cloudspaces

SEVENTH FRAMEWORK PROGRAMME

SEVENTH FRAMEWORK PROGRAMME

CloudSpaces

(FP7-ICT-2011-8)

Open Service Platform for the Next Generation of Personal Clouds

D2.3 Validation and Feedback analysis from open Internet trials

Due date of deliverable: 31-10-2014
Actual submission date: 15-10-2014

Start date of project: 01-10-2012 Duration: 36 months

Summary of the document

Document Type	Deliverable
Dissemination level	Public
State	Final
Number of pages	23
WP/Task related to this document	WP3
WP/Task responsible	URV
Author(s)	Raúl Gracia-Tinedo
Partner(s) Contributing	URV
Document ID	CLOUDSPACES_D2.3_141015_Public.pdf
Abstract	In this document, we present an analysis of the workload characteristics and user behavior of the UbuntuOne (U1) service. Making use of the traces supplied by Canonical, we provide interesting insights on the functioning and dynamics of the U1 service. Furthermore, we present various optimizations that we applied in the field of Personal Clouds supported by our analysis.
Keywords	Cloud storage, synchronization, sharing, interoperability, Personal Cloud

Table of Contents

1 Executive summary

In this document, we present an analysis of the U1 Personal Cloud service. Based on the traces provided by Canonical, we provide a battery of interesting insights regarding the storage workload, the file system structure and the user behavior in U1.

Thanks to our analysis, we contributed with three Personal Cloud optimizations: (i) *Advanced content distribution*, (ii) *elastic file synchronization* and (iii) *energy-efficiency for deduplication systems in the storage back-end*.

Finally, we also discuss our future research directions regarding the exploitation of the U1 traces.

2 The U1 Personal Cloud

In the context of the EU FP7 CloudSpaces project, we define a Personal Cloud as: "The Personal Cloud is a unified digital locker for users' personal data offering, at least, three key services: *storage*, *synchronization* and *sharing*". Ubuntu One (U1) falls in into this definition, like other services such as DropBox and iCloud. U1 is a suite of online services from Canonical Ltd. that enables users to store and sync files online and between computers, as well as sharing files/folders with others using file synchronization.

Until the service shutdown in July 31, U1 provided support to desktop and mobile clients and a Web front-end. U1 also integrated other Ubuntu services in its operation, like Tomboy for notes and U1 Music Store for music streaming.

2.1 Storage Protocol

In order to communicate clients and the server-side infrastructure, U1 uses its own protocol called `ubuntuone-storageprotocol` based on Google Protocol Buffers[1]. In contrast to most commercial solutions, the protocol specifications and implementation are publicly available[2].

To provide a comprehensive description of the protocol, we distinguish between *entities* and *operations*. Operations can be seen as end-user actions intended to manage one/many entities, such as a file or a directory.

2.1.1 Protocol Entities

Next, we define the main entities in the protocol. In our analysis, we characterize and quantify the role of these entities in the operation of U1.

Volume: It can be considered as a directory. During the installation of the U1 client, the client creates an initial volume to store files with id=0 (root). There are 3 types of volumes: i) *root/predefined*, ii) *udf* (user defined folder, which is a folder created by the user) and iii) *share* (sub-volume of another user to which the current user has access).

In the back-end, U1 decouples the logical file/directory objects from the actual file contents. Drawing a comparison to a file system, the inodes are stored in a PostgreSQL database and the extents are stored in Amazon S3 (see Section 2.3). The protocol supports CRUD operations on files (e.g. list volumes, put/get content, delete, etc.) to manage user volumes. The protocol itself uses UUIDs as identifiers for directory objects and their contents (files). Requests are always initiated from the client end (pull).

View: On the user's computer, a `view` is a directory that is actively being mirrored. `Views` can be shared with other users. The `view` given to another user can be either *read-only*, or *read-write*. To detect changes in a `view`, users execute a protocol operation that requests all

[1] `https://wiki.ubuntu.com/UbuntuOne`
[2] `https://launchpad.net/ubuntuone-storage-protocol`

the changes to a volume from a certain *generation* onwards. The client keeps track of the generation point, and on reconnection asks for changes from that point onwards.

Session: A user interacts with the server in the context of a U1 storage protocol session (not HTTP or any other session type). This session is used to identify the requests of a single user during the session lifetime. Sessions do not usually expire automatically. A client may disconnect, or a server process may go down, and that will end the session.

Regarding session management, an OAuth [1] token is used for authentication between the desktop and server side infrastructures. This token is stored in the platform's keyring (for Ubuntu it is `gnome-keyring`).

2.1.2 Protocol Operations

In what follows, we describe the most important protocol operations between users and the server-side infrastructure. We traced these operations to quantify the system's workload and the behavior of users.

ListVolumes: This operation is normally performed at the beginning of a session and lists all the volumes of a user (root, udf, shared).

ListShares: This operation lists all the volumes of a user that are *type share*. In this operation, the field `shared by` is the owner of the volume and `shared to` is the user to which that volume was shared with. In this operation, the field `shares` represents the number of volumes *type share* of this user.

(Put/Get)ContentResponse: these are the actual file uploads and downloads, respectively. The notification goes to the U1 back-end but the actual data is stored in a separate service (Amazon S3). A special process is created to forward the data to Amazon S3.

Make: This operation is equivalent to a "touch" operation in the U1 back-end. Normally this is associated with RPC upload jobs, in consequence of a file upload.

Unlink: Delete a file or a directory from a volume.

Move: Moves a file from one directory to another.

CreateUDF: Creates a User Defined Volume.

DeleteVolume: Deletes a volume.

GetDelta: Get the differences between the server volume and the local one (generations).

Authenticate: Operations managed by the servers to create sessions for users.

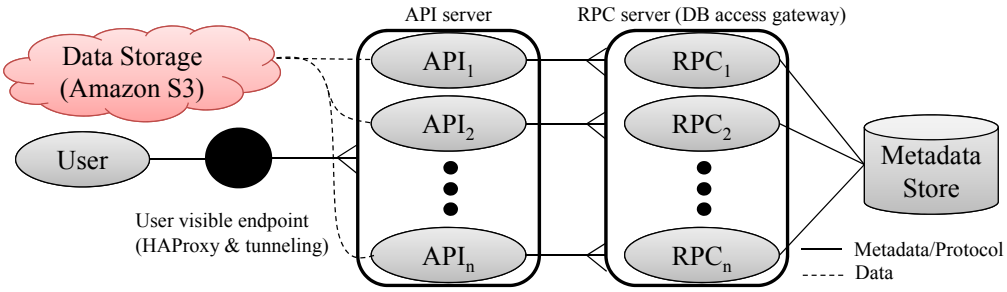


Figure 1: Architecture of U1 backend.

2.2 Synchronization Client

U1 provides a user friendly desktop client with graphical interface that enables users to manage files. Internally, it runs a daemon in the background that exposes a DBus API interface. It does the actual work of deciding what to synchronize in which direction and handles doing so.

Other services like DropBox, split files into content-based chunks attempting to minimize the network overhead related to file updates and to improve cross-user deduplication [2]. However, U1 does not apply chunking to uploaded files. This decision is based on the observation that most updates are directed to small files (e.g. documents), whereas large ones normally remain unmodified. Instead, U1 resorts to file-based cross-user deduplication to reduce the waste of storing repeated files [3]. To detect duplicated files, U1 desktop clients provide to the server the SHA-1 hash of a file prior to the content upload. U1 engineers reported storage savings of 11% with the introduction of this deduplication scheme.

By default, one folder called `~/Ubuntu One/` is automatically created and configured for mirroring (root `volume`) during the client installation. Changes to this folder (and any others added) are watched using `inotify`. Synchronization metadata about directories being mirrored is stored in `~/.cache/ubuntuone`. When remote content has changed, the agent acts on the incoming unsolicited notification sent by API slave and starts downloading. The implementation is written in Python (GPLv3).

2.3 Back-end Infrastructure

The entire U1 back-end is all inside a single datacenter and its objective is to manage the service metadata. The back-end architecture appears in Fig. 1 and consists of *metadata servers* (API/RPCDB), *metadata store* and *data store*.

The gateway to the back-end servers is the load balancer. The load balancer (HAProxy, ssl, etc.) is the visible endpoint for users and it is composed by two racked servers. Beyond the load balancer we find the API and RPC database servers that run on separate racked servers. API servers receive commands from the user, perform authentication, and translate the commands into RPCDB calls. Subsequently, RPC database workers translate RPCDB calls into database queries and route queries to the appropriate database shards. API/RPCDB processes are more numerous than physical machines, so that they can migrate

among machines for load balancing purposes.

U1 stores metadata in a PostgreSQL database cluster composed by 20 large Dell racked servers, configured in 10 shards (master-slave). Internally, the system routes operations by user identifier to the appropriate shard. Thus, metadata of a user's files and folders reside always in the same shard. This data model effectively exploits sharding, since normally there is no need to lock more than one shard per operation (i.e. lockless). Only operations related to shared files/folders may require to involve more than one shard in the cluster.

As other popular Personal Clouds, such as DropBox or SugarSync, U1 advocates to store user files in a separate cloud service. Concretely, U1 resorts to Amazon S3 (`us-east`) to store user data. This solution enables a service to rapidly scale out without a heavy investment in storage hardware. In its latests months of operation, U1 had a $\approx 20,000$\$ monthly bill in storage resources (1 Petabyte), thus becoming the most important Amazon S3 client in Europe.

With this infrastructure, U1 scaled up to 4 million registered users and 0.5 million of daily active users.

3 Dataset and Methodology

We present a back-end analysis of the U1 service. In contrast to many measurements of Cloud services [2, 4], we did not deploy a measurement infrastructure to analyze the service externally. Instead, we inspected directly the U1 metadata servers to build our trace. This has been done in collaboration with Canonical Ltd. in the of the project under review.

3.1 Measurement methodology

The traces are taken at both *API server* and *RPCDB server* stages. In the former stage we collected important information about the storage workload (e.g. mimetypes, file size), whereas the second stage provided us valuable information about the internal life-cycle of requests.

We built the trace capturing a series of service logfiles. Each logfile corresponds to the entire activity of a single API/RPCDB process in a machine for a period of time. Each logfile is within itself strictly sequential and timestamped. Therefore, causal ordering is ensured for operations done for the same user. However, the timestamp between servers is not dependable, even though machines are synchronized with NTP (clock drift may be in the order of ms). The mapping between services and servers is dynamic within the time frame of analyzed logs.

To gain better understanding on this, consider a line in the trace with this logname: `production-whitecurrant-23-20140128`. They will all be `production`, because we only looked at production servers. After that prefix is the name of the physical machine, followed by the number of the server process. The server process number can migrate between nodes to load balance. After that is the date the logfile was "cut" (there is one log file per server/service and day).

Normally there are between $8-16$ processes per physical machine. The identifier of the process is unique within a machine, although it can migrate from one machine to another. Sharding is in the metadata storage backend, so it is behind the point where traces are taken. This means that in these traces any combination of server/process can handle any user. To have a strictly sequential notion of the activity of a user we should take into account the U1 *session* and sort the trace by timestamp (a user may have more than one parallel connection). A session starts in the least loaded machine and lives in the same node until it finishes, making user events strictly sequential. Thanks to this information we can estimate system and user service times.

Finally, user names and identifiers have been anonimyzed by Canonical prior to this analysis. Approximately 1% of traces have not been analyzed due to failures in parsing of the logs.

Sample Trace duration	32 hours
Unique users	320K
Unique files	1.6M
Unique volumes	19K
Total upload traffic	2.6TB

Table 3.1a: Summary of the trace sample used in this analysis.

3.2 Dataset

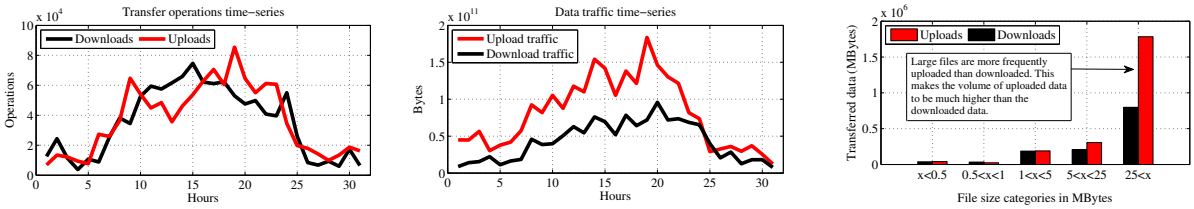
The trace is the result of merging all the logfiles of the U1 servers for 28 days (1.9TB of `.csv` text). To ease the analysis, we split the trace in days, sorting the operations by timestamp. A summary of a trace sample used in this analysis is shown in Table 3.1a[3].

The trace contains the API operations (request type `storage/storage_done`) and their translation into RPC calls (request type `rpc`), as well as the session management of users (request type `session`). This provides different sources of valuable information. For instance, we can analyze the *storage workload* supported by a massive and real-world cloud service (users, files, operations). Since we captured file properties such as file size, hash and folder we can study the storage system in high detail. This trace also provides information about the behavior of users in this kind of system, such as their activity and interactions. This information may guide us to propose new system optimizations.

Dataset limitations. We mentioned that timestamps among servers are not dependable since they may be different (in order of ms). This may lead to certain (but not dramatic) bias on the calculation of system time-based metrics. We only gathered information from desktop clients. This means that other sources of traffic in U1 (e.g. web front-end, mobile clients) are not included in this analysis, since they are handled by different software stacks that were not logged. Finally, this service is mainly used as an online backup, so the analysis of sharing among users is limited.

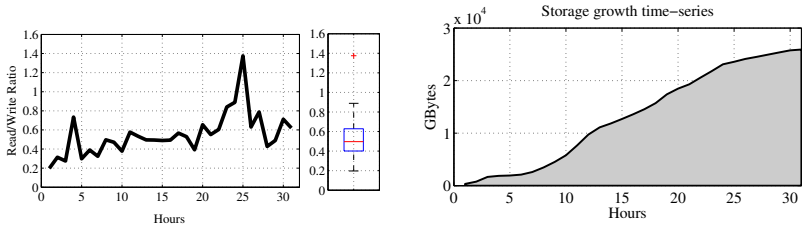
The traces and the necessary documentation to work with them will be publicly available for the community after the publication of the corresponding research analysis. We believe

[3]Sample of 32 hours of the trace corresponding to 1/3 of the servers.



(a) Transfer operations time-series. (b) Transferred traffic time-series. (c) Traffic related to files of a certain size category.

Figure 2: Macroscopic characterization of transfers in U1.



(a) Read/write ratio time-series. (b) Storage growth time-series.

Figure 3: Storage growth and read/write ration in U1.

that this trace represents an unprecedented opportunity for researchers to study in depth a widely used Personal Cloud.

4 Storage Workload

4.1 Daily Usage

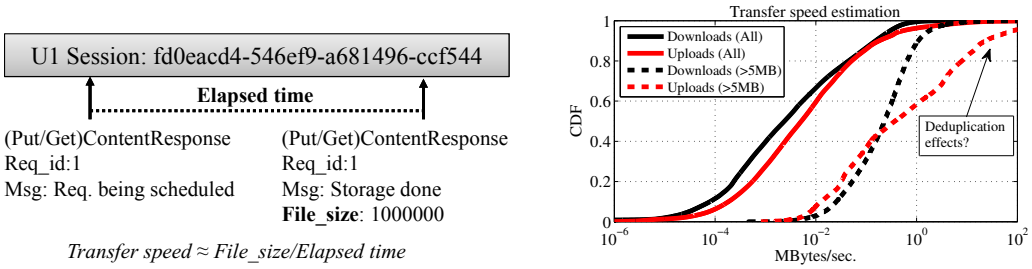
In this section, we focus on macroscopic workload metrics to understand the usage of U1.

Fig. 2(a) provides a time-series view of the storage operations received during one day. We observe in Fig. 2(a) that U1 exhibits important *daily patterns*. To wit, the volume of uploads per hour can be x10 times higher in the central day hours compared to the nights. This observation is aligned with previous works, that detected time-based variability in the usage and performance of Personal Cloud services [2, 4]. This effect is probably related to the work habits of users, since U1 desktop clients are by default initiated automatically when users turn on their machines (see Section 6.1).

Fig. 2(b) shows that U1 users upload more data to the system than they download (write-dominated workload). In this trace, the amount of uploaded data (2.34TB) is 46% higher than the downloaded data (1.26TB, costing 149.9$ based on Amazon S3 pricing[4]). This suggests that a large fraction of users make use of U1 as a backup application.

However, if we compare figures 2(a) and 2(b), we observe that the dominance of uploads is more significant with respect to the traffic than for the number operations. The underlying reason has to do with the size of transferred files.

[4] `http://calculator.s3.amazonaws.com/`



(a) Transfer speed estimation methodology. (b) Transfer speed distribution.

Figure 4: Transfer speed estimation of uploads/downloads in U1.

In this sense, Fig. 2(c) illustrates the amount of transferred data for files of predefined size ranges. As we can observe, small files account only for a small fraction of the total traffic, being similar for both uploads and downloads. However, users store more larger files than they retrieve ($>$ 25MBytes), increasing the total amount of upload traffic.

Although the storage workload is clearly write-dominated, we want to inspect the variability of the read/write (r/w) ratio. This ratio represents the relationship between the incoming and outgoing data in a period of time (i.e. hour).

Fig. 3(a) illustrates that the variability of the read/write ratio can be important. In fact, during the same day the r/w ratio can exhibit a difference of x8 comparing the maximum and minimum values. We also observe that this ratio increases during the morning/night hours of the day, indicating more presence of download traffic. The most plausible reason is that users initiate the U1 service with unsynchronized folders in those day times, that in many cases require file downloads. This peak of downloads increases the r/w ratio. Such a phenomenon can be exploited to reduce bandwidth costs with sophisticated content distribution techniques [5].

The dominance of upload traffic has an immediate effect on the storage demands that U1 should satisfy. In Fig. 3(b) we depict the growth of data that is stored in U1 per hour. We observe that in a single day, U1 stores in Amazon S3 around 2.34TB of data. Based on Amazon S3 pricing, keeping such amount of data represents an expense of 71.05$/month to U1. The ever-increasing cost in storage might be hard to amortize and it could have been one of the probable reasons for Canonical to interrupt the U1 service.

4.2 Estimating Transfer Speeds

Next, we focus on estimating the transfer speeds that users experienced, which is an important indicator of the service quality. We compare our results with previous measurements of Amazon S3, in order to contrast our results.

The logs collected describe the transfer life-cycle between users and the U1 service. We are interested on measuring the client transfer speed, which can be tracked by inspecting `Put/GetContentResponse` events (see Section 2.1.2).

To estimate the speed of client transfers, we elaborated the following methodology (see Fig. 4(a)). We analyze the sessions of users, since they provide a strictly sequential view of

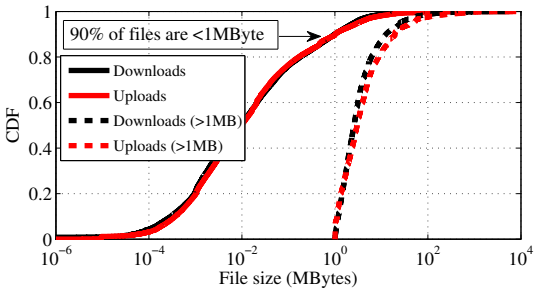


Figure 5: File size distribution.

their activity. A transfer (`XContentResponse`) starts by notifying in the message field that the request is being scheduled. Moreover, log operations contain request identifiers that remain until they are completed. Thus, we keep track of the session and request identifiers, as well as the timestamp. Subsequently, we find a new event with the same session and request identifiers, notifying that the transfer has finished. This event also contains the size of the transferred file. In that point, we estimate the transfer speed by dividing the time lapse between both events by the file size.

Upon the reception of a new transfer, U1 instantiates a new process to store/retrieve the file contents to/from Amazon S3. Such a process may take lapse of time that might affect the estimation accuracy for small files. However, for medium/large files (e.g. $>$ 5MBytes) our method can provide an acceptable degree of accuracy. In this sense, Fig. 4(b) shows the distribution of transfer speeds at file granularity, i.e. the result of dividing the file size by the time needed to transfer it.

Taking into account the distribution of all files, in Fig. 4(b) we observe that in general uploads are slightly faster than downloads. This observation is in line with our previous active measurement of Personal Cloud APIs [4]. However, the differences are not dramatic since most of the values are related to very small files, where transfers are underutilized and the synchronization overhead is relatively important. This is also a reason for the poor transfer speeds exhibited in these distributions.

Expectedly, for larger files we see that the distribution is completely different, exhibiting higher transfer speed values. In fact, around the 90% download transfer speeds fall between 0.01 and 1 MByte/sec. Furthermore, these results are much more heterogeneous than our previous measurement, where file transfer heterogeneity was between 0.2 and 0.8 MBytes/sec. In our view, the transfer heterogeneity in our trace is due to the quality of clients' Internet connections and their geographic locations.

Finally, we observe that an important fraction of file uploads present a very fast transfer speed, deviating importantly from the download distribution. Although in our previous measurement we observed that Personal Cloud API services exhibit faster uploads than downloads, the deviation exceeds our previous results [4]. The most likely reason is that U1 deduplicates files for many uploads, thus avoiding to actually perform the data transfer. This makes upload transfer speeds to be abnormally high in many cases.

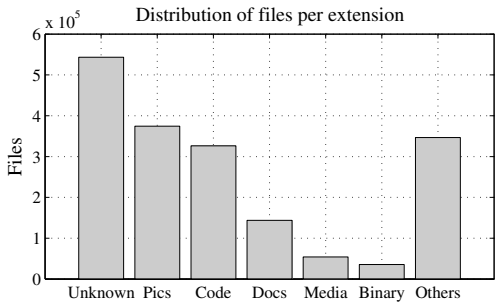


Figure 6: Number of files per category.

4.3 File-centric Usage

In this section, we focus on file-centric workload characteristics that can help us to understand the behavior of files in the system, potentially guiding system optimizations.

First, we focus on the file size distribution in U1. Fig. 5 shows the distribution of transferred files in the system. At first glance, we realize that the *vast majority of files are small*. To wit, 90% of files are smaller than 1MByte. In our view, this can have important implications on the performance of the back-end storage system. The reason is that Personal Clouds like U1 use object storage services offered by Cloud providers as data back-end, which are not specifically optimized for very small files. We discuss this in Section 7.

In Fig. 5 we also depict the size distribution of transferred files larger than 1MByte. Unlike the distribution of all file sizes, larger files tend to be more related to uploads than to downloads. This is also supported by the fact that upload traffic is dominant.

5 File System

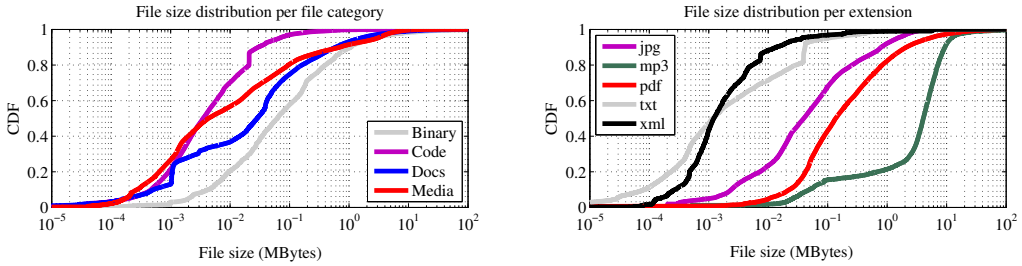
Next, we analyze the structure of the U1 file system. Unlike a traditional file system, U1 stores file metadata in a database cluster to build a virtual organization of files, directories and volumes. The actual file contents are stored in Amazon S3. Users send file system management operations through their clients that are received via the U1 API front-end. In the trace sample, we identified around 1.6M files, 206K directories and 19K non-root volumes.

5.1 File Taxonomy

In this section, we analyze the characteristics of files that users store in U1.

Fig. 6 depicts the number of files grouped in the most relevant categories that we identified by mime-type[5]. In Fig. 6 we observe that the most relevant file category identified is Pictures (`.jpg`, `.png`, etc.). This is not surprising, since Personal Clouds are specifically designed to manage personal information of users.

[5]In our traces, file mime-types can only be extracted from `Make` operations.



(a) Size of files per category. (b) Size of files per extension.

Figure 7: Characterization of file sizes.

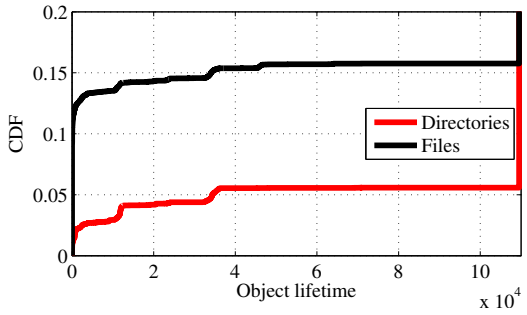


Figure 8: Lifetime of files and directories in the trace sample.

Interestingly, we also found that the Code category is also very popular (`.html`, `.java`, etc.). This suggests that there is an important fraction of users that are code developers. In fact, editing code in a synchronized folder represents a stress test for the synchronization protocol, as well as a huge network overhead in many cases. This observation may lead to specifically tune the synchronization protocol for this kind of users [6].

In this sense, figures 7(a) and 7(b) show the size distribution of files per category and extension. Fig. 7(a) shows that the most popular file categories may not be responsible for the largest part of consumed storage. That is, we can observe that the median size of binary objects is around 50 times larger than code files. Thus, although binary files are less common in number, they occupy a much larger fraction of the storage capacity of U1.

In Fig. 7(b) we observe the distribution of file sizes for the most common file extensions found. Clearly, this provides interesting clues to emulate the size of files for realistic Personal Cloud benchmarks [7].

5.2 Object Lifetime

In this section, we focus on the lifetime of file system objects (files, directories) within a volume (see Fig. 8).

Fig. 8 illustrates and interesting fact: the 16% of files and the 5.5% of directories are deleted within 32 hours. This observation can be used to optimize the write path of files within a storage service.

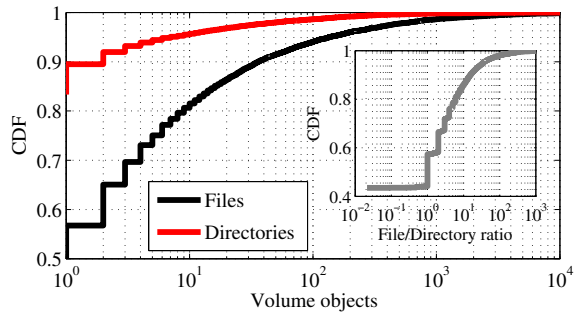


Figure 9: Number of detected files and directories, as well as the ratio of files/directories per volume.

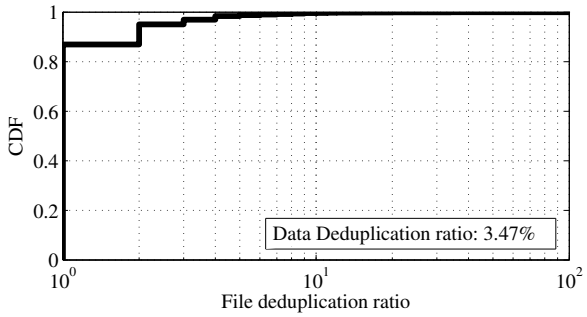


Figure 10: File-level deduplication ratio.

Moreover, Fig. 8 shows that most of the objects are deleted just after their creation (specially for files). It is also interesting to note the correlation between the deletion of directories and files. Actually, deleting a directory triggers a deletion in cascade of all the files inside it.

5.3 Volume Content

Next, we analyze the content of volumes, in terms of files and directories (see Fig. 9). Concretely, Fig. 9 the fraction of volumes in which we identified files and during during the trace sample analyzed.

Unsurprisingly, Fig. 9 shows that files are much more numerous than directories. That is, in the trace sample, around 50% of volume identifiers has been associated with at least one volume. This percentage is 17% in the case of directories.

Another interesting issue is how files and directories are distributed across directories. To this end, the inner plot in Fig. 9 shows the ratio of files w.r.t. directories within a volume. Clearly, this ratio is dominated by files meaning that we can find many more files than directories within a volume. Only for a small fractions of volumes ($< 2\%$) this ratio is lower than 1.

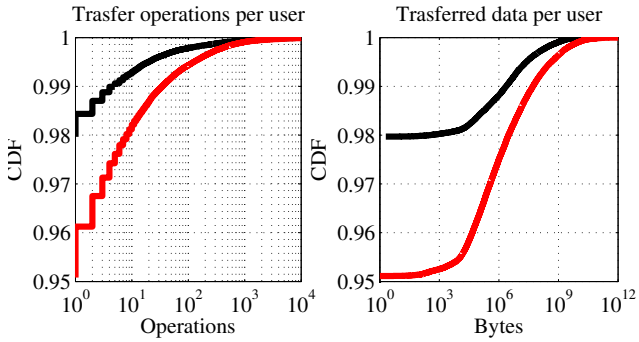


Figure 11: Storage operations and transferred data per user.

5.4 Deduplication Estimation

Deduplication is a popular technique used to increase the storage capacity of a system by removing duplicated data, at either chunk or file levels. U1 implements a file-based cross-user deduplication scheme. This means that, upon the arrival of a client upload operation, the system checks if the file to be uploaded already exists (e.g. by *file hash*). In the affirmative case, the user is *logically linked* to the existing file and the actual data transfer does not take place. To make this possible, desktop clients send the hash of the file to be uploaded to the server. In our interviews with U1 engineers, this deduplication scheme saved 11% of Cloud storage resources.

In Fig. 10 we provide a simple example of the effectiveness of deduplication. In the trace sample, we found a deduplication ratio of 3.47 only within a 30 hour time window. Clearly, the deduplication ratio is more important as more data is already stored in the system. However, we cannot get the actual system-wide deduplication ratio since it would require from having all the file hashes already stored in U1.

Moreover, we can observe that the distribution of file deduplication ratios is *heavy tailed*. This means that a small number of files account for very high number of duplicates (e.g., popular songs), whereas most files have a very low number of duplicates.

6 Modeling User Behavior

Understanding the behavior of users is a key source of information to optimize large-scale systems. This section provides several insights about the behavior of users in U1.

6.1 User Activity

In this section, we analyze the interactions of users with the U1 service.

In terms of storage operation (data transfers), Fig. 11 shows the distribution of uploads/downloads that users performed during the trace sample. We observe that, as stated

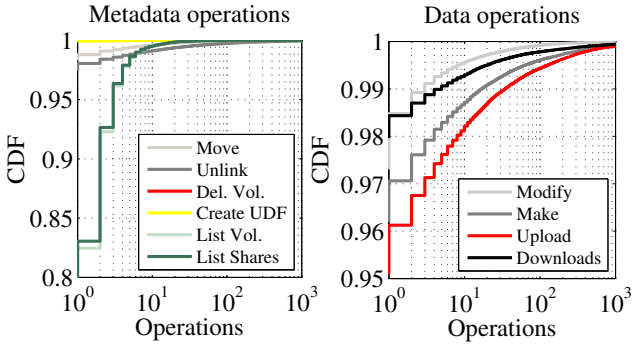


Figure 12: Protocol operations per user.

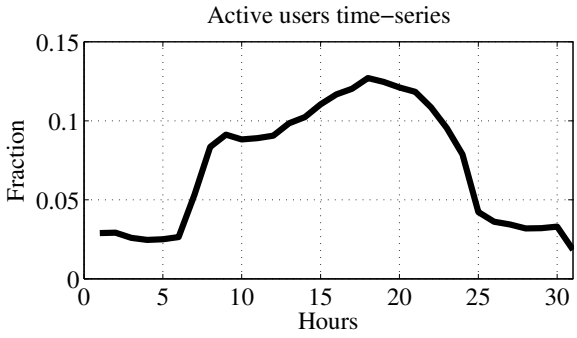


Figure 13: Fraction of active users per hour.

above, users perform more uploads than downloads. This observation hold for both the number of storage operations and for the amount of data transferred.

Fig. 12 provides a wider view of the interactions of users with the server-side infrastructure. We observe that the different protocol operations present a very disparate behavior. That is, there are operations such as deleting/creating volumes which are uncommon, whereas operations such as listing volumes are very frequent —they are executed when the desktop client is started. Thus, if common operations are not optimized in terms of performance, they can become an important system bottleneck.

Furthermore, both Figures 11 and 12 present heavy tailed distributions. This means that there is a small fraction of users exhibit a really high activity, whereas most user remain almost inactive. This observation can be potentially used to identify and handle the most active users in a specific way to optimize the system.

6.2 Population Dynamics

In what follows, we analyze the dynamics of the U1 user population. In the trace sample analyzed, we identified 320K users, which we believe is a representative number.

Fig. 13 depicts a time-series view of the fraction of active users per hour with respect to the total number of users detected. We consider a user as active if its desktop client has exchanged at least one message with the server within one hour slot.

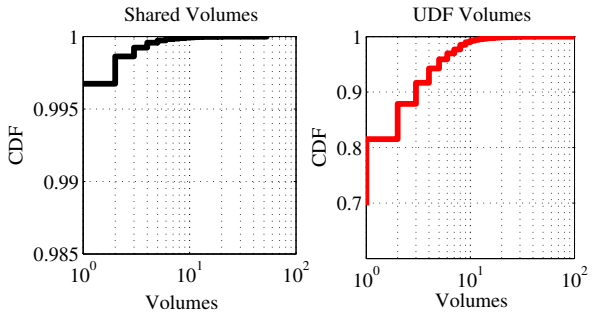


Figure 14: Distribution of shared/udf volumes across users.

In Fig. 13 we observe that the population exhibits remarkable daily patterns. That is, in the central hours of the day there can be up to x5 times more users than during the nights. This is due to the working and connection habits of users, which can be predicted to perform a better workload consolidation on the server side.

Moreover, we see that a significant fraction of users do not interact with the service during the day, which can suggest that at that point other Personal Clouds provided more attractive services than U1. This means that the market is currently dominated by very big players, whereas smaller Personal Clouds have problems to compete.

6.3 Sharing

Sharing is a main feature of Personal Clouds. In Fig. 14 we observe the number of udf and shared volumes of users.

As observed by Canonical engineers, sharing is not a popular feature in U1. Fig. 14 shows that for only the 0.003% of users we could detect at least one shared volume. Although analyzing the full trace may increase this number, we observe that udfs volumes are much more popular for the same trace period.

The conclusion is that U1 has been used more as an on-line backup service rather than for working collaboratively among users.

7 Optimizations for Personal Clouds

Understanding the operation of a real-world Personal Cloud like U1 is only the first part of our objective. In this section, we describe a number of system optimizations for Personal Clouds that are supported by the insights of our analysis.

7.1 Content Distribution

Cloud storage services have become very popular these days as a paradigm that enables individuals and organizations to store, edit and retrieve data stored in remote servers and

which can be accessed all over the Internet. Such systems are generally equipped with a set of features that allow sharing and collaboration between the users. That is why, nowadays, millions of users are putting their files into online cloud storage systems (like Dropbox, Google Drive or Box...) in order to be able to access them wherever they go or as a backup. When a client adds a new file to his personal folder, the content is divided into small entities called "chunk" that are pushed to the storage servers while the meta-data is kept in a different server. Once the file is uploaded, all the synchronized devices will be notified of the addition of the new file and will request a copy of it from the cloud storage servers. This will result in the repeated distribution of the same content in a short period of time. To avoid that, the cloud can benefit from the clients upload capacities to save its bandwidth. In this context, we propose to introduce the BitTorrent [8] protocol when the load on a specific file becomes high. In fact, the efficiency of the peer-assisted paradigm makes it especially suitable for files shared between a set of devices. In such scenarios, it is possible to benefit from the common interest of users in the same file and use their upload bandwidth to offload the cloud from doing all the serving.

Unfortunately, the use of BitTorrent may incur a longer download time compared to HTTP especially for small files. According to previous studies, these small files form a very high percentage of the data stored in personal cloud servers; 99% of the files are of size smaller than 16 MB according to [2]. The main challenge is to decide when it is worth switching to BitTorrent. The key elements in making the decision are the gain in download time and the peers' contribution. The former represents the difference in download time between HTTP and BitTorrent. The latter measures the total amount of data that can be obtained from the peers. To our knowledge, there were no previous studies that compare the BitTorrent and HTTP protocols for distributing small files. Thus, there is the need to draw a complete comparative study between both protocols. We first study the download time in HTTP and BitTorrent and measure the efficiency of each protocol. Then, we propose a dynamic switching algorithm that can be applied in real personal cloud systems. Our key contributions are the following:

1. We conduct a comparative study between BitTorrent and HTTP. We first confute the general statement that BitTorrent is not effective for small files based on a real experimental study. Then, we propose an analytical estimation of the distribution time in BitTorrent that takes into account the overheads related to the nature of the protocol. In addition, we introduce two general metrics to decide when it is better to use one protocol with respect to the other: the gain and the offload ratios. The gain measures the degree of improvement in download time of BitTorrent relative to HTTP. The offload ratio quantifies the amount of data that can be offloaded if the peers adopt BitTorrent. We validate all the proposed formulas with focus on small files.

2. We propose a dynamic algorithm for the decision of the most appropriate download protocol. The algorithm uses simple parameters that can be collected by the system and predicts the efficacy of HTTP and BitTorrent for each case. The most suitable protocol is decided based on the predefined constraints.

3. We analyze a public trace of the Ubuntu One system and study the users and files characteristics and the access patterns. Later, we apply our algorithm on the trace and measure the amount of data that can be offloaded based on different time constraints. We notice that the overall offloaded data volume exceeds 16% of the total amount of

data exchanged. From an economic point of view, this corresponds to savings of the order of hundreds to thousands of dollars per month. We also study the effect of file bundling on the trace and notice that it can only improve the overall offload by a small increment.

This work has been published in IEEE P2P 2014 under the title "Reducing Costs in the Personal Cloud: Is BitTorrent a Better Bet?".

7.2 Elastic Synchronization for Personal Clouds

In the last years, we have witnessed a rush of Personal Cloud Storage services offering file synchronization to millions of users. In this line, Dropbox [2] has achieved massive scalability thanks to a decoupled architecture that separates control flows (Dropbox sync servers) from data flows (Amazon S3 Object Storage). While the elasticity of Cloud Object Storage Services like Amazon S3 is ensured, the design of elastic and scalable file synchronization protocols is complex [9]. Among the major challenges, we outline the following two issues: fine-grained programmable elasticity and efficient change notification to millions of users. The first challenge is related to the observation that scaling up some types of cloud applications is not straightforward using traditional VM resource utilization metrics (CPU, RAM, etc.) [10], because, for instance, they are not CPU or memory intensive, but I/O bound, as is the case for file synchronization [9]. In those cases, it is better to rely on metrics such as the average and message handling response times exhibited by VM instances to cope with the varying demand. This implies that fine-grained elasticity management components must be built for the synchronization service as argued in the paper.

The other challenge is that the high read-write ratio of file syncing services makes it more suitable to make use of one-to-many push communication for rapid notification. Analogously, to efficiently maintain the consistency of files, any change performed elsewhere must be advertised as soon as they occur to reduce conflicts [2], in particular, when a file is susceptible to be modified by more than one client at the same time. This requires the file syncing service to operate as quickly as possible to commit changes, along with an efficient notification service to inform clients about file mutations.

To face the above challenges, we propose a novel architecture for elastic file synchronization. The major contributions of our work are:

1. ObjectMQ: a lightweight framework for providing programmatic elasticity to distributed objects using message queues as their underlying communication middleware. The efficient use of indirect communication in our middleware removes the need for preprocessing client stubs for scaling out and down, it provides transparent load balancing mechanisms based on queues, it simplifies one-to-many communications, and it enables flexible programmatic elasticity based on queue message processing.

2. StackSync: an elastic file synchronization architecture decoupling metadata and data flows in structured and object storage services. StackSync implements predictive and reactive provisioning policies on top of ObjectMQ that adapt to real traces from the

Ubuntu One service. Furthermore, the ObjectMQ unicast and multicast communication primitives have considerably simplified the code of the synchronization protocol. It also enables efficient change notification in a transparent way on top of the underlying messaging service.

3. StackSync has been extensively tested using real traces from the Ubuntu One system to validate its elasticity and efficient use of resources. Furthermore, we extended an open benchmark [4] for Personal Clouds which provides trace generators and test scripts. Using this benchmark, we compared our service with Dropbox, Box, OneDrive, and Google Drive. StackSync is a stable open source project after two years of development that is being used in several public institutions and data centers.

This work has been published in ACM/IFIP/USENIX Middleware 2014 under the title "StackSync: Bringing Elasticity to Dropbox-like File Synchronization".

7.3 Energy-awareness for In-line Deduplication Systems

Nowadays, data volumes to be managed by enterprises and data-centers are growing at an exponential rate [11]. This motivates researchers to devise novel techniques to improve storage efficiency. In this sense, *data deduplication*, a technique intended to eliminate data redundancies by splitting files into smaller and indexed chunks for avoiding storing repeated ones, has attracted much attention from both industry and academia. Particularly, *in-line deduplication clusters* are emerging and becoming increasingly popular, since they reproduce the operation of a single-node deduplication system at a larger scale [3, 12, 13, 14, 15].

Inherently, the design of in-line deduplication clusters is geared towards high performance and scalability. To wit, in a deduplication cluster, the proxy *exploits parallel access* to storage nodes depending on the *data routing algorithm*. Thus, superchunks belonging to a file are addressed based on their content towards different storage nodes [12, 13, 15]. This design leverages horizontal scalability since storage nodes can be dynamically added to the cluster to increase capacity and throughput. Existing commercial products provide high performance in-line deduplication services that are increasingly fitting the needs of scalable storage and archival of enterprises and organizations [16, 17, 18].

However, in terms of energy, the unintended consequence of this design is that *it may prevent storage nodes' disks from remaining in standby mode for larger periods to save up energy* [19, 20]. To wit, Fig. 15 (left) shows the fraction of activated nodes storing a single file in a simulated deduplication cluster. Clearly, Fig. 15 suggests that a large fraction of nodes may be in accessed in parallel on each request.

The actual problem arises if we consider the *variation on workload intensity* typically found in enterprises and organizations as a result of users' habits [21, 4] (see Fig. 15, right). The parallel and randomized nature of data routing in a deduplication cluster makes it difficult for hard disks by themselves to *fully exploit load valleys to save up energy* by switching to low-power mode. This may lead to a potentially high expense in disk energy, since disks of storage nodes would be *kept idle even during low load periods*.

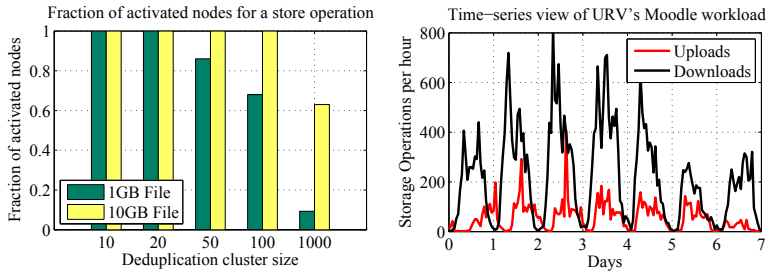


Figure 15: Fraction of nodes involved in a store operation in a deduplication cluster (10MB superchunks) (left). Workload supported by URV's Moodle servers (right).

Our motivation is to make deduplication clusters able to exploit load valleys to save up disk energy. To this end, we *investigate the feasibility of deferred writes, diverted access and workload consolidation* in this particular context. Technically, we contribute with:

- We propose an *analytical model to estimate the energy savings bring out by deferred writes in a deduplication cluster*. We provide insights about when deferring writes in a subset of cluster nodes pays off.
- We propose a *workload consolidation algorithm* based on our analytical model. Based on the workload, the algorithm dynamically diverts writes to a subset of nodes (or not) *considering the potential energy savings*.

To materialize our insights, we present `eWave`: a novel energy-efficient storage middleware targeted at supporting existing in-line deduplication systems. The main goal of `eWave` lies on enabling the energy-aware operation of deduplication clusters through a *temporary energy-aware data management that takes place during low load periods*. `eWave` does not modify the deduplication layer (e.g. data routing, deduplication rate) and it can be integrated with most existing systems that use stateless data routing [3, 12, 13, 14, 15]. Through extensive simulations and experiments in an 8-machine cluster, we conclude that `eWave` is able to achieve energy savings from 16% to 60% in several scenarios with moderate impact on performance during low load periods.

This work has been published in ACM SYSTOR 2014 under the title "eWave: Leveraging Energy-Awareness for In-line Deduplication Clusters".

8 Related Work

The performance evaluation of Cloud storage services is an interesting topic with several papers appearing recently. The authors in [22] explore the performance of Microsoft Azure, including storage. In this line, the authors in [23] execute an extensive measurement against Amazon S3 to elucidate whether Cloud storage is suitable for scientific Grids or not. Similarly, [24] presents a performance analysis of the Amazon Web Services, with no insights regarding Personal Clouds.

File hosting and file sharing Cloud services have been analyzed in depth by several works [25, 26]. They provide an interesting substrate to understand both the behavior of users and the QoS of major providers (e.g. RapidShare).

Closer to our work, the term Personal Cloud has gained momentum in the latest years. Personal Clouds are built on top of traditional cloud storage to provide advanced services to users: *file storage*, *synchronization* and *sharing*. Despite their commercial popularity, only few research works have turned attention to measure and analyze the performance of Personal Cloud storage services [27, 2, 4].

The first work to specifically analyze Personal Cloud services we are aware of is [27]. Hu et. al. [27] compare Dropbox, Mozy, Carbonite and CrashPlan backup services. However, their analysis of the performance, reliability and security levels is rather lightweight, and more work is needed to characterize these services with enough rigor. In this sense, authors in [4] provide an extensive analysis of the REST interface provided by three major Personal Clouds, analyzing important aspects of their QoS (e.g. variability, failures).

Recently, the authors in [2] presented an extensive measurement of DropBox in two scenarios: in a university campus and in residential networks. They analyzed and characterized the traffic transmitted by users, as well as the functioning and architecture of the service. Authors in [28] also modeled the client behavior of DropBox users.

Conversely to most works, instead of measuring a Personal Cloud from outside, we analyzed the *metadata back-end servers* of UbuntuOne. This approach provides to us a global view of the service, as well as golden opportunities to characterize important aspects of UbuntuOne (e.g. storage workload, user behavior). Furthermore, we believe that our insights will help to guide future works on improving several aspects of these services, such as synchronization [6, 29], security [30] and bandwidth reduction [5].

9 Conclusions and Future Directions

In this document, we analyzed in depth the characteristics of the U1 service. We analyzed the storage workload, the file system and the behavior of users, among other aspects. This analysis revealed important insights that can be common in other Personal Clouds, which may motivate potential optimizations from the research community.

Furthermore, thanks to our analysis, we contributed with three Personal Cloud optimizations: (i) *Advanced content distribution*, (ii) *elastic file synchronization* and (iii) *energy-efficiency for deduplication systems in the storage back-end*.

Our current and future research directions are the following ones:

- *Analysis of the U1 back-end*: At this moment, we analyzed the workload of U1 in various domains. However, we also gathered information about the internal metadata back-end which we believe important to understand. We are currently preparing the whole trace to work on this issue. Our insights in this regard may also motivate optimizations in the metadata back-end.

- *Personal Cloud Benchmarking Framework*: We are developing an integrated benchmarking framework for Personal Clouds that largely extend the existing proposals [7]. Among other novelties, we feed our benchmarking framework with U1 traces to replay real workloads in the Personal Clouds under test.

References

[1] E. Hammer-Lahav, "The OAuth 1.0 Protocol," http://tools.ietf.org/html/rfc5849, 2010.

[2] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: understanding personal cloud storage services," in ACM SIGCOMM IMC'12, 2012, pp. 481–494.

[3] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in MASCOTS'09, 2009, pp. 1–9.

[4] R. Gracia-Tinedo, M. Sánchez-Artigas, A. Moreno-Martınez, C. Cotes, and P. Garcıa-López, "Actively measuring personal cloud storage," in IEEE CLOUD'13, 2013, pp. 301–308.

[5] R. Chaabouni, P. García-López, and M. Sánchez-Artigas, "Reducing costs in the personal cloud: Is bittorrent a better bet?" in IEEE P2P'14, 2014, pp. 1–10.

[6] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai, "Efficient batched synchronization in dropbox-like cloud storage services," in ACM/IFIP/USENIX Middleware'13, 2013, pp. 307–327.

[7] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, "Benchmarking personal cloud storage," in ACM SIGCOMM IMC'13, 2013, pp. 205–212.

[8] B. Cohen, "Incentives build robustness in bittorrent," in Workshop on Economics of Peer-to-Peer systems, vol. 6, 2003, pp. 68–72.

[9] "How we have scaled dropbox," https://www.youtube.com/watch?v=PE4gwstWhmc.

[10] K. Jayaram, "Elastic remote methods," in Middleware 2013. Springer, 2013, pp. 143–162.

[11] T. Bostoen, S. Mullender, and Y. Berbers, "Power-reduction techniques for data-center storage systems," ACM Computing Surveys, vol. 45, pp. 33:1–33:38, 2011.

[12] D. Frey, A.-M. Kermarrec, and K. Kloudas, "Probabilistic deduplication for cluster-based storage systems," in SoCC'12, 2012, pp. 17:1–17:14.

[13] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, and Y. Wan, "Debar: A scalable high-performance de-duplication storage system for backup and archiving," in IPDPS'10, 2010, pp. 1–12.

[14] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," in FAST'11, 2011, pp. 15–29.

[15] C. Dubnicki et. al., "Hydrastor: A scalable secondary storage," in FAST'09, 2009, pp. 197–210.

[16] "Emc centera," http://www.emc.com.

[17] "Sepaton s2100," http://www.sepaton.com.

[18] "Nec hydrastor," http://www.necam.com/hydrastor/.

[19] J. Leverich and C. Kozyrakis, "On the energy (in) efficiency of hadoop clusters," ACM SIGOPS Op. Sys. Review, vol. 44, no. 1, pp. 61–65, 2010.

[20] Z. Li, K. M. Greenan, A. W. Leung, and E. Zadok, "Power consumption in enterprise-scale backup storage systems," in USENIX FAST'12, 2012, pp. 1–6.

[21] A. W. Leung, S. Pasupathy, G. R. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads." in USENIX ATC'08, vol. 1, no. 2, 2008, pp. 213–226.

[22] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of windows azure," in HPDC '10, 2010, pp. 367–376.

[23] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in DADC'08, 2008, pp. 55–64.

[24] A. Bergen, Y. Coady, and R. McGeer, "Client bandwidth: The forgotten metric of online storage providers," in IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2011, pp. 543–548.

[25] J. Sanjuàs-Cuxart, P. Barlet-Ros, and J. Solé-Pareta, "Measurement based analysis of one-click file hosting services," Journal of Network and Systems Management, vol. 20, pp. 276–301, 2012.

[26] D. Antoniades, E. P. Markatos, and C. Dovrolis, "One-click hosting services: a file-sharing hideout," in ACM SIGCOMM IMC'09, 2009, pp. 223–234.

[27] W. Hu, T. Yang, and J. Matthews, "The good, the bad and the ugly of consumer cloud storage," ACM SIGOPS Operating Systems Review, vol. 44, no. 3, pp. 110–115, 2010.

[28] G. Gonçalves, I. Drago, A. P. C. da Silva, A. B. Vieira, and J. M. Almeida, "Modeling the dropbox client behavior," in Proceedings of the International Conference on Communications, ICC, vol. 14, 2014.

[29] P. García-López, S. Toda-Flores, C. Cotes-González, M. Sánchez-Artigas, and J. Lenton, "Stacksync: Bringing elasticity to dropbox-like file synchronization," in ACM/IFIP/USENIX Middleware'14, 2014, p. In press.

[30] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E.Weippl., "Dark clouds on the horizon: Using cloud storage as attack vector and online slack space," in USENIX Security, 2011, pp. 5–8.