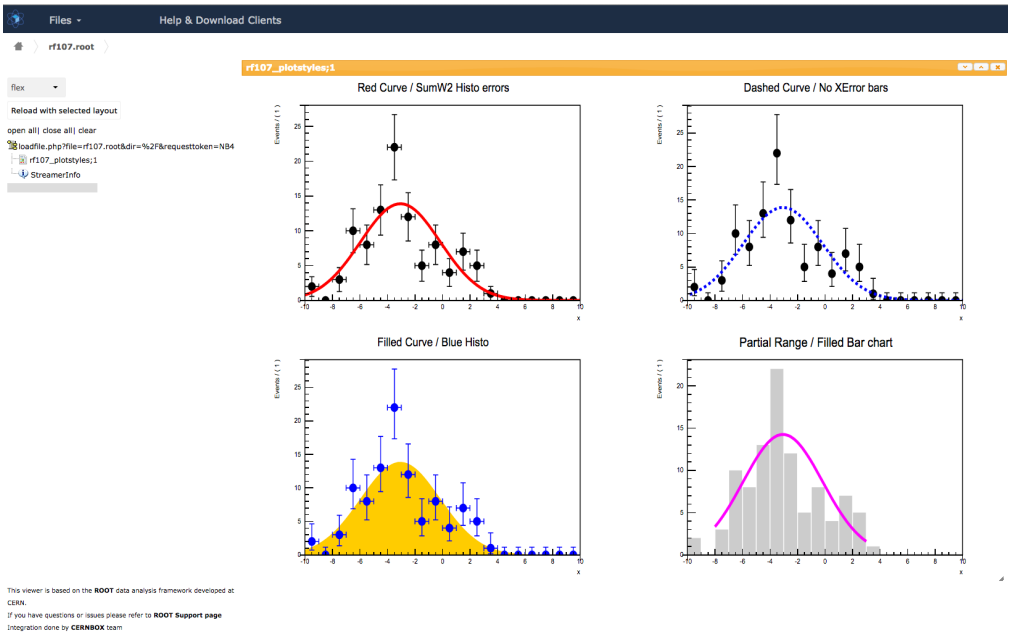


Figure 25: Using JSROOT inside CERNBox UI to plot data in 2D

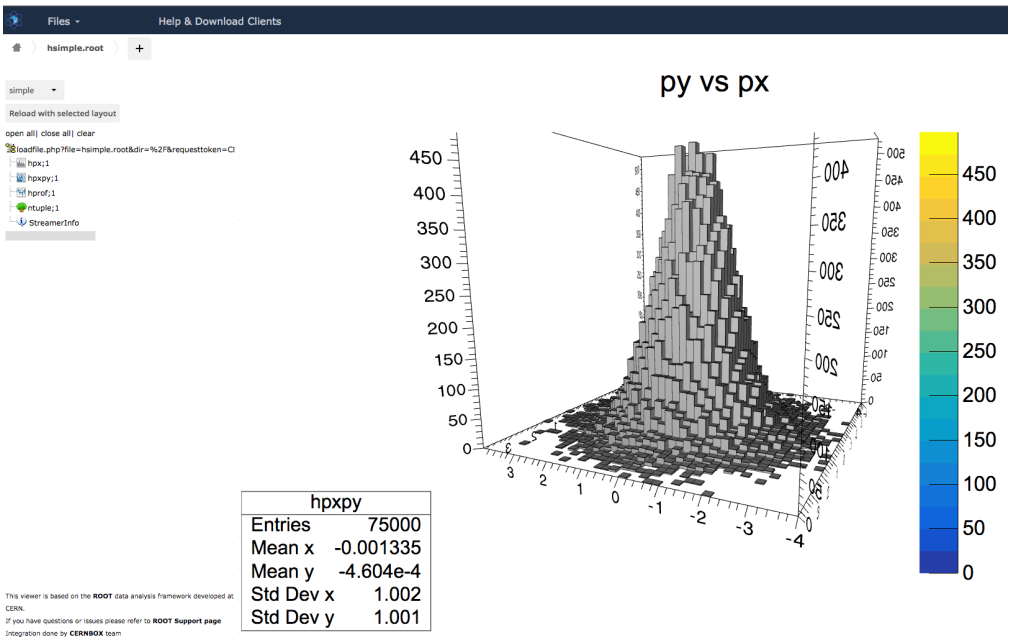


Figure 26: Using JSROOT inside CERNBox UI to plot a 3D model

5.4 NBViewer: Visualisation of interactive analysis

The Notebook (also called Jupyter Notebook or IPython Notebook) is a web application that allows to create documents that contain live code, equations, visualisations and explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, machine learning and much more. The Notebook has support for over 40 programming languages, including those popular in Data Science such as Python, R, Julia and Scala.

The ROOT team at CERN started a project called Root as a Service (ROOTaaS) to integrate NoteBook with ROOT technology. This new infrastructure will allow to use ROOT in the cloud, from a simple browser. This system will store data to EOS, therefore CERNBox becomes the main entry point for sharing and syncing of NoteBooks files. CERNBox does not just offer an application to visualise ROOT files as explained in the previous section, it also offers an application called NBViewer to visualise the NoteBooks[24]. Figures 27 and 28 show an overview of the displaying capabilities.

[24]The creation of the NoteBook is done by ROOTaaS, CERNBox just offers a visualisation layer.

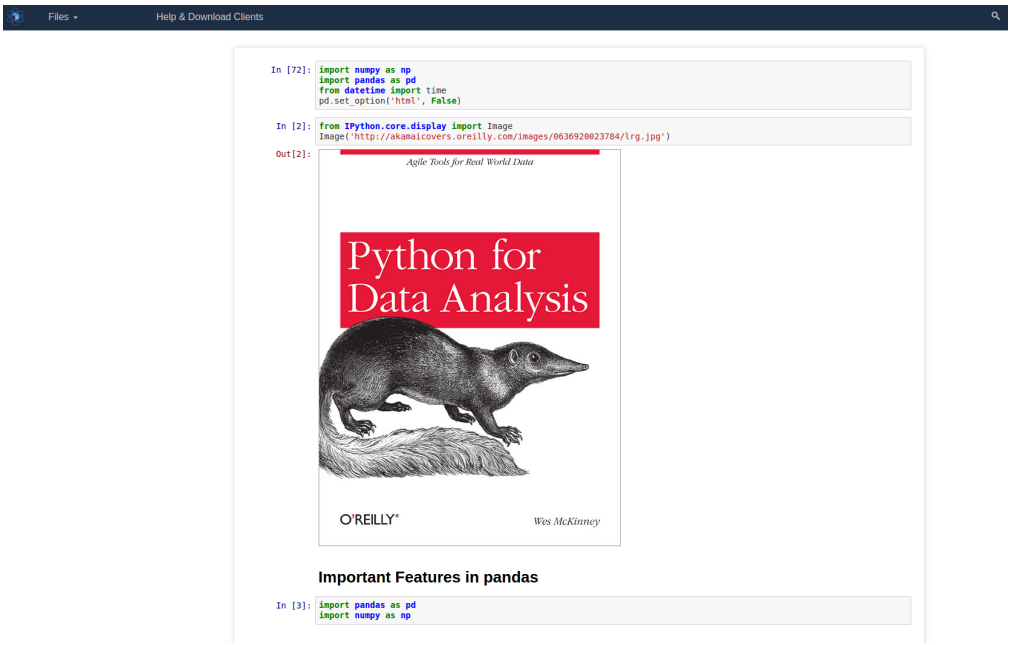


Figure 27: Using NBViewer inside CERNBox UI to display a NoteBook

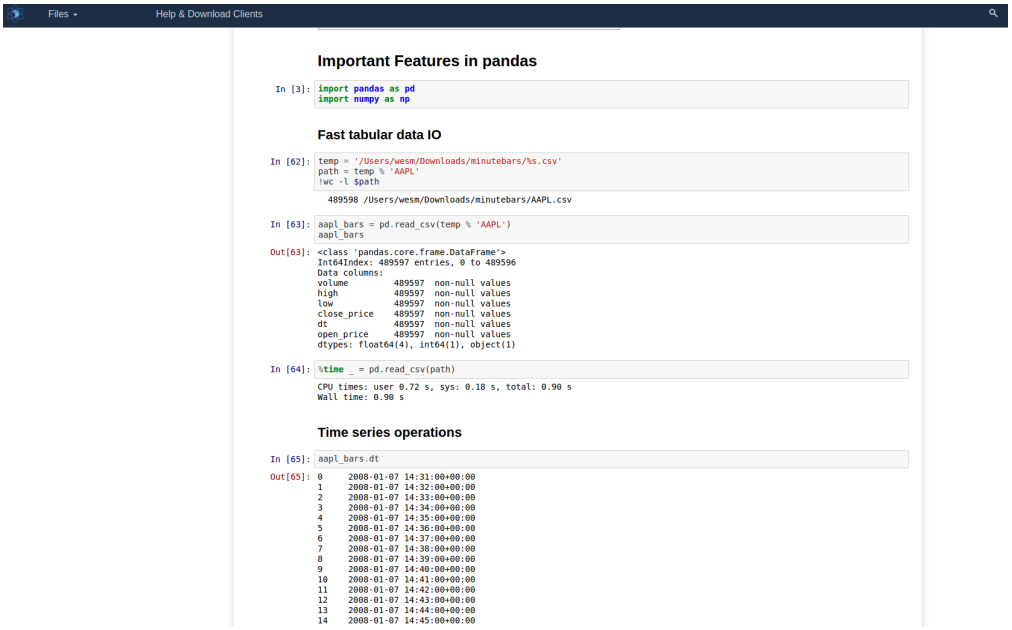


Figure 28: Using NBViewer inside CERNBox UI to display another NoteBook

6 Deployment and service metrics

6.1 Deployment

CERNBox uses a variety of open source products and infrastructure. The production service that has been running since 2015 relies on a set of servers located in the CERN Computer Centre at Geneva. These servers are bare metal servers (not virtualised) and live under a critical area, thus having redundant power supplies and UPSes[25] in case an outage occurs. EOS has 50 percent of the servers in the same computer centre and the other half in Budapest. The specifications for both CERNBox and EOS servers are outlined in Table 1.

Server	Component	Value
EOS Master MGM	RAM CPU	256 GiB E5-2630 v3@2.4Ghz
EOS Slave MGM	RAM CPU	256 GiB E5-2630 v3@2.4Ghz
FST Server	RAM CPU Disk	128 GiB E5-2630 v3@2.4Ghz 24 X 6TiB SaS/6Gib/s HBA
CERNBox Master Apache WebServer	RAM CPU	32 GiB E5-2630 v3@2.4Ghz
CERNBox Hot-spare Apache WebServer	RAM CPU	32 GiB E5-2630 v3@2.4Ghz
CERNBox Master NGINX	RAM CPU	32 GiB E5-2630 v3@2.4Ghz
CERNBox Hot-spare NGINX	RAM CPU	32 GiB E5-2630 v3@2.4Ghz

Table 1: CERNBox and EOS servers

6.2 Service Usage

EOS is the CERN disk storage for physics data and it has more than 140 PiB deployed, operates more than 1300 storage nodes and uses more than 40000 disks. The space used at EOS is divided in instances, often by experiment and use cases on. EOS has 6 different instances (see Figure 29) with different number of files (see Figure 35).

[25] An uninterruptible power supply, also uninterruptible power source, UPS or battery/flywheel backup, is an electrical apparatus that provides emergency power to a load when the input power source fails.

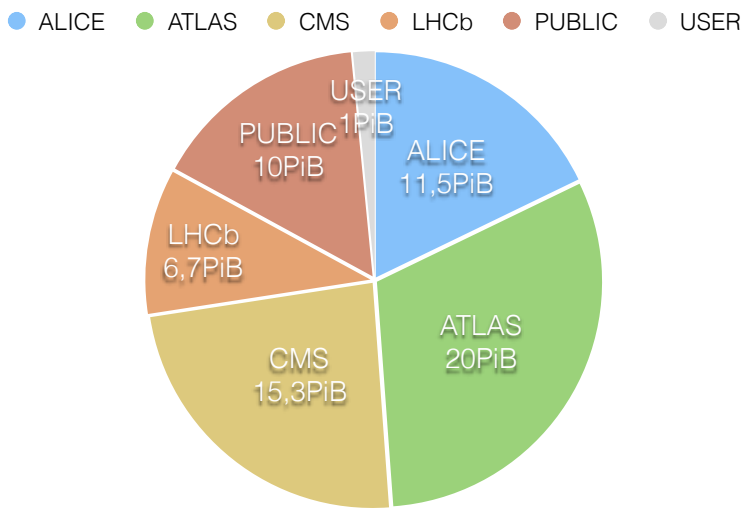


Figure 29: EOS instances with raw deployed capacity

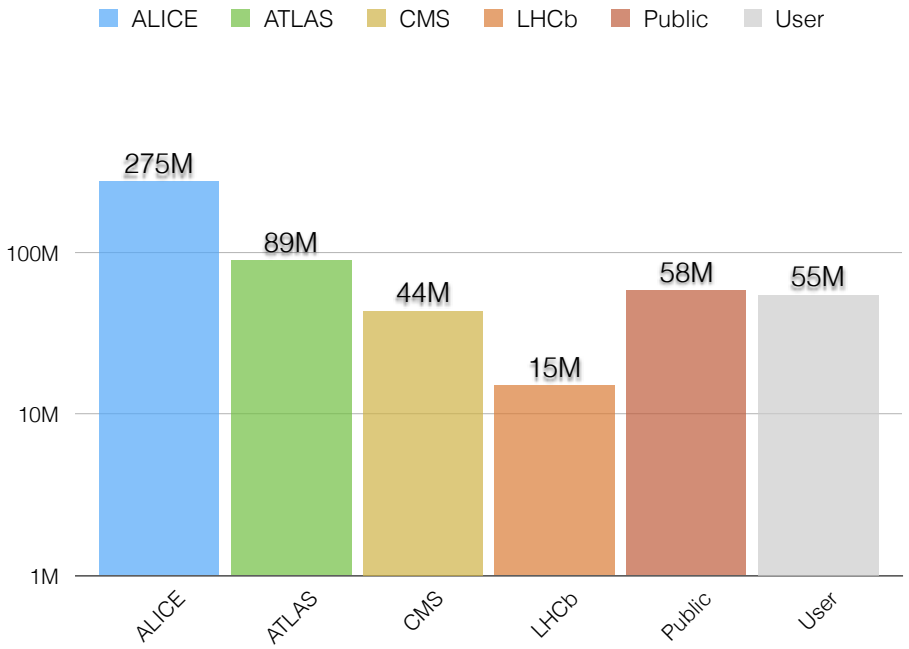


Figure 30: EOS file number distribution

The EOS instance where CERNBox currently operates is in EOSUSER (access to other instances is planned). EOSUSER is the EOS space reserved for general purpose storage, fulfilling the same uses cases as using DropBox and also offering new workflows using new tools as explained in Section 5.

Table 2 shows the CERNBox service numbers on the EOSUSER instance.

Users	4074
Number of files	55 Million
Number of directories	7.2 Million
Quota	1TiB/user
Used Space	104 TiB
Deployed Space	1.3 PiB

Table 2: CERNBox service numbers

CERNBox service can be accessed from a variety of clients from all major operative systems. Figure 31 shows the distribution of clients in the three major operating systems: OSX, Unix and Windows.

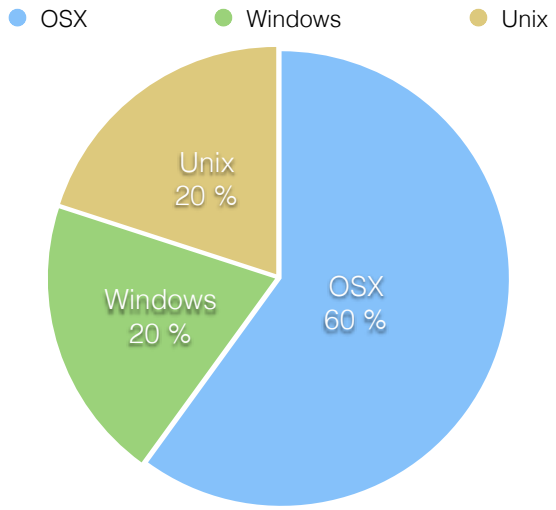


Figure 31: Distribution of operative systems from CERNBox desktop clients

Figures 32 and 33 show the worldwide geolocation of CERNBox users before Christmas holidays and during holidays respectively. Figure 34a and 34b show the geolocation for centre Europe.

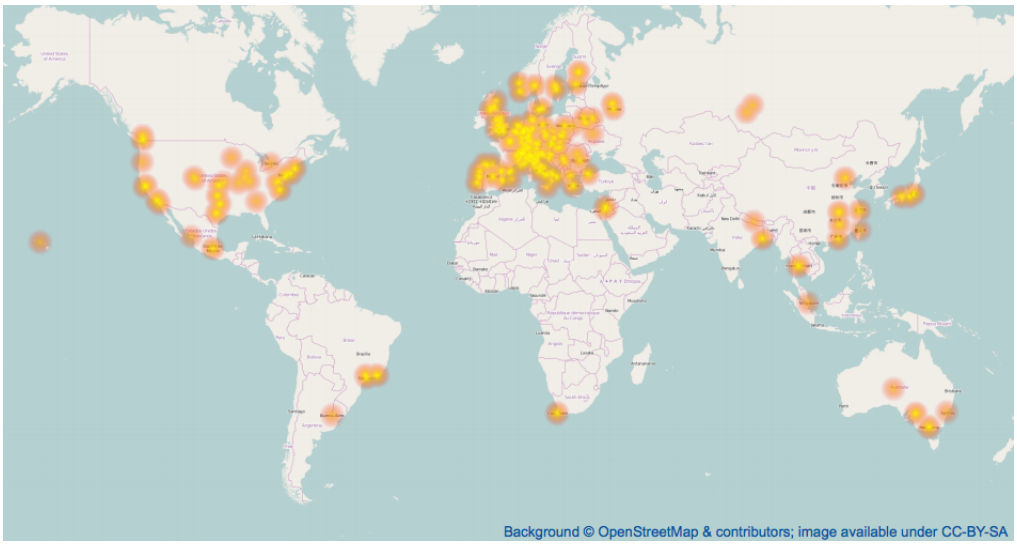


Figure 32: Geolocation of CERNBox users before Christmas holidays worldwide

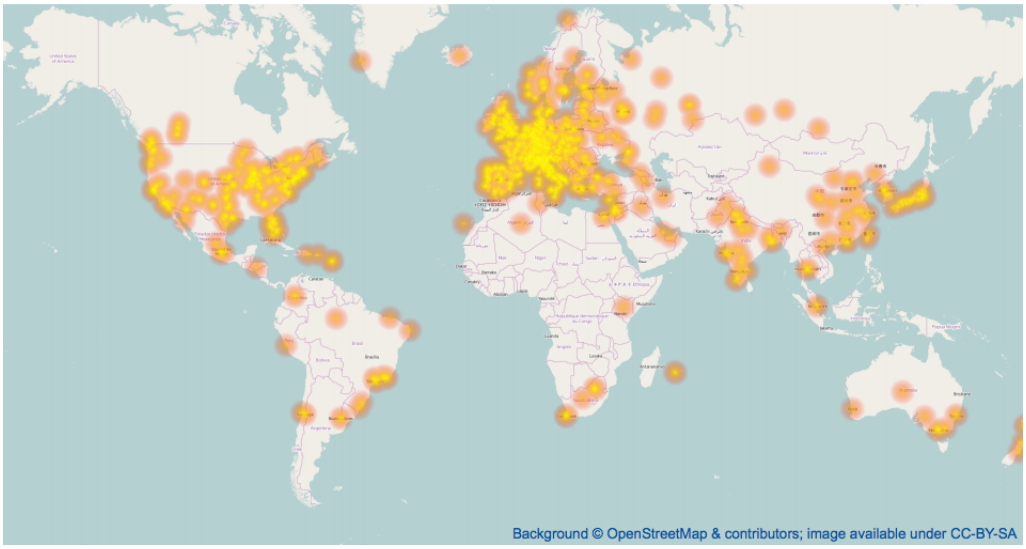
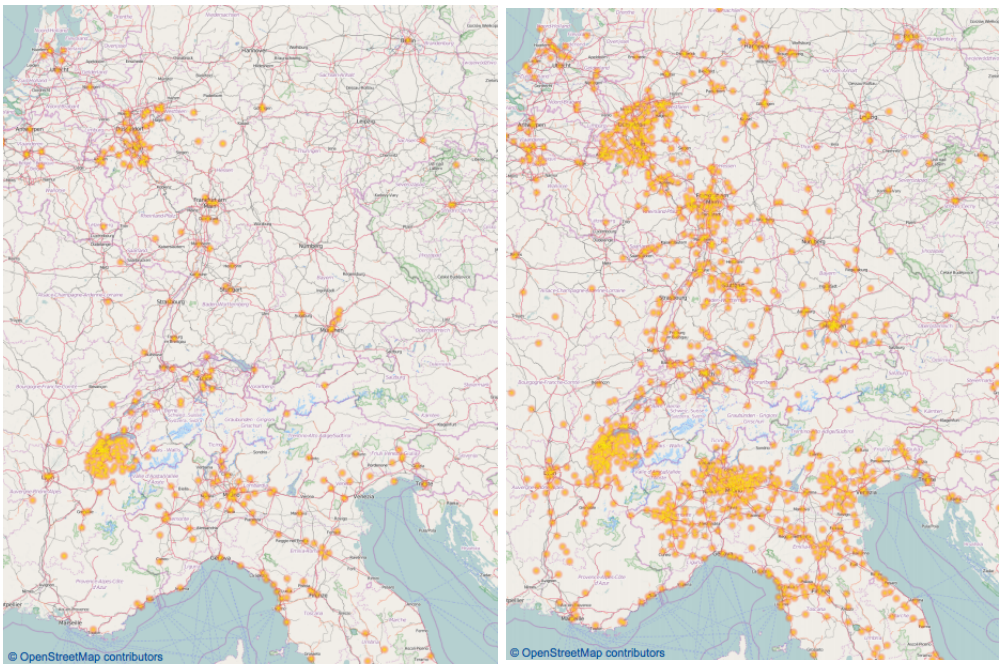


Figure 33: Geolocation of CERNBox users during Christmas holidays worldwide



(a) Before Christmas holidays (b) During Christmas holidays

Figure 34: Geolocation of CERNBox users in centre Europe

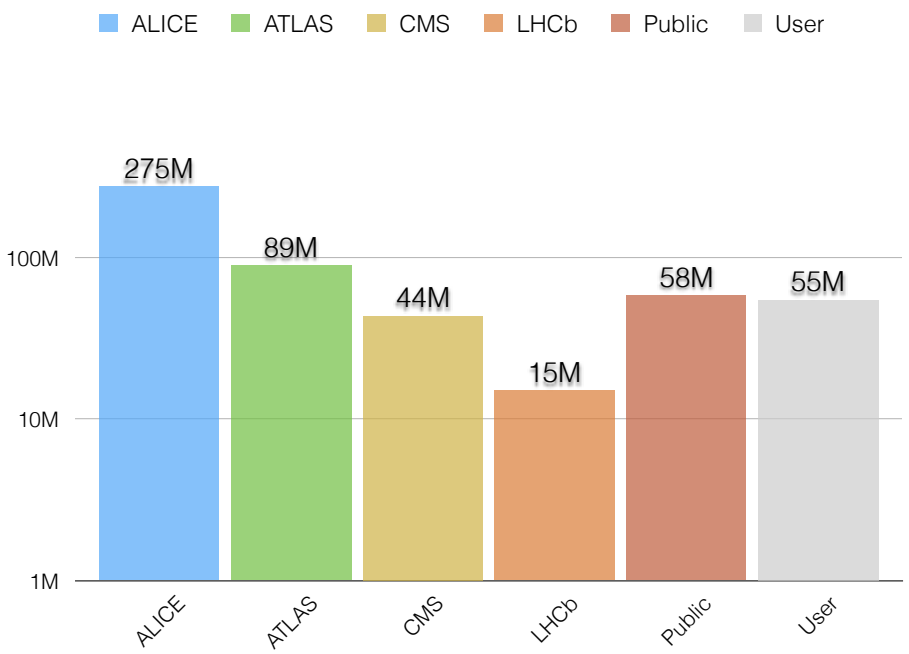


Figure 35: EOS file number distribution

Figures 36 and 37 show plots of different operations over the last 6 months of the production service

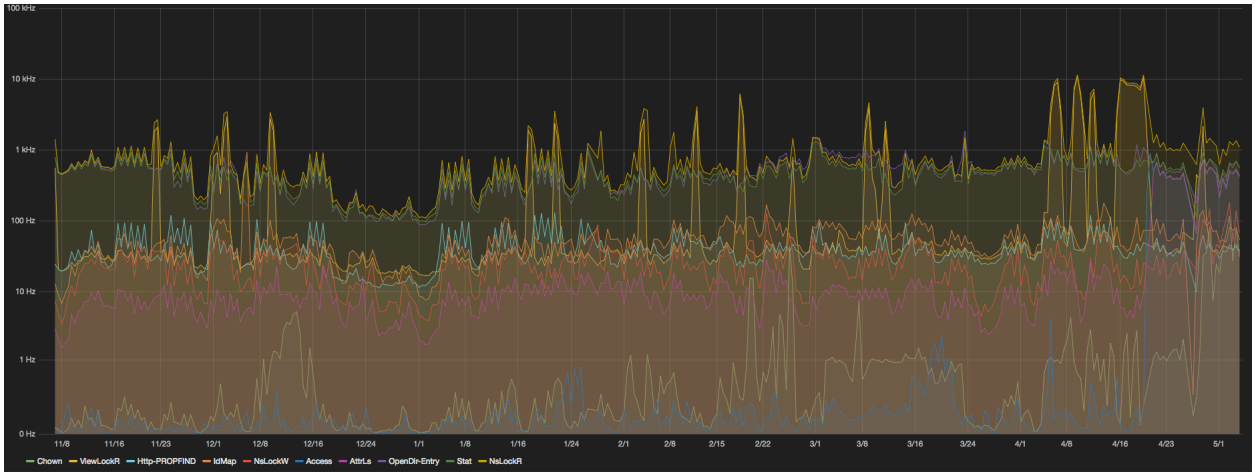


Figure 36: EOS plot showing the rate of different file operations

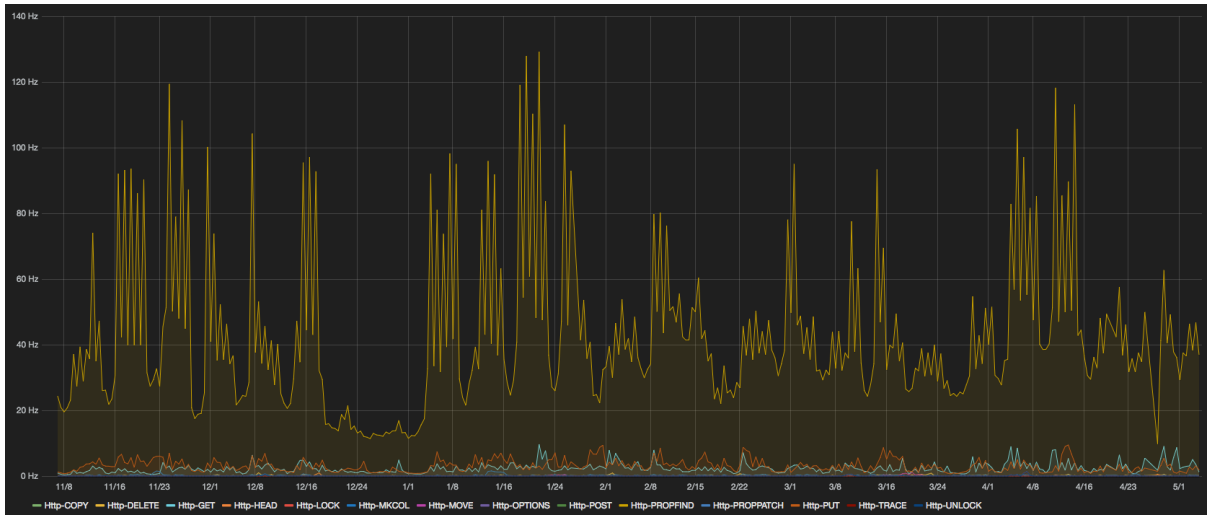


Figure 37: EOS plot showing the rate of different HTTP/WebDAV operations

6.3 Successful stories

6.3.1 E-Science

CIMA foundation[26] along with UNOSAT[27] and UNITAR[28] are involved in the project to develop a global Flood Early Warning System (flooding is the most common and widespread hazard worldwide) with the following objectives:

- Improve disaster response planning with timely identification of potential affected areas, in particular for critical areas of the world with lack of data.
- Support humanitarian actors during flood emergency with data and analysis.
- Guide satellite image acquisition to overcome delays due to the triggering process of satellite imagery.

With support of CERN IT department, the operational use of the modelling chain has been tested using a variery of CERN computing facilities: data have been uploaded to EOS through CERNBox to be handled from the batch system to model the forecast. The key features that CERNBox offers to this project are:

- Enabling non-experts to easily use CERNBox for data transfers.
- Powerful integration with the batch system
- Simplicity to share data with collaborators.

6.3.2 CERN Press Office

In March 2015, LHC Run 2 started. Run 2 of the LHC follows a 2-year technical stop that prepared the machine for running at almost double the energy of the LHC's first run. With this new energy level, the LHC will open new horizons for physics and for future discoveries. During the day of 28 of April, media around the world were waiting for the CERN Press Office to release information about the start of the run. The CERN Press team contacted the IT-DSS group to use CERNBox as the service to gather LHC 2 run information and share it with media around the world through protected link shares. The setup and work flow for this scenario were the following:

[26] Non-profit research organisation fostering training, scientific research and technological development within the fields of Civil Protection, Disaster Risk Reduction and Biodiversity.

[27] UNOSAT is a technology-intensive programme delivering imagery analysis and satellite solutions to relief and development organisations within and outside the UN system to help make a difference in critical areas such as humanitarian relief, human security, strategic territorial and development planning.

[28] The United Nations Institute for Training and Research (UNITAR) is a principal training arm of the United Nations, working in every region of the world. It empowers individuals, governments and organisations through knowledge and learning to effectively overcome contemporary global challenges.

- Photographers of the CERN Press team uploaded their photos to a shared CERNBox account to gather photos from different places.
- Editors of the CERN Press team prepared footages with the photos available in CERN-Box and uploaded these large videos (around 50GiB) using XROOTD clients directly to EOS, making them available from CERNBox.
- The CERN Press team shared a folder containing videos footages with members of different media partners across the globe.

6.4 Future uses cases

There are two ongoing research projects alongside the development of CERNBox: a worldwide Research and Development Network using EOS Word-Wide and the Future Home Directory for CERN users.

6.4.1 EOS Worldwide Deployment

The EOS and CERNBox projects are both open source, therefore they are just not limited to be used inside CERN. There are a couple of organisations that are studying to implant EOS in their service offerings and others that already started to use it. There are two organisations that have a working EOS instance: ASGC[29] and AARNET[30]. Figure 38 shows different EOS instances.

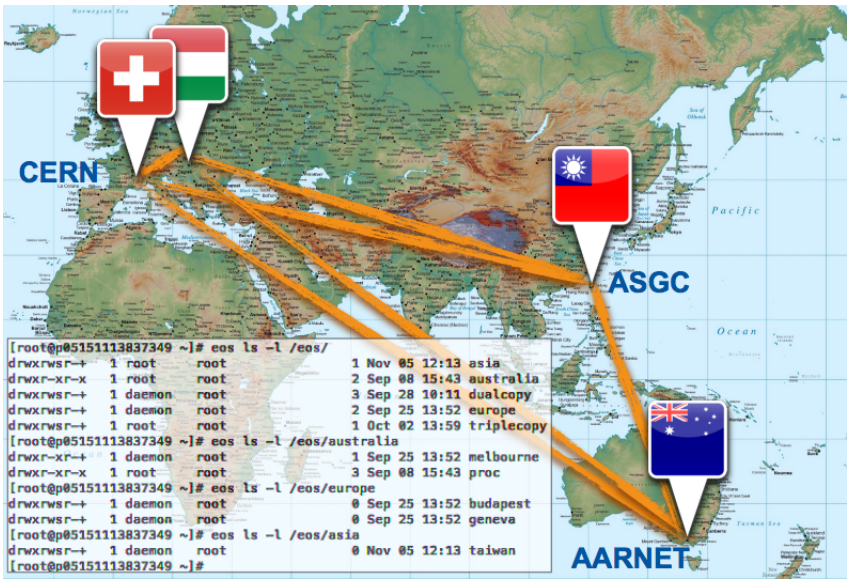


Figure 38: EOS worldwide deployments

[29] ASGC (Academia Sinica Grid Computing Centre) is the only Worldwide LHC Computing Grid Tier-1 Centre in Asia.

[30] AARNET is the Australian Academic and Research Network provides Internet services to the Australian education and research communities and their research partners.

6.4.2 Future Home Directory ($HOME)

A home directory is a file system directory on a multi-user operating system containing files for a given user of the system. The specifics of the home directory (such as its name and location) is defined by the operating system involved; for example, in Unix systems is often located at '/home/<username>' and can be referred with the '$HOME' environmen variable or just with '~'.

The AFS (Andrew File System) Service provides networked file storage for CERN users, in particular home directories, work spaces and project spaces. The AFS Service is based on OpenAFS, an open-source distributed filesystem which provides a client-server architecture for location-independent, scalable, and secure file sharing.

EOS is under study to see if it can replace the current AFS service to offer home directories, work and project spaces. Using EOS as the home directory will open new ways to access data: from the web through CERNBox Web UI, from desktop synchronisation clients, WebDAV clients and so on. Figure 39 shows an overview of data access paths using EOS as the user home directory storage.

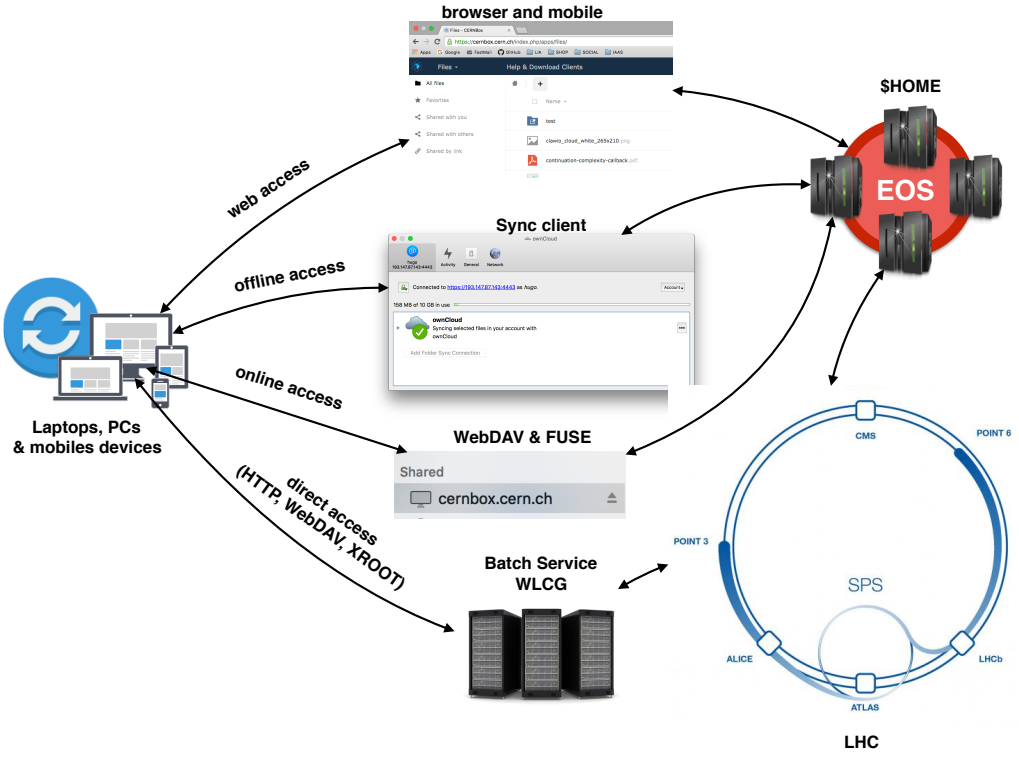


Figure 39: EOS home directory accesses

7 Conclusion

CERNBox is an innovative service with a fast user growing rate and positive feedback. The integration with the LHC petabyte storage and with existing workflows makes CERNBox unique. CERNBox also brings data closer to users with the same easiness as commercial services; furthermore it provides new ways to interact with data (see Sections 5.3 and 5.4). The running period of CERNBox and EOS has shown that it is an innovative platform for scientific computing with proven scalability and performance. The good experience in running CERNBox and the encouraging user feedback confirm our initial assumptions on the actual need of a cloud storage service for file synchronisation and sharing targeted to scientific communities. More importantly this service has the potential to become a central place for user data access since it addresses a number of issues, such as mobility and multi-platform access. It is an innovative service for sharing data between users and we expect this to become an environment that could fit physicists, engineers and general laboratory staff. We believe that CERNBox system could provide new communities an entry point for 'big data' activities as discussed in recent cloud storage events.

8 Future research

From our point of view the most important challenge is to use ownCloud out-of-the-box without having to apply patches and hacks to make it working with EOS. Fortunately, steps to solve this issue have started since beginning of year 2016 to make ownCloud 9 easier to integrate with EOS [31] thanks to a new design of ownCloud's internal architecture. However, release 9 did not meet at 100% the expectations we had, therefore more research needs to be made to obtain a more flexible design that allows seamless integration with EOS.

The development of CERNBox brought a series of ideas and designs to use the storage filesystem as the primary source from data retrieval without having to use a SQL database. These features currently apply to EOS, but they could be applied to a broader range of storage systems; this is the idea behind the ClawIO project[32].

[31] https://owncloud.org/blog/owncloud-server-9-0-released/
https://opensource.com/business/16/3/cern-and-owncloud
[32] http://clawio.github.io

A Software diagrams

oc_filecache		
PK	fileid	INTEGER(11)
	storage	INTEGER(11)
	path	VARCHAR(4000)
	path_hash	VARCHAR(32)
	parent	INTEGER(11)
	name	VARCHAR(250)
	mimetype	INTEGER(11)
	mimepart	INTEGER(11)
	size	BIGINT(20)
	mtime	INTEGER(11)
	storage_mtime	INTEGER(11)
	encrypted	INTEGER(11)
	unencrypted_size	BIGINT(20)
	etag	VARCHAR(40)
	permissions	INTEGER(11)

Figure 40: ownCloud file cache data model

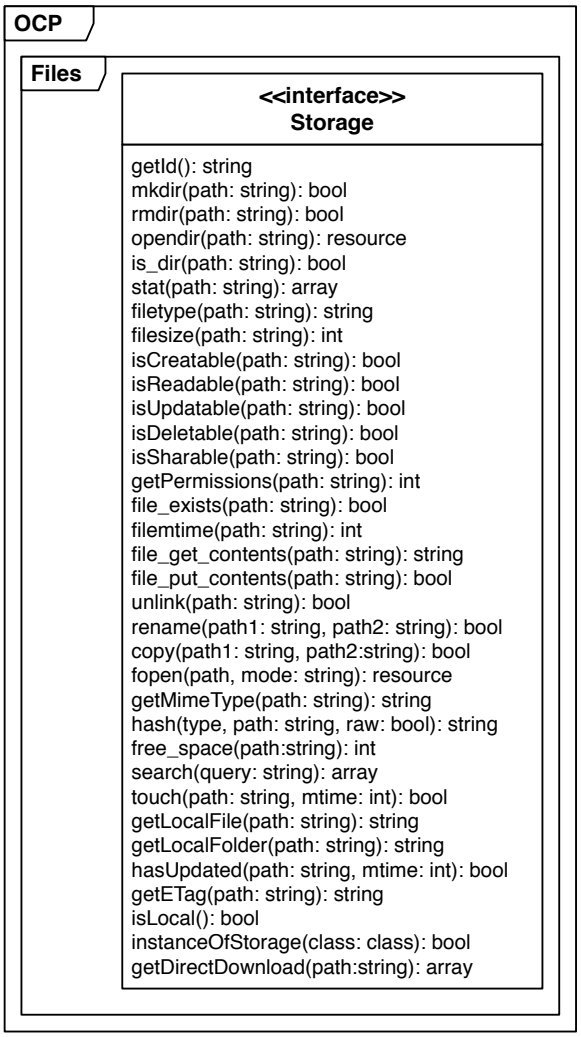


Figure 41: \OCP\Files\Storage

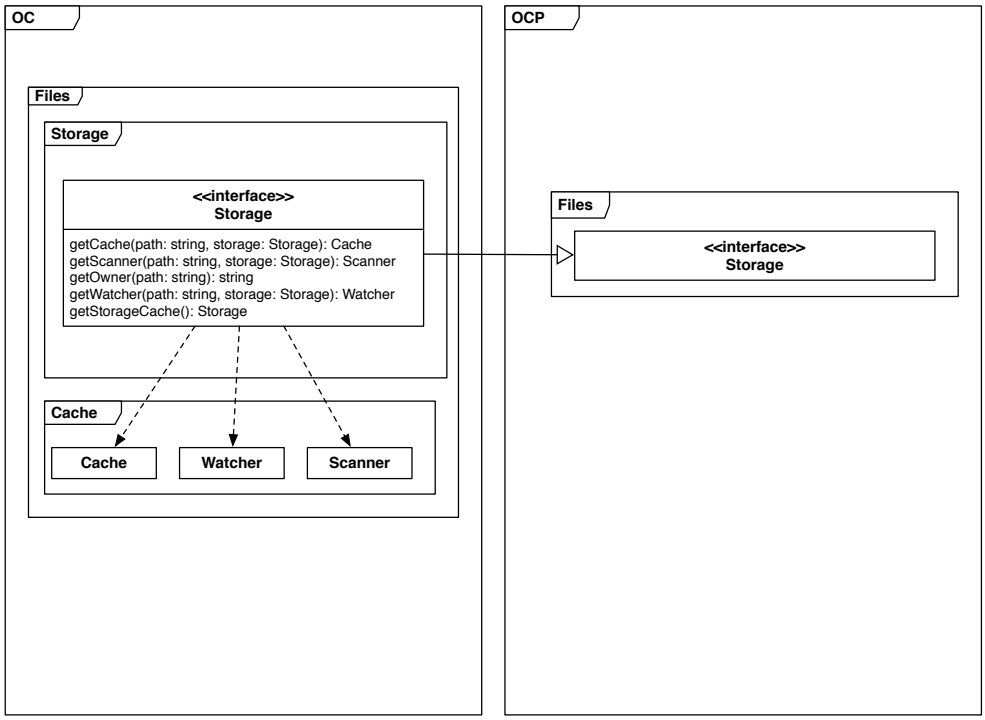


Figure 42: \OC\Files\Storage\Storage

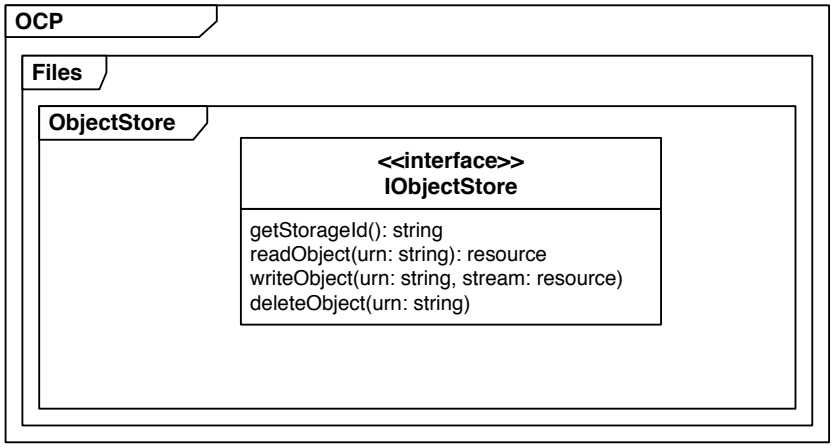


Figure 43: \OCP\Files\ObjectStore\IObjectStore

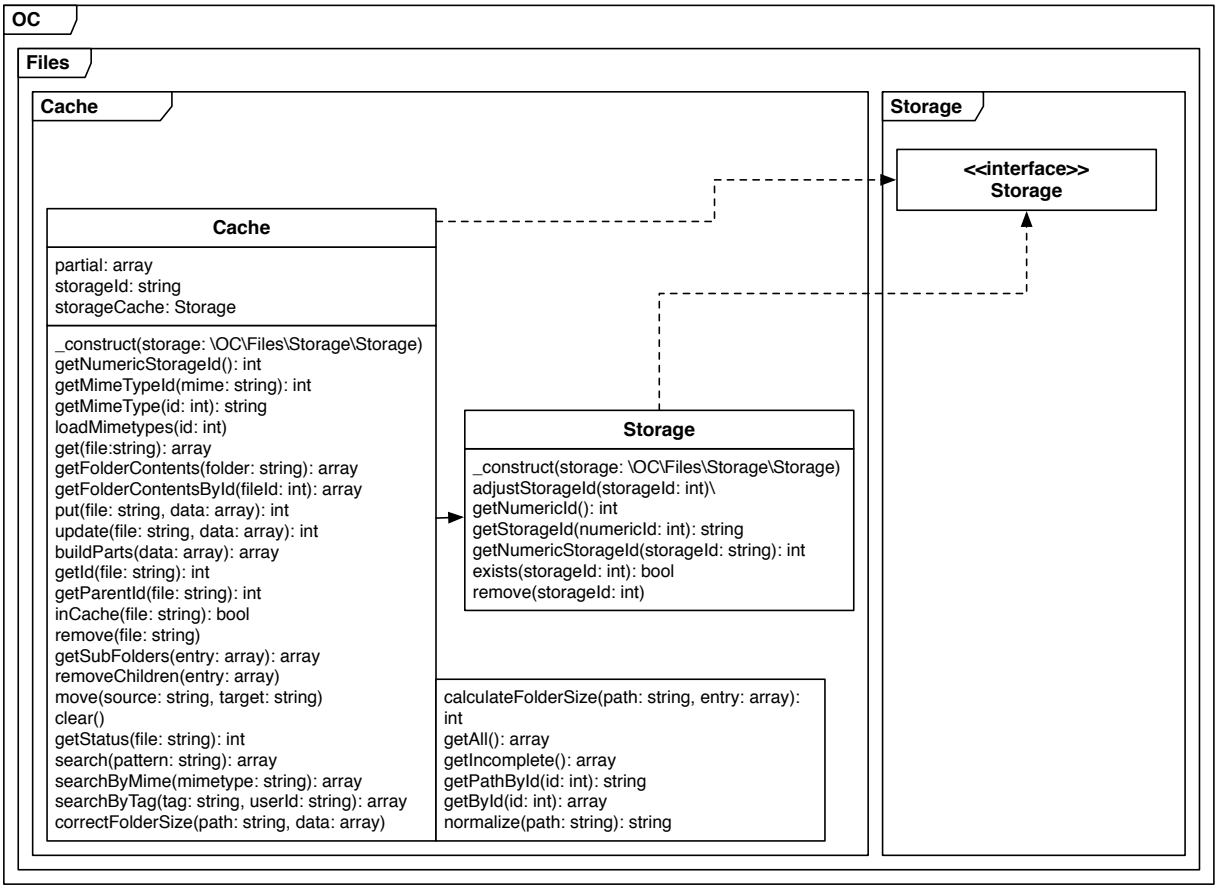


Figure 44: \OC\Files\Cache\Cache

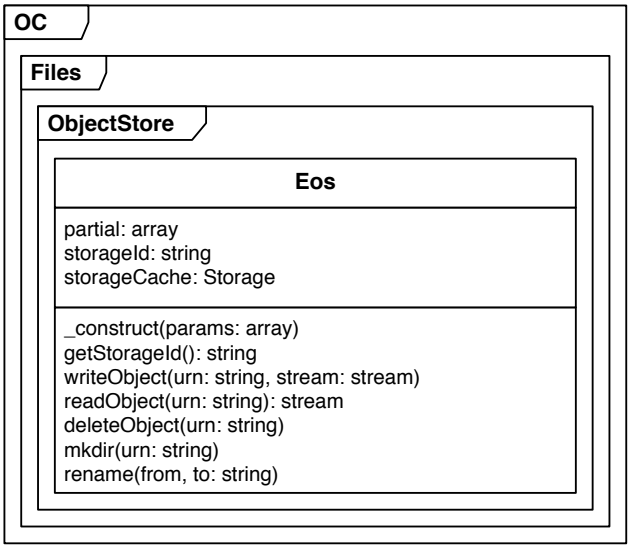


Figure 45: \OC\Files\ObjectStore\Eos

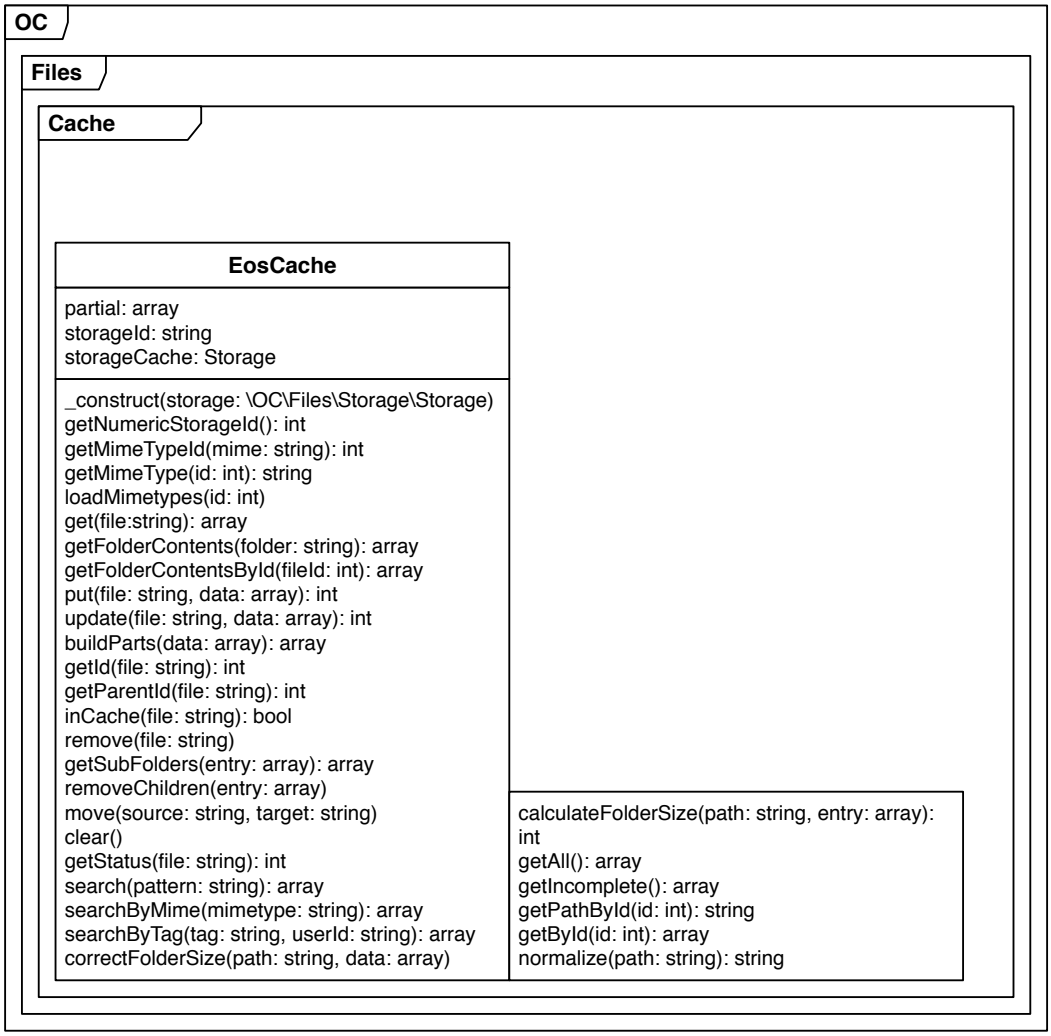


Figure 46: \OC\Files\Cache\EosCache

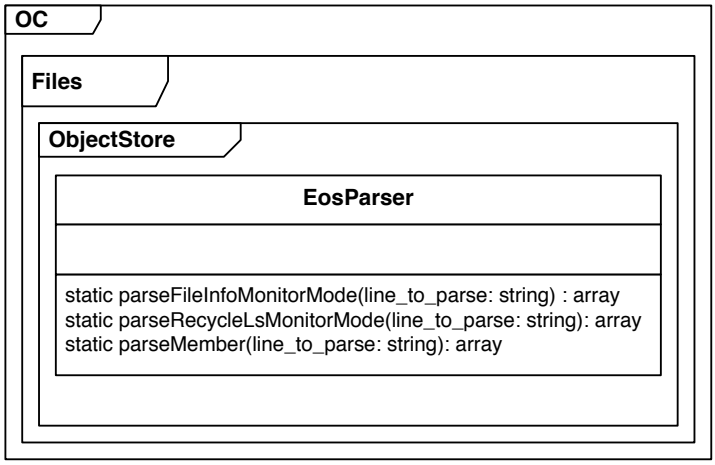


Figure 47: \OC\Files\ObjectStore\EosParser

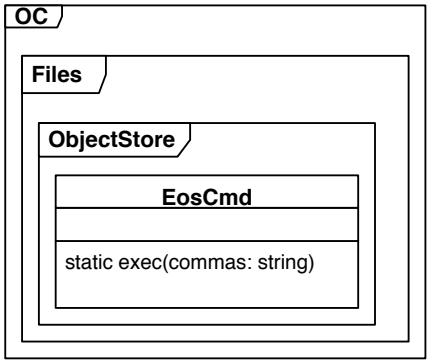


Figure 48: \OC\Files\ObjectStore\EosCmd

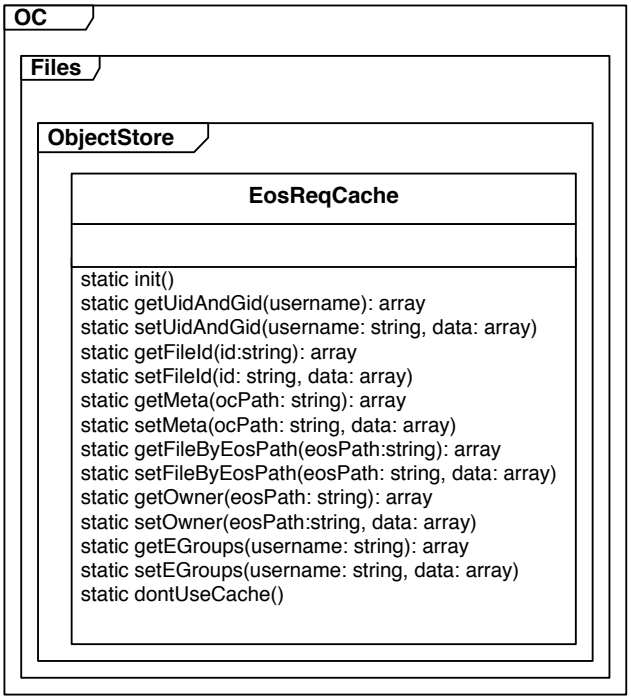


Figure 49: \OC\Files\ObjectStore\EosReqCache

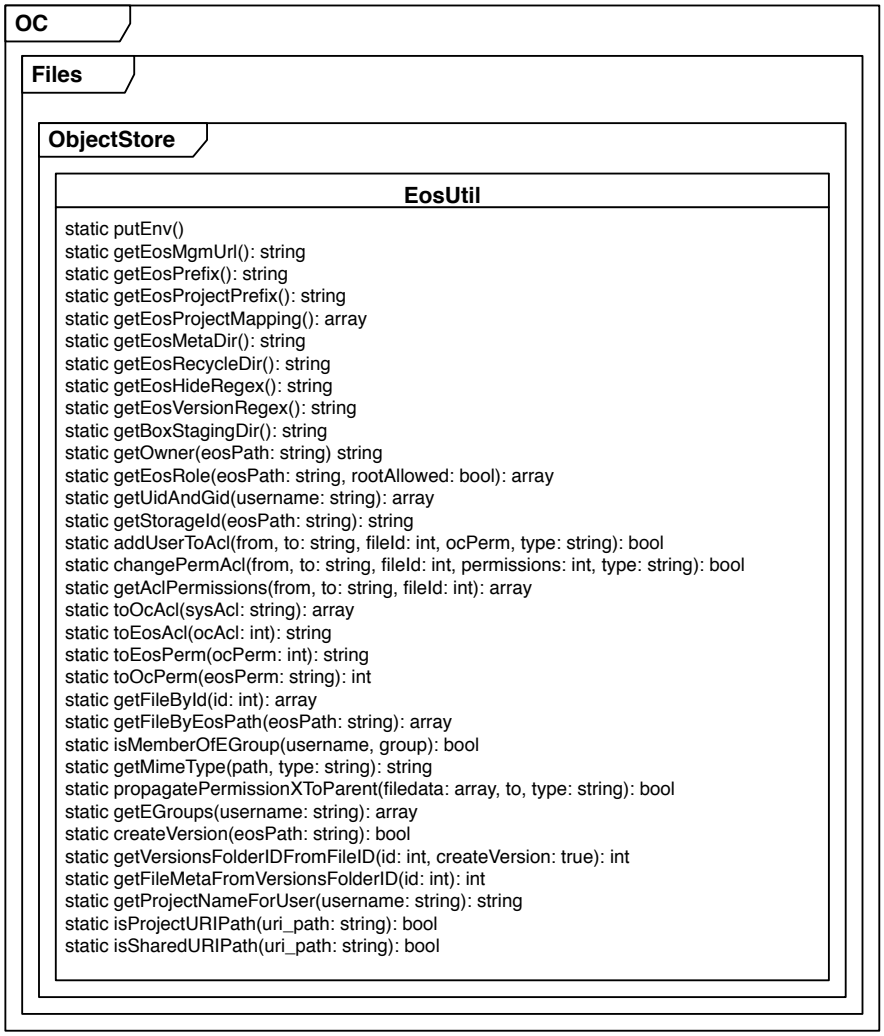


Figure 50: \OC\Files\ObjectStore\EosUtil

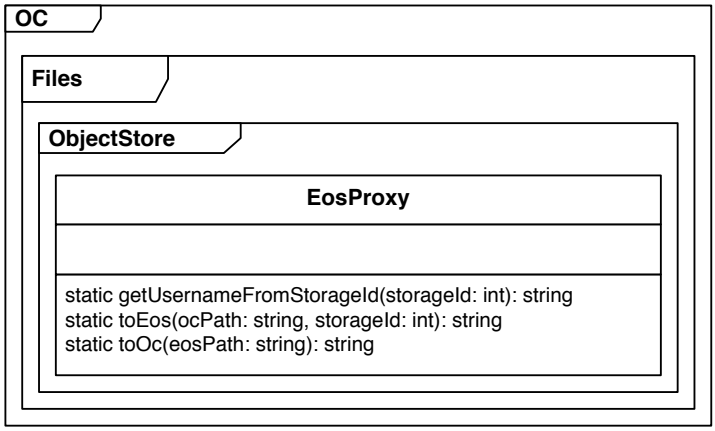


Figure 51: \OC\Files\ObjectStore\EosProxy

B EOS: The LHC Disk Storage

The following information has been mainly extracted from the publication 'EOS as the present and future solution for data storage at CERN' [17].

EOS is an open source distributed disk storage system in production since 2011 at CERN. Development focus has been on low-latency analysis use cases for LHC1 and non- LHC experiments and life-cycle management using JBOD2 hardware for multi PB storage installations. The EOS design implies a split of hot and cold storage and introduced a change of the traditional HSM3 functionality based workflows at CERN.

The 2015 deployment brings storage at CERN to a new scale and foresees to breach 100 PB of disk storage in a distributed environment using tens of thousands of (heterogeneous) hard drives. EOS has brought to CERN major improvements compared to past storage solutions by allowing quick changes in the quality of service of the storage pools. This allows the data centre to quickly meet the changing performance and reliability requirements of the LHC experiments with minimal data movements and dynamic reconfiguration. For example, the software stack has met the specific needs of the dual computing centre set-up required by CERN and allowed the fast design of new workflows accommodating the separation of long-term tape archive and disk storage required for the LHC Run II.

EOS is based on three components: management server, message queue and file storage services. All components are currently implemented as plug-ins for the xrootd storage server. Figure 52 show EOS architecture.

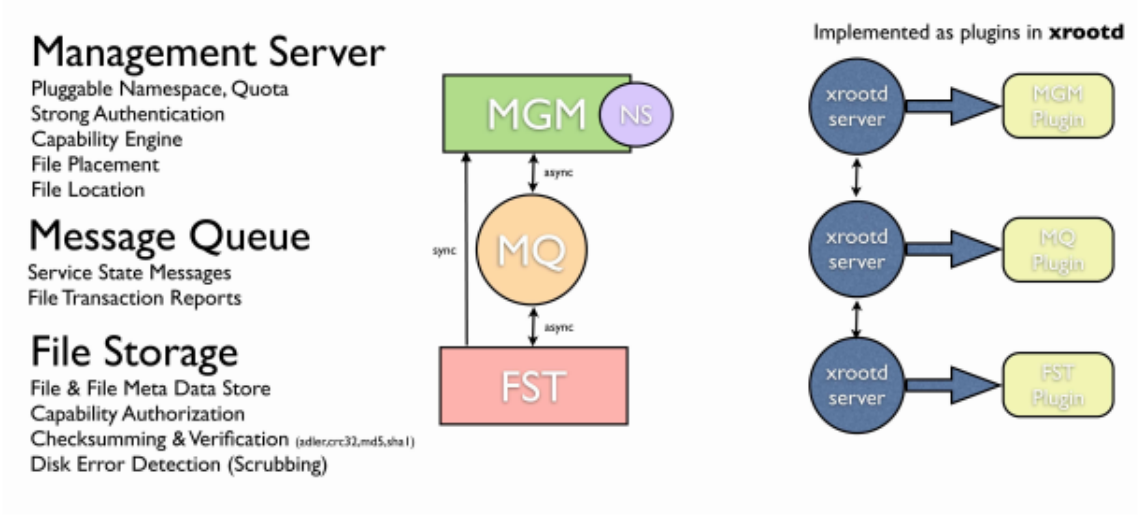


Figure 52: EOS architecture

A fundamental concept of EOS is to use a set of single disks (JBOD) as storage media without the need to build local RAID arrays. All storage nodes are divided into groups and within one group files are placed using file-level network RAID algorithms. The currently implemented algorithm is RAID-1(N) where (N) is the number of replicas for each file. Dual and multiple parity RAID algorithms are also considered. The storage cluster is self-healing: missing replicas are recreated on the fly. For RAID-1(N) at least one replica has to be accessible to guarantee

file availability. The used redundancy algorithm defines the quality of service in terms of file availability in case of single or multiple disk failures. File systems can be migrated online between nodes to simplify life-cycle management. The namespace is a common bottleneck in storage systems. In EOS it is implemented as a pluggable component which can be exchanged easily. The current implementation is built on in-memory hash maps. The required centrality makes a trade-off between scalability and latency allowing to scale out read open requests. Write open scaling is possible by splitting the namespace (see Figure 53). This limitation is sightly compensated by the fact that roughly only one quarter of all requests are indeed write requests according to 2010 LHC data.

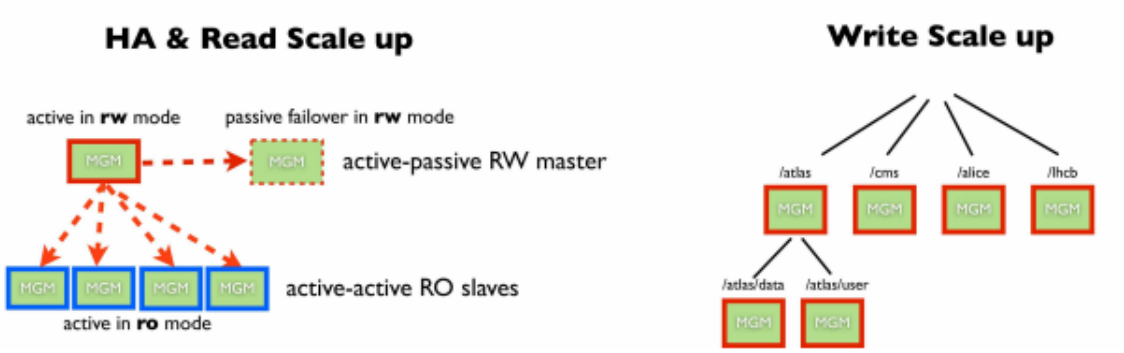


Figure 53: High Availability and Scale-Out overview

The hierarchical in-memory namespace is built on top of the google sparse hash implementation, which has a minimal size overhead for each hash entry. Every creation, modification and deletion of meta data of files and directories is stored in a sequential change-log file. On startup the change-log file is read and the in-memory hash maps are rebuilt. After repetitive meta data updates on files obsolete meta data records accumulate in the change-log file. EOS provides a tool to compact the change-log files by removing obsolete records. The namespace provides two views: a directory view and a file system view.

The directory view provides a familiar tree structure. The file system view allows to list all file ID-number on a particular file system. Keeping all meta data in memory is the fastest possible solution. The size is limited by the amount of available memory. The startup time is correlated to the in-memory size and the amount of change-log entries. A typical namespace uses 1 GB of memory for 1 million files. Size variations are dominated by the average length of file base names.

To obtain further details about EOS, there are some publications available [18, 19, 17, 14, 13, 6, 20].

Documentation to install and configure EOS can be found at the following url: http://eos.readthedocs.org/en/latest/.

References

[1] E. Bocchi. “Experiences of Cloud Storage Service Monitoring: Performance Assessment and Comparison”. Cloud Services for Synchronisation and Sharing (CS3). 2016 (cit. on p. 9).

[2] Enrico Bocchi, Idilio Drago, and Marco Mellia. “Personal Cloud Storage Benchmarks and Comparison”. In: (2015) (cit. on p. 9).

[3] Enrico Bocchi, Marco Mellia, and Sofiane Sarni. “Cloud storage service benchmarking: Methodologies and experimentations”. In: *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE. 2014, pp. 395–400 (cit. on p. 9).

[4] Yong Cui et al. “QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services”. In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM. 2015, pp. 592–603 (cit. on p. 9).

[5] Idilio Drago et al. “Benchmarking personal cloud storage”. In: *Proceedings of the 2013 conference on Internet measurement conference*. ACM. 2013, pp. 205–212 (cit. on p. 9).

[6] Xavier Espinal et al. “Disk storage at CERN: Handling LHC data and beyond”. In: *Journal of Physics: Conference Series*. Vol. 513. 4. IOP Publishing. 2014, p. 042017 (cit. on p. 60).

[7] Klass Freitag. “The File Sync Algorithm of the ownCloud Desktop Clients”. Workshop on Cloud Services for File Synchronisation and Sharing. 2014 (cit. on p. 9).

[8] Yaron Goland et al. *HTTP Extensions for Distributed Authoring–WEBDAV*. 1999 (cit. on p. 9).

[9] H González Labrador. “CERNBox: Site Report”. Workshop on Cloud Services for File Synchronisation and Sharing. 2014 (cit. on p. 5).

[10] H González Labrador. “From oc_filecache to an scalable and flexible OwnCloud namespace”. OwnCloud Contributor Conference. 2015 (cit. on p. 5).

[11] H. Gonzalez Labrador. “CERNBox: To Share or Not To Share”. CERN ITLT. 2015 (cit. on p. 5).

[12] L. Mascetti and H. Gonzalez Labrador. “CERNBox”. CERN IT Technical Forum. 2015 (cit. on p. 5).

[13] L. Mascetti et al. “CERNBox+ EOS: end-user storage for science”. In: *Journal of Physics: Conference Series*. Vol. 664. 6. IOP Publishing. 2015, p. 062037 (cit. on pp. 6, 60).

[14] Luca Mascetti et al. “Disk storage at CERN”. In: *Journal of Physics: Conference Series*. Vol. 664. 4. IOP Publishing. 2015, p. 042035 (cit. on pp. 6, 60).

[15] JT Mościcki, H González Labrador, and M Lammana. “Cloud Storage for (Big and Small) Science”. 15th TF-Storage Meeting. 2014 (cit. on p. 6).

[16] P. Mrowczynski. “Benchmarking and testing ownCloud, Seafile, Dropbox and CERNBox using smashbox”. Cloud Services for Synchronisation and Sharing (CS3). 2016 (cit. on p. 9).

[17] AJ Peters, EA Sindrilaru, and G Adde. “EOS as the present and future solution for data storage at CERN”. In: *Journal of Physics: Conference Series*. Vol. 664. 4. IOP Publishing. 2015, p. 042042 (cit. on pp. 59, 60).

[18] Andreas J Peters and Lukasz Janyst. “Exabyte scale storage at CERN”. In: *Journal of Physics: Conference Series*. Vol. 331. 5. IOP Publishing. 2011, p. 052015 (cit. on p. 60).

[19] Andreas-Joachim Peters, Elvin Alin Sindrilaru, and Philipp Zigann. “Evaluation of software based redundancy algorithms for the EOS storage system at CERN”. In: *Journal of Physics: Conference Series*. Vol. 396. 4. IOP Publishing. 2012, p. 042046 (cit. on p. 60).

[20] Elvin-Alin Sindrilaru, Andreas-Joachim Peters, and Dirk Duellmann. “Archiving tools for EOS”. In: *Journal of Physics: Conference Series*. Vol. 664. 4. IOP Publishing. 2015, p. 042049 (cit. on p. 60).

[21] E James Whitehead Jr and Meredith Wiggins. “WebDAV: IEFT standard for collaborative authoring on the Web”. In: *Internet Computing, IEEE* 2.5 (1998), pp. 34–40 (cit. on p. 9).