# ClawIO: Cloud Synchronisation Benchmarking Framework

Hugo González Labrador[a,*], Arno Formella[a], Jakub T. Mościcki[b]

[a] *ESEI, University of Vigo, Computer Science Department, Ourense, Spain*
[b] *CERN, IT-ST-FDO, Geneve, Switzerland*

**Abstract**

This paper presents the results of the research that led to the development of ClawIO: a cloud synchronisation benchmarking framework. The framework allows to build distributed and highly decoupled cloud File Synchronisation and Sharing platforms in order to obtain performance metrics after being stressed with different test suites. The framework introduces a flexible and extendable layered distributed architecture.

We develop a testing scenario for the framework that understands ownCloud's synchronisation protocol on top of a local filesystem. In this scenario, we present various performance analysis related to the usage of filesystem extended attributes and in-memory databases. We also introduce the benefits of using extended attributes for data consistency.

*Keywords:* cloud, benchmark, file synchronisation, ownCloud, filesystem

*Corresponding author
*Email addresses:* contact@hugo.labkode.com (Hugo González Labrador ), formella@uvigo.es (Arno Formella ), Jakub.Moscicki@cern.ch (Jakub T. Mościcki )

## 1. Motivation

File synchronisation between user devices and sharing files within a group of users, commonly known as File Sync and Share (FSS) services are among the most demanded services in recent years[1]. Such services allow to share and access data anywhere, from any device at any time, promoting data mobility and network pervasiveness. The number of FSS has grown over last years and the product catalog is now wider and more diverse than ever. For this reason, researchers around the world started to benchmark FSS services to obtain metrics that facilitate the election of a particular service against others in terms of synchronisation time, data and metadata consistency, and other features. Most of these studies compare services across different vendors, while we propose a study that measures the impact of the underlying technology that powers the synchronisation of such services.

Although existing studies [1, 2] provide meaningful insights of product capabilities, the results of measurements of about synchronisation performance and scalability are very vendor dependant, as the synchronisation client is not the same from vendor to vendor. To overcome this bias, we provide ClawIO, a cloud synchronisation benchmarking framework to test the performance and scalability of synchronisation protocols against a matrix of databases and storage backends; from the same benchmarking client.

## 2. Contributions

This study provides two meaningful contributions to the research communities working in the field of FSS services.

The first contribution is the development of ClawIO[2], a flexible, extendable and technology agnostic cloud synchronisation benchmarking framework. ClawIO is an undergoing development project to create a framework to test dif-

[1] Rise in cloud-based file sync-and-share services grabs IT's attention, Rich Castagna

[2] `http://clawio.github.io`

ferent synchronisation protocols against a matrix of databases and storage backends to observe the impact of such technologies in the synchronisation process. In this current phase of the project, the protocol tested is the synchronisation protocol used by ownCloud (OSP)[3], as ownCloud is one of the biggest players in the open source market and has been included in Gartner's first Magic Quadrant for the extremely competitive Enterprise File Sync and Share solution[4].

As a second contribution, this study also introduces how local filesystem extended attributes and in-memory databases can be used to handle the metadata needed for synchronisation and how this new way of storing such data leads to scalable, performant and reliable designs.

The remainder of this paper is organised as follows. Section 3 reviews related work in the area of cloud benchmarking. In Section 4, we present our architecture and the technologies involved. In Section 5, we introduce the test bed and the operations to be benchmarked (from a perspective of user actions). In Section 6, we explain the results obtained. In Section 7, we give our conclusions and outlook for further development and research.

## 3. Related Work

Over the last years, performance and feature comparative analysis have been done on FSS platforms. Feature comparisons can be found with a quick search in any search engine and vast amounts of results will appear [5]. On the other hand, data for the performance of different FSS platforms are hard to obtain. Examples of the latter include: QuickSync[1] (shows the performance characteristic of the DropBox synchronisation protocol), benchmarks led by Mr. Bocchi and Mr. Drago [2, 3, 4] (present various performance metrics on known FSS

[3] OC Sync Protocol, `https://git.io/vwZsI`

[4] ownCloud is Youngest, Smallest and only Open Source Company to Reach Gartners First Enterprise File Sync and Share Magic Quadrant, ownCloud, `https://goo.gl/OWaKDe`

[5] `https://en.wikipedia.org/wiki/Comparison_of_file_synchronization_software`, `https://www.getfilecloud.com/enterprise-file-sync-and-share-product-comparisons/`, `http://www.pcmag.com/article2/0,2817,2413556,00.asp`

platforms), a couple of studies presented at the Cloud Services for Synchronisation and Sharing conference[6] [5, 6] and also previous analysis done by PhD. Wang [7].

On the other hand, studies that measure the performance of a storage backend against a particular synchronisation protocol have not been found, therefore, we aim to provide new results to the research community.

## 4. Architecture

FSS software provides synchronisation layer on top of a variety of backend storages such as local filesystems and object storages. In case of some platforms, such as ownCloud, a SQL database is used to support the synchronisation requirements. We have tested how different backend technology configurations impact the ownCloud HTTP-based synchronisation protocol. Performance and scalability analysis were performed based on benchmarking the system with different end–user operations. The results have been produced using the ClawIO framework prototype. This software provides a base architecture to stress different storage solutions against different cloud synchronisation protocols. This architecture is influenced by two projects: the IETF Storage Sync Internet Draft [8] and CERN EOS project [9, 10, 11]. Both sources agree that data and metadata operations must be handled by different components to increase scalability through scale-out design. We achieve the separation of concerns by using highly decoupled microservices [12] connected to each other with high performance communication protocols such gRPC[7] and efficient data serialisation formats like Google Protobuf [13].

To broader the usage of ClawIO, it has been developed with four aspects in mind:

- Composability: ClawIO follows the micro service design and every compo-

[6] `http://cs3.ethz.ch`
[7] http://www.grpc.io

nent is encapsulated into a single logical unit with clear interfaces. Therefore, every component can be easily deployed and connected to others.

- Universality: ClawIO is built on top of two universal technologies: HTTP and gRPC. HTTP is used for binary data transfers and gRPC for inter component metadata communication using Google Protocol Buffers data format.

- Flexibility: every component inside ClawIO can be built with the technologies and programming languages, as long as the programming language is able to create HTTP and gRPC connections.

- Scalability: ClawIO has been designed with scalability in mind. The design is based on CERNBox [14, 15, 11] and EOS, components that power the synchronisation and sharing of huge amounts of scientific data from the Large Hadron Collider [11]. Furthermore, every logical component can be scaled–out with minimum effort.

FSS we analysed (Pydio[8] and ownCloud[9], and SeaFile[10]) have the following logical components:

- Data component: responsible for data upload and download.

- Metadata component: responsible for metadata operations like listing the contents of a directory or obtaining metadata like file size, modification time (the last time an entry has been modified) or the type of the file (commonly known as mime type, e.g. "text/plain" or "image/png").

- Authentication component: responsible for authenticating the user to use the FSS platform.

[8] `https://pydio.com/`
[9] `https://owncloud.org/`
[10] `https://www.seafile.com/en/home/`

- Synchronisation component: responsible for implementing the synchronisation protocol used by the FSS platform. The implementation technology varies depending on the vendor; in case of ownCloud, this component exposes WebDAV [16, 17] endpoints.

- Share component: responsible for sharing data. ClawIO focuses on the synchronisation side at this initial phase, thus the share component has not being included in the demonstration architecture.

ownCloud bundles these components into one physical software product. Others, like Seafile[11], offer a more distributed approach, but at the time of this writing, we have not found clear evidence of any FSS platform that splits every logical unit into a separate service, thus making ClawIO a pioneer in this distributed architecture.

[11] `http://manual.seafile.com/overview/components.html`

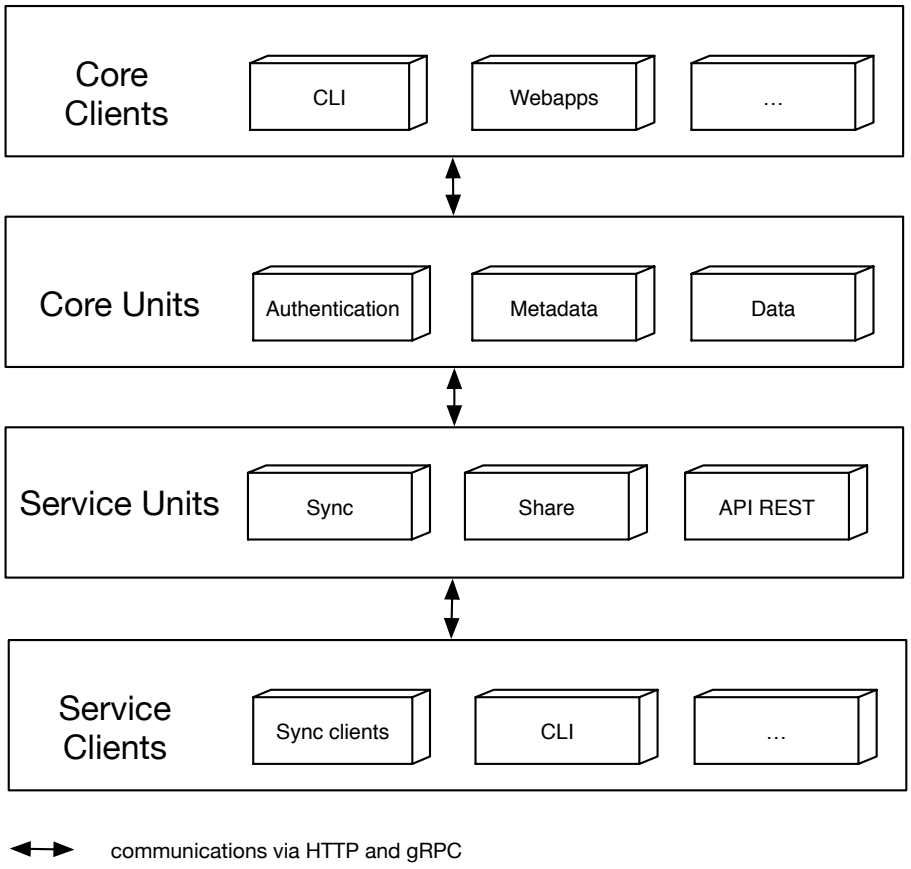



Figure 1: ClawIO layered and distributed design.

Figure 1 shows ClawIO logical architecture. This architecture consists of four layers:

- Core Clients Layer: this layer groups the clients that communicate directly with the services offered by Core Units Layer using gRPC and HTTP.
- Core Units Layer: this layer contains the units that are the core of ClawIO. They implement the minimum services required. These units are the authentication, metadata, and data services that all FSS platforms have in common. This layer contains the logic necessary to upload and download files, create and list directories and copy, move and delete resources.

- Service Units Layer: this layer embraces all the services built on top of the Core Units to provide extra functionality like synchronisation, sharing, REST APIs or other developer-defined services. This layer communicates with the upper layer using gRPC and HTTP. The services in this layer can expose arbitrary protocols for communication. For example, the Sync Unit can expose a WebDAV endpoint in the case of ownCloud or a REST API (architectural style of the World Wide Web [18, 19]) in case of DropBox[12].

- Service Clients Layer: this layer embraces all the clients that communicate with the services in the Service Units Layer.

Following this logical layout, Figures A.10a, A.10b and A.10c (inside AppendixA) illustrate the three different implemented architectures that were tested: alpha, beta and gamma. The three architectures share the Core Clients, Service Units and Service Clients Layers. The Core Clients Layer contains a command line tool (*clawio*) to interact with the data and metadata services and also the *clawiobench* tool to launch benchmarks against the Core Units Layer. In the Service Units Layer, the only service currently implemented is the Sync service, which implements the synchronisation protocol of ownCloud and exposes a WebDAV interface for clients in the bottom layer. The Service Clients Layer groups the ownCloud desktop and command–line synchronisation clients, cURL (a command line tool to perform HTTP requests), davfs2 (a Linux tool for connecting to WebDAV shares as though they were local disks) and any other WebDAV-talking clients. The Core Units Layer is the one that changes from one architecture to another, but still maintains the Authentication Unit, which authenticates users against an SQLite3 database. The changes are in the implementation of the data and metadata services.

Before explaining the three possible storage configurations, the synchronisation protocol used by ownCloud (OSP), implemented by the Sync Unit, is briefly explained to understand what metadata need to be kept in the metadata

[12] `https://www.dropbox.com/developers-v1/core/docs`

backend.

We have chosen this protocol because ownCloud is one of the most used open source FSS platforms and because we had previous experience with it, therefore speeding up the process to obtain preliminary results. ClawIO validates its implementation of the OSP using the SmashBox tool[13], part of the CERNBox project to ensure the reliability and quality of the service.

The OSP is state-based and it has a cycle of three clearly defined phases [20] 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 Sync Client has to decide what to do with each file based on the collected metadata. The result of this phase if 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 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 sync clients to perform a local rename instead expensive delete and download operations.
- 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

[13] `https://github.com/cernbox/smashbox`

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 sync client looses its sync journal and has to decide which file to kept in case of conflict.

Apart from these metadata, the following is also used in ClawIO:

- Size: the size of the resource in bytes.
- Resource type: specifies if the resource is a file or a directory.
- Mime type: identifies the format of the file.
- Checksum: fingerprint of resource data.

After having identified the metadata used in the three architecture (Figures A.10a, A.10b and A.10c), we focus now on the differences between them.

The Alpha architecture (Figure A.10a) uses MySQL to save the file ID, ETag, Mtime and checksum. The MySQL database is also used for keeping track of the propagation of ETags and Mtimes. The size, mime type, resource type and children are retrieved directly from the filesystem with operating system commands.

The Beta architecture (Figure A.10b) changes the way it retrieves the file ID and checksum attributes. Instead of obtaining them from the MySQL database, they are obtained from EXT4 extended attributes, i.e. they are attached to the file. This change brings two important consequences:

- The file ID update is atomic. Traditional FSS platforms obtain the file hierarchy from an intermediate component between the filesystem and

the user, which is often a SQL database. In these platforms the move operation has to be performed both on the filesystem (physical move) and in the database (logical move — to update the path the file ID is pointing to after the physical move), hence losing the atomicity of the file ID update operation and reducing the consistency of the platform. In filesystems with support for extended attributes (ext2, ext3, ext4, XFS, JFS, reiserfs ...) the file ID will be attached to the resource, and when moved, the xattrs will continue to be attached to the resource (ensured by the filesystem), therefore making the update of file ID atomic and increasing the consistency of the system.

- Checksum resides along the data. Traditional FSS platforms with checksum support keep the checksum in an intermediate component as in the previous case. When a new file is uploaded, the file is saved to disk and then the checksum is kept in the database, but in the time period between these operations the file could have been already modified, thus saving a wrong checksum. Keeping the checksum in an extended attribute guarantees that the checksum will correspond to the contents of the file.

The Gamma architecture (Figure A.10c) replaces the MySQL database with Redis [21], an in-memory database in order to increase the performance of the metadata operations as it will be shown in next section.

Another aspect to highlight is the way of obtaining directory entries (listing the contents of a folder) in ClawIO, as it differs from the way is done in traditional FSS platforms. While other FSS platforms use the database for storing the file tree hierarchy, ClawIO uses for another approach with local filesystems. As the local filesystem implements a file hierarchy itself, we believed there was no need to maintain a cached copy of the file catalog in the database. As a consequence, the filesystem–to–database synchronisation job is avoided, thus allowing access to the underlying storage via external actors and still allowing users to see the latest state of their data, without having to wait until a filesystem–to–database synchronisation job finishes.

Before diving into the testing section, Figure A.11a, A.11b and A.11c (inside AppendixA) show the flow of some end–user actions (upload, mkdir and stat) between the different components that forms ClawIO.

## 5. Testing environment

To test the storage configurations, each unit runs in its own virtualised Docker[14] container. Therefore, all units have been encapsulated into Dockerfiles read by Docker Compose[15] to orchestrate the setup. All the containers run in the same physical machine (specifications reflected in Table 1) to avoid including network saturation biasses in the results.

| | |
|---|---|
| CPU | 64 cores Intel(R) Xeon(R) CPU E5-4640 v2 @ 2.20GHz |
| RAM | 64 GB |
| DISK | SAS-3 12 Gb/s 4 TB Seagate ST4000NM3401 (RAID6) |
| OS | CentOS 6.7 |

Table 1: Hardware specifications of the testing machine.

The system was stressed against with two operations (stat and upload), on different levels of concurrency to reach the maximum possible number of requests. Both operations are defined as follows:

- Stat operation: this operation is similar to a Unix 'stat' command[16] or a WebDAV PROPFIND request[17] with depth zero. It retrieves the metadata from the database and the filesystem. For each level of concurrency 10000 requests are triggered and the test is repeated 5 times to avoid biasses. This operation runs the *clawiobench* tool against the metadata server of each storage configuration using gRPC for communication.

[14] https://www.docker.com/
[15] https://docs.docker.com/compose/
[16] `http://man.he.net/?topic=stat&section=all`
[17] `https://msdn.microsoft.com/en-us/library/aa142960(v=exchg.65).aspx`

- Upload operation: this operation uploads a file randomly chosen from a fixed set of 100 files. File sizes in this sample follow a distribution of file sizes in production system CERNBox [22]. The chosen file is uploaded 5000 times per concurrency level to random target destinations to avoid overwrites. The benchmark is repeated 5 times. This operation is run by *clawiobench* tool against the data servers of each storage configuration and uses HTTP/1.1 for communication. Furthermore, this operation triggers the propagation of metadata for each uploaded file.

## 6. Results

All plots shown below measure concurrency in the x–axis and the frequency (number of requests per second) in the y–axis.

### *6.1. Stat results*

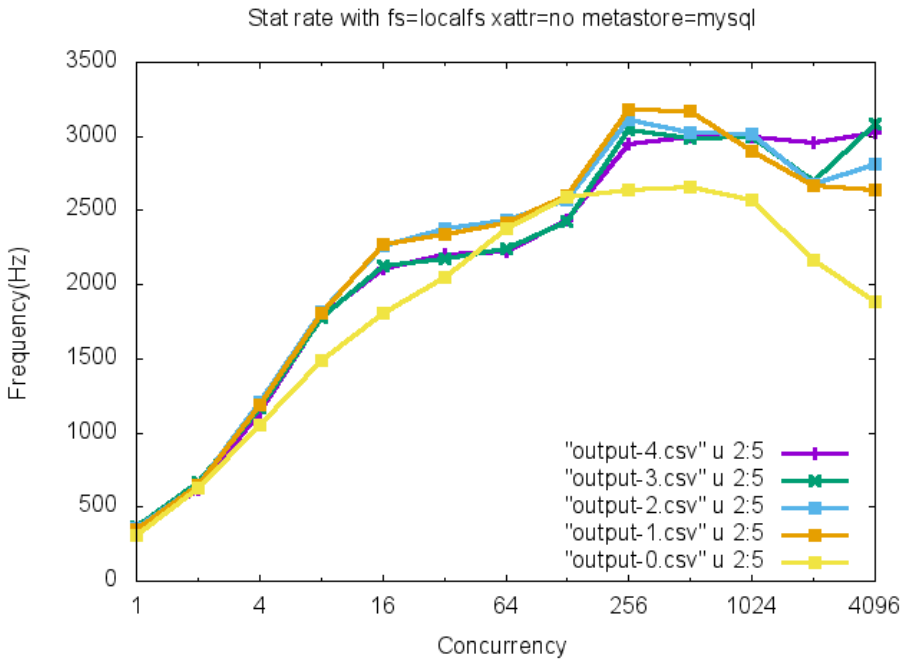



Figure 2: 'stat' rate against Alpha storage configuration.

Figure 2 shows the plot obtained after running the stat operation against the Alpha storage configuration, i.e. local filesystem with MySQL. The maximum number of requests per second observed with this configuration was around 3200. It is observed a linear increase until a concurrency of 256. Afterwards, the system stabilises.

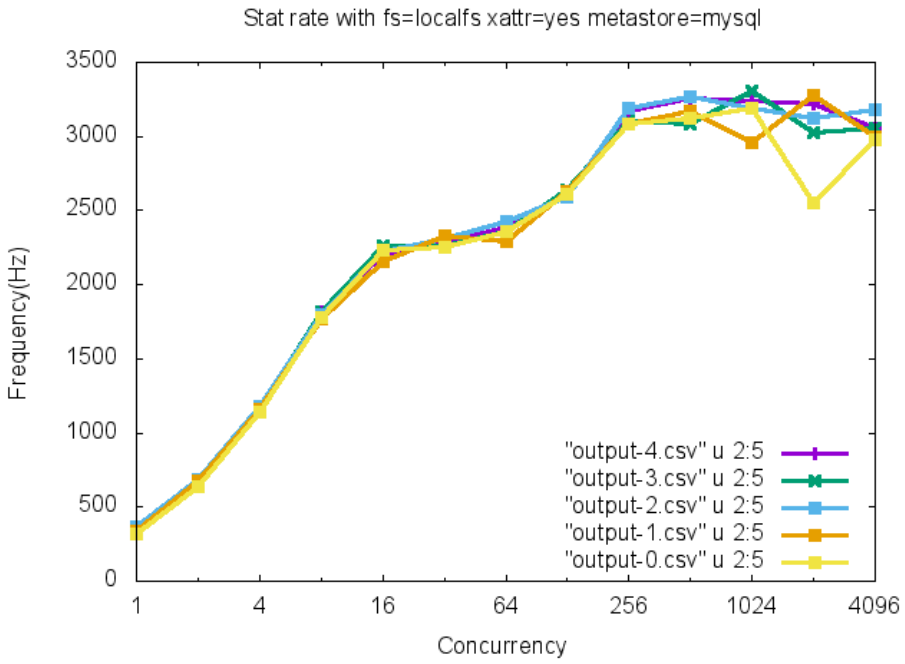



Figure 3: 'stat' rate against Beta storage configuration.

Figure 3 shows the plot obtained after running the same operation against the Beta storage configuration, i.e. local filesystem with extended attributes and MySQL. The maximum frequency is around 3400 Hz. It is observed similar shape as in the Alpha case: linear increase until 256 and stabilisation towards the end.

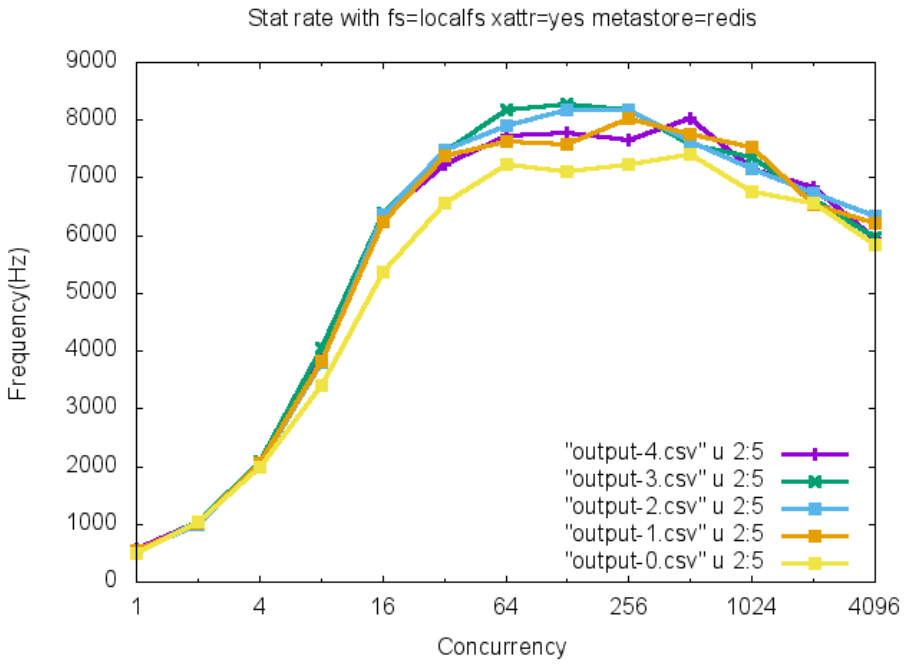



Figure 4: 'stat' rate against Gamma storage configuration.

Figure 4 shows the plot obtained after stressing the same stat operation against the Gamma storage configuration, i.e. local filesystem with extended attributes and Redis. The maximum number of requests per second observed with this configuration was around 8200, almost three times more as in the Alpha and Beta cases. The shape changes a little bit respect to the other plots. There is a pronounced linear increase until a concurrency level of 64 (the number of available processors), then a stabilisation phase until concurrency level 512 is reached, followed by a smooth continuous decay.

*6.2. Upload results*

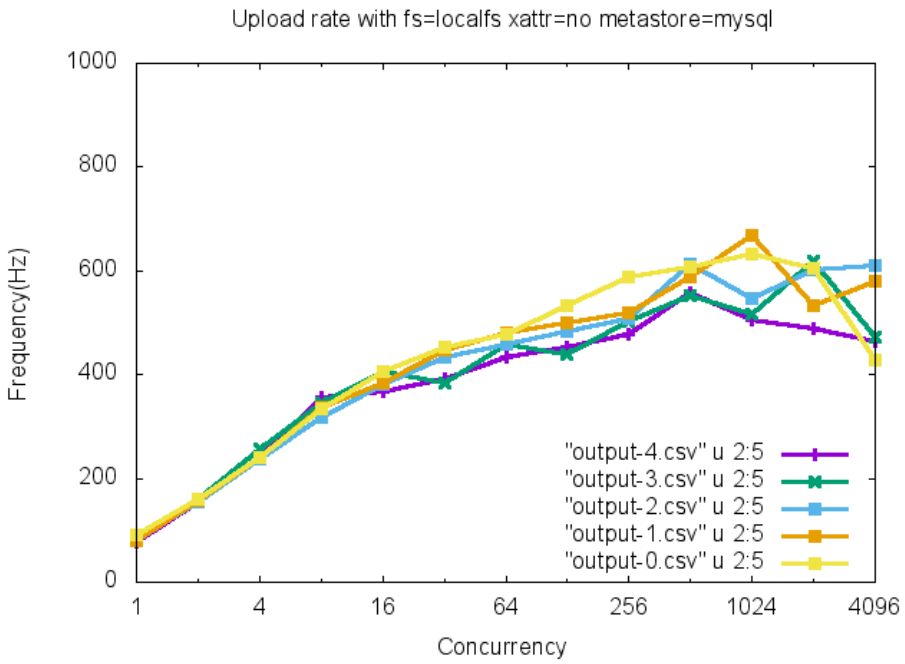



Figure 5: Upload rate against Alpha storage configuration.

Figure 5 shows the plot obtained after running the upload operation against Alpha storage configuration, i.e. local filesystem with MySQL. The maximum number of uploaded files per second observed with this configuration was close to 700. There is a linear increase until a concurrency of 512; afterwards, the system stabilises.

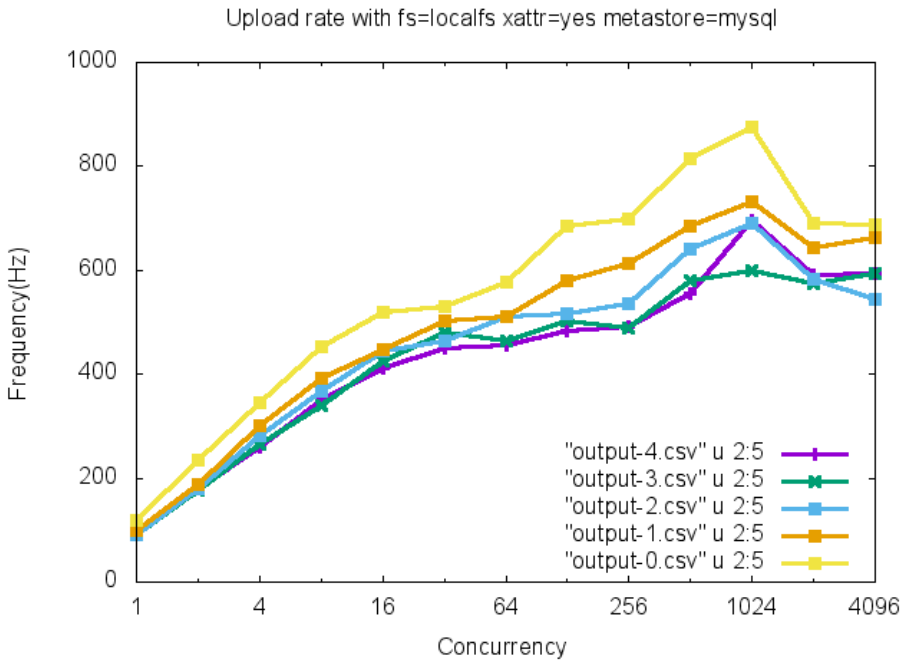



Figure 6: Upload rate against Beta storage configuration.

Figure 6 shows the plot obtained after running the same operation against the Beta storage configuration: local filesystem with extended attributes and MySQL. the maximum frequency is around 900 Hz. There is the same plot shape as in Alpha case: linear increase until 1024 concurrency and stabilisation afterwards.

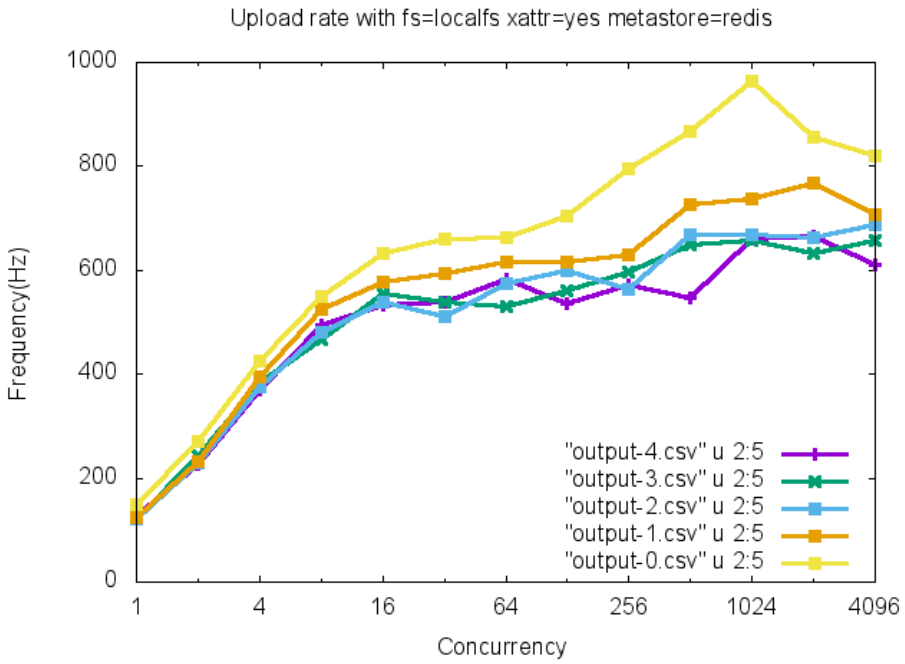



Figure 7: Upload rate against Gamma storage configuration.

Figure 7 shows the plot obtained after stressing the same upload operation against the Gamma storage configuration, i.e. local filesystem with extended attributes and Redis. The maximum number of requests per second observed with this configuration was close to 1000. There is a linear increase until a concurrency of 1024 followed by a stabilisation.

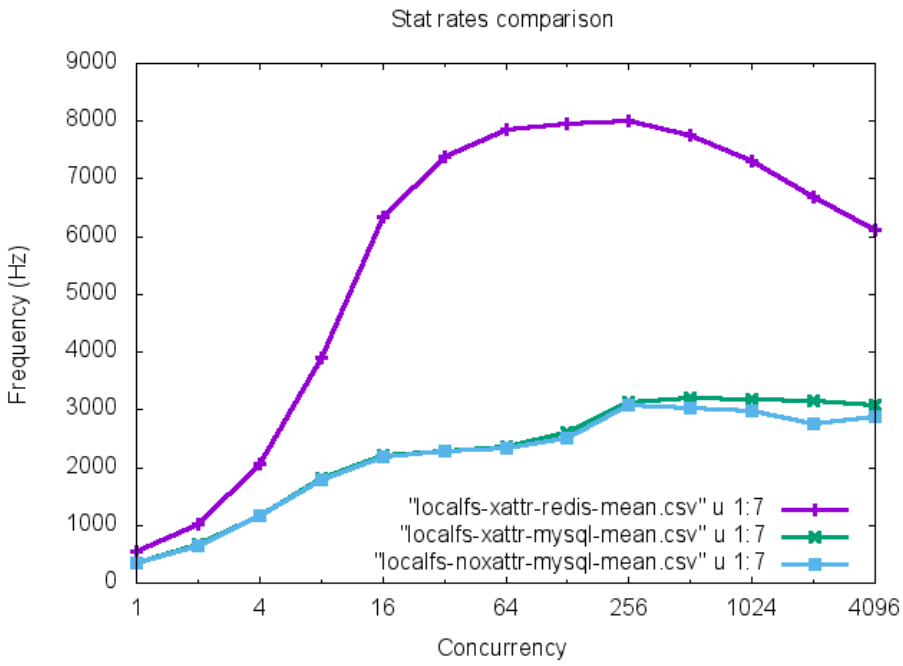



Figure 8: Average rate of 'stat' operation of Alpha, Beta and Gamma storage configurations.

Figure 8 shows a plot with the means of the stat operations previously shown in Figures 2, 3 and 4. Analysing this plot we conclude the following:

- The use of extended attributes for retrieving file IDs does not impact the performance of the system and makes the system more consistent because move operations are now atomically as explained in Section 4.
- The use of the Redis in-memory database almost triples the performance of the system when dealing with the stat operations, the most common request made from ownCloud clients – ownCloud clients poll every 30 seconds for server updates.
- Redis has memory footprint of 70 bytes per entry (including the metadata). This allows to fit large namespaces directly into memory, for example, with 64 GB RAM the system could handle close to $10^9$ files.

Figure 9 shows a plot with the means of the stat operations previously shown in figures 5, 6 and 7. From this plot we can affirm the following:

- Independently of the database type and usage or not of extended file attributes, the upload operation is mostly constraint by the underlying filesystem.

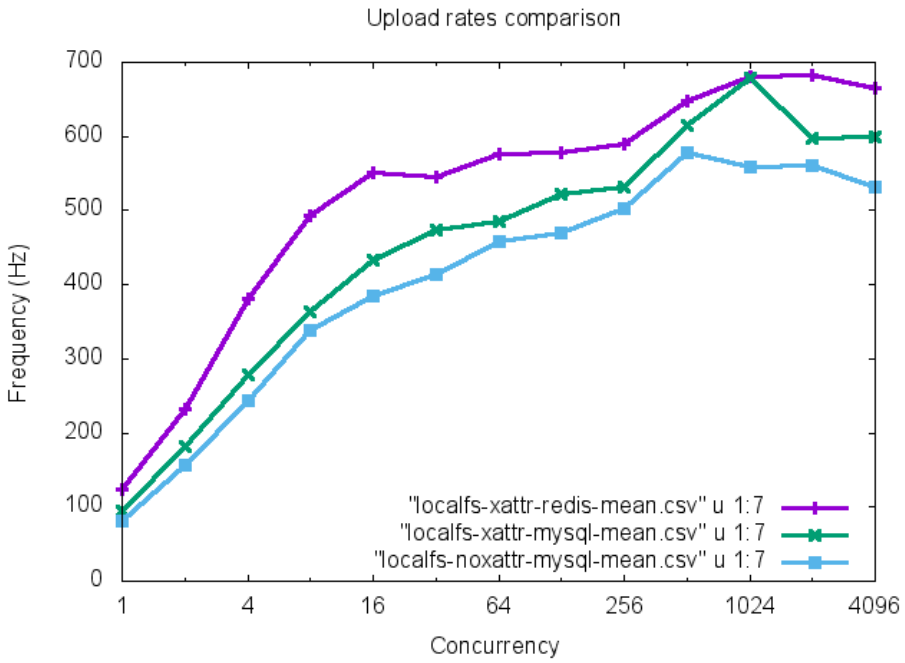



Figure 9: Average rate of upload operations of Alpha, Beta and Gamma storage configurations.

The results obtained also show that ClawIO has been designed to scale, as within one single physical machine it can handle vast amounts of requests across multiple microservices, thus maintaining the flexibility and scale-out features.

During the benchmarking phase, we had encountered a series of challenges.

In the first place, the sixteen microservices used in the testing scenario run in the same physical box, therefore the number of used file descriptors grew linearly regarding the number of connections, thus exhausting the available memory. For

this reason, we decided to run the concurrency tests until 4096 to not exhaust the number of available file descriptors.

In the second place, most plots show a stabilisation phase at the end except the 'stat' test against Redis (Figure 4). The decay at the end is caused by a bottleneck, but we are still not sure if it was the filesystem or Redis; further investigation is needed.

## 7. Conclusion and Future Work

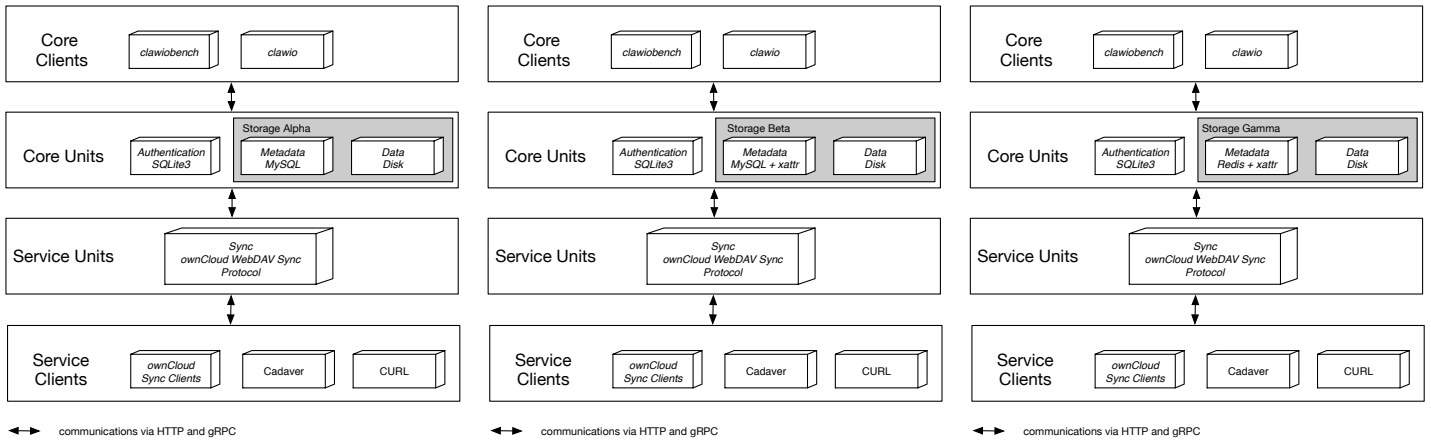
This study has introduced ClawIO as a useful framework to benchmark synchronisation protocols from storage perspective. In this phase of the project, we have observed how it is possible to build high-performance and consistent FSS platforms using in-memory databases to track fast data updates and file system extended attributes to ensure data consistency. Also, we have seen how we can provide real time access to the underlying filesystem without using intermediate namespace components like a SQL database.

Some future research challenges are as follows: benchmarking the system in different physical boxes may yield better results as the number of connections are not constrained as in one single box. Another challenge is to run the benchmark against a cluster of machines (most ClawIO components scale out) to yield even higher frequencies. Also, the storage backends were local filesystems (EXT4), but there are other alternatives like object storages or miscellaneous systems like CERN EOS being worth to be tested. It is also important to benchmark more operations and intercalations between them.

## 8. Acknowledgments

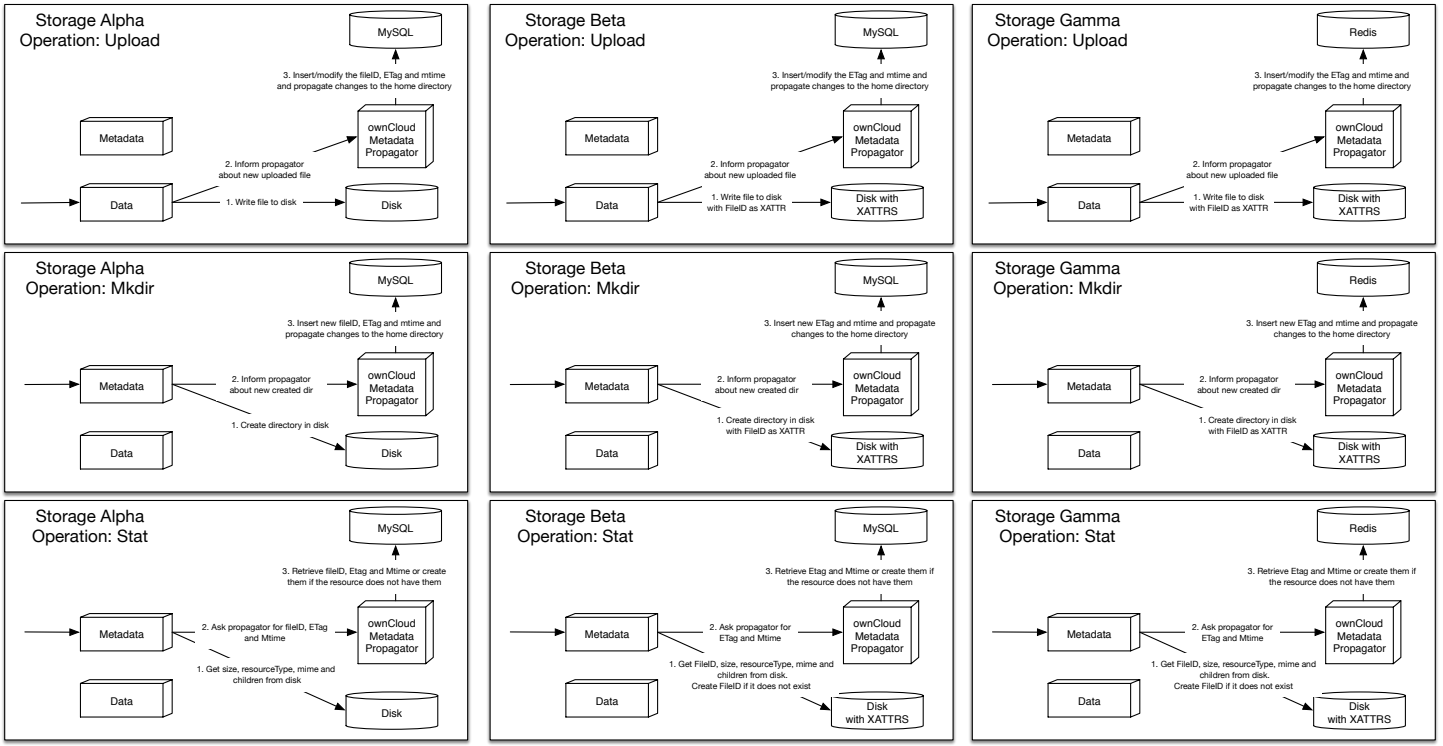
This research was partially supported by research grant "Beca de Colaboración" from Spanish Education Ministry. We thank CITI (Centro de Investigación e Innovación) for providing the physical testing machine to conduct the benchmark.

## AppendixA.




(a) Alpha architecture. (b) Beta architecture. (c) Gamma architecture.

Figure A.10: ClawIO Storage Architectures




(a) ClawIO Alpha Operations (b) ClawIO Beta Operations (c) ClawIO Gamma Operations

Figure A.11: ClawIO Operations Flow

## References


[1] Y. Cui, Z. Lai, X. Wang, N. Dai, C. Miao, 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.

[2] E. Bocchi, I. Drago, M. Mellia, Personal cloud storage benchmarks and comparison.

[3] E. Bocchi, M. Mellia, S. Sarni, Cloud storage service benchmarking: Methodologies and experimentations, in: Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on, IEEE, 2014, pp. 395–400.

[4] I. Drago, E. Bocchi, M. Mellia, H. Slatman, A. Pras, Benchmarking personal cloud storage, in: Proceedings of the 2013 conference on Internet measurement conference, ACM, 2013, pp. 205–212.

[5] P. Mrowczynski, Benchmarking and testing owncloud, seafile, dropbox and cernbox using smashbox, cloud Services for Synchronisation and Sharing (CS3) (2016).

[6] E. Bocchi, Experiences of cloud storage service monitoring: Performance assessment and comparison, cloud Services for Synchronisation and Sharing (CS3) (2016).

[7] H. Wang, R. Shea, F. Wang, J. Liu, On the impact of virtualization on dropbox-like cloud file storage/synchronization services, in: Proceedings of the 2012 IEEE 20th international workshop on quality of service, IEEE Press, 2012, p. 11.

[8] Y. Cui, L. Sun, Z. Lai, Internet Storage Sync: Problem Statement, Internet-Draft draft-cui-iss-problem-03, Internet Engineering Task Force, work in Progress (Nov. 2015).
URL `https://tools.ietf.org/html/draft-cui-iss-problem-03`

[9] A. J. Peters, L. Janyst, Exabyte scale storage at cern, in: Journal of Physics: Conference Series, Vol. 331, IOP Publishing, 2011, p. 052015.

[10] A. Peters, E. Sindrilaru, G. Adde, Eos as the present and future solution for data storage at cern, in: Journal of Physics: Conference Series, Vol. 664, IOP Publishing, 2015, p. 042042.

[11] L. Mascetti, E. Cano, B. Chan, X. Espinal, A. Fiorot, H. G. Labrador, J. Iven, M. Lamanna, G. L. Presti, J. Mościcki, et al., Disk storage at cern, in: Journal of Physics: Conference Series, Vol. 664, IOP Publishing, 2015, p. 042035.

[12] S. Newman, Building Microservices, " O'Reilly Media, Inc.", 2015.

[13] A. Sumaray, S. K. Makki, A comparison of data serialization formats for optimal efficiency on a mobile platform, in: Proceedings of the 6th international conference on ubiquitous information management and communication, ACM, 2012, p. 48.

[14] J. T. Mościcki, M. Lamanna, Prototyping a file sharing and synchronization service with owncloud, in: Journal of Physics: Conference Series, Vol. 513, IOP Publishing, 2014, p. 042034.

[15] L. Mascetti, H. G. Labrador, M. Lamanna, J. Mościcki, A. Peters, Cernbox+ eos: end-user storage for science, in: Journal of Physics: Conference Series, Vol. 664, IOP Publishing, 2015, p. 062037.

[16] E. J. Whitehead Jr, M. Wiggins, Webdav: Ieft standard for collaborative authoring on the web, Internet Computing, IEEE 2 (5) (1998) 34–40.

[17] Y. Goland, E. Whitehead, A. Faizi, S. Carter, D. Jensen, Http extensions for distributed authoring–webdav (1999).

[18] R. T. Fielding, Rest apis must be hypertext-driven, Untangled musings of Roy T. Fielding.

[19] M. Fowler, Richardson maturity model: steps toward the glory of rest, Online at http://martinfowler. com/articles/richardsonMaturityModel. html.

[20] K. Freitag, The file sync algorithm of the owncloud desktop clients, workshop on Cloud Services for File Synchronisation and Sharing (2014).

[21] S. Sanfilippo, P. Noordhuis, Redis (2009).

[22] W. Jarosz, A study of delta-sync and other optimisation in http/webdav synchronisation protocols, cloud Services for Synchronisation and Sharing (CS3) (2016).