

Dissecting UbuntuOne: Autopsy of a Global-scale Personal Cloud Back-end

Raúl Gracia-Tinedo
Universitat Rovira i Virgili
raul.gracia@urv.cat

Hamza Harkous
EPFL
hamza.harkous@epfl.ch

Marc Sánchez-Artigas
Universitat Rovira i Virgili
marc.sanchez@urv.cat

Yongchao Tian
Eurecom
yongchao.tian@eurecom.fr

John Lenton
Canonical Ltd.
john.lenton@canonical.com

Josep Sampé
Universitat Rovira i Virgili
josep.sampe@urv.cat

Pedro García-López
Universitat Rovira i Virgili
pedro.garcia@urv.cat

Marko Vukolić
IBM Research - Zurich
mvu@zurich.ibm.com

ABSTRACT

Personal Cloud services, such as Dropbox or Box, have been widely adopted by users. Unfortunately, very little is known about the internal operation and general characteristics of Personal Clouds since they are proprietary services.

In this paper, we focus on understanding the nature of Personal Clouds by presenting the *internal structure* and a *measurement study* of UbuntuOne (U1). We first detail the U1 architecture, core components involved in the U1 metadata service hosted in the datacenter of Canonical, as well as the interactions of U1 with Amazon S3 to outsource data storage. To our knowledge, this is the first research work to describe the internals of a large-scale Personal Cloud.

Second, by means of tracing the U1 servers, we provide an extensive analysis of its *back-end activity* for one month. Our analysis includes the study of the *storage workload*, the *user behavior* and the performance of the U1 *metadata store*. Moreover, based on our analysis, we suggest improvements to U1 that can also benefit similar Personal Cloud systems.

Finally, we contribute our dataset to the community, which is the first to contain the back-end activity of a large-scale Personal Cloud. We believe that our dataset provides unique opportunities for extending research in the field.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Measurement techniques;
K.6.2 [Management of Computing and Information Systems]: Installation management—Performance and usage measurement

Keywords

Personal cloud; performance analysis; measurement

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

IMC'15, October 28–30, 2015, Tokyo, Japan.

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-3848-6/15/10 ...\$15.00.

DOI: <http://dx.doi.org/10.1145/2815675.2815677>.

1. INTRODUCTION

Today, users require ubiquitous and transparent storage to help handle, synchronize and manage their personal data. In a recent report [1], Forrester research forecasts a market of \$12 billion in the US related to personal and user-centric cloud services by 2016. In response to this demand, Personal Clouds like Dropbox, Box and UbuntuOne (U1) have proliferated and become increasingly popular, attracting companies such as Google, Microsoft, Amazon or Apple to offer their own integrated solutions in this field.

In a nutshell, a Personal Cloud service offers automatic backup, file sync, sharing and remote accessibility across a multitude of devices and operating systems. The popularity of these services is based on their easy to use Software-as-a-Service (SaaS) storage facade to ubiquitous Infrastructure-as-a-Service (IaaS) providers like Amazon S3 and others.

Unfortunately, due to the proprietary nature of these systems, very little is known about their performance and characteristics, including the workload they have to handle daily. And indeed, the few available studies have to rely on the so-called “black-box” approach, where traces are collected from a single or a limited number of measurement points, in order to infer their properties. This was the approach followed by the most complete analysis of a Personal Cloud to date, the measurement of Dropbox conducted by Drago et al. [2]. Although this work describes the overall service architecture, it provides no insights on the operation and infrastructure of the Dropbox’s back-end. And also, it has the additional flaw that it only focuses on small and specific communities, like university campuses, which may breed false generalizations.

Similarly, several Personal Cloud services have been externally probed to infer their operational aspects, such as data reduction and management techniques [3, 4, 5], or even transfer performance [6, 7]. However, from external vantage points, it is impossible to fully understand the operation of these systems without fully reverse-engineering them.

In this paper, we present results of our study of U1: the Personal Cloud of Canonical, integrated by default in Linux Ubuntu OS. Despite the shutdown of this service on July 2014, the distinguishing feature of our analysis is that it has been conducted using data collected by the provider itself. U1 provided service to 1.29 million users at the time of the study on January–February 2014, which constitutes the first complete analysis of the performance of a Personal Cloud in

<i>UbuntuOne Analysis</i>	Finding	Implications and Opportunities
Storage Workload (§ 5)	90% of files are smaller than 1MByte (P).	Object storage services normally used as a cloud service are not optimized for managing small files [8].
	18.5% of the upload traffic is caused by file updates (C).	Changes in file metadata cause high overhead since the U1 client does not support delta updates (e.g. .mp3 tags).
	We detected a deduplication ratio of 17% in one month (C).	File-based cross-user deduplication provides an attractive trade-off between complexity and performance [5].
	DDoS attacks against U1 are frequent (N).	Further research is needed regarding secure protocols and automatic countermeasures for Personal Clouds.
User Behavior (§ 6)	1% of users generate 65% of the traffic (P).	Very active users may be treated in an optimized manner to reduce storage costs.
	Data management operations (e.g., uploads, file deletions) are normally executed in long sequences (C).	This correlated behavior can be exploited by caching and prefetching mechanisms in the server-side.
	User operations are <i>bursty</i> ; users transition between long, idle periods and short, very active ones (N).	User behavior combined with the user per-shard data model impacts the metadata back-end load balancing.
Back-end Performance (§ 7)	A 20-node database cluster provided service to 1.29M users without symptoms of congestion (N).	The <i>user-centric data model</i> of a Personal Cloud makes relational database clusters a simple yet effective approach to scale out metadata storage.
	RPCs service time distributions accessing the metadata store exhibit long tails (N).	Several factors at hardware, OS and application-level are responsible for poor tail latency in RPC servers [9].
	In short time windows, load values of API servers/DB shards are very far from the mean value (N).	Further research is needed to achieve better load balancing under this type of workload.

C: Confirms previous results, P: Partially aligned with previous observations, N: New observation

Table 1: Summary of some of our most important findings and their implications.

the wild. Such a unique data set has allowed us to reconfirm results from prior studies, like that of Drago et al. [2], which paves the way for a general characterization of these systems. But it has also permitted us to expand the knowledge base on these services, which now represent a considerable volume of the Internet traffic. According to Drago et al. [2], the total volume of Dropbox traffic accounted for a volume equivalent to around one third of the YouTube traffic on a campus network. We believe that the results of our study can be useful for both researchers, ISPs and data center designers, giving hints on how to anticipate the impact of the growing adoption of these services. In summary, our contributions are the following:

Back-end architecture and operation of U1. This work provides a comprehensive description of the U1 architecture, being the first one to also describe the back-end infrastructure of a real-world vendor. Similarly to Dropbox [2], U1 decouples the storage of *file contents* (data) and *their logical representation* (metadata). Canonical only owns the infrastructure for the metadata service, whereas the actual file contents are stored separately in Amazon S3. Among other insights, we found that U1 API servers are characterized by long tail latencies and that a sharded database cluster is an effective way of storing metadata in these systems. Interestingly, these issues may arise in other systems that decouple data and metadata as U1 does [10].

Workload analysis and user behavior in U1. By tracing the U1 servers in the Canonical datacenter, we provide an extensive analysis of its *back-end activity* produced by the *active user population* of U1 for one month (1.29M distinct users). Our analysis confirms already reported facts, like the execution of user operations in long sequences [2] and the potential waste that file updates may induce in the system [4, 5]. Moreover, we provide new observations, such as a taxonomy of files in the system, the modeling of burstiness in user operations or the detection of attacks to U1, among others. Table 1 summarizes some of our key findings.

Potential improvements to Personal Clouds. We suggest that a Personal Cloud should be aware of the *behavior of users* to optimize its operation. Given that, we discuss the implications of our findings to the operation of U1. For instance, file updates in U1 were responsible for 18.5% of upload traffic mainly due to the lack of delta updates in the desktop client. Furthermore, we detected 3 DDoS attacks

in one month, which calls for further research in automatic attack countermeasures in secure and dependable storage protocols. Although our observations may not apply to *all* existing services, we believe that our analysis can help to improve the next generation of Personal Clouds [10, 4].

Publicly available dataset. We contribute our dataset (758GB) to the community and it is available at <http://cloudspaces.eu/results/datasets>. To our knowledge, this is the first dataset that contains the back-end activity of a large-scale Personal Cloud. We hope that our dataset provides new opportunities to researchers in further understanding the internal operation of Personal Clouds, promoting research and experimentation in this field.

Roadmap: The rest of the paper is organized as follows. § 2 provides basic background on Personal Clouds. We describe in § 3 the details of the U1 Personal Cloud. In § 4 we explain the trace collection methodology. In § 5, § 6 and § 7 we analyze the storage workload, user activity and back-end performance of U1, respectively. § 8 discusses related work. We discuss the implications of our insights and draw conclusions in § 9.

2. BACKGROUND

A Personal Cloud can be loosely defined as a unified digital locker for users' personal data, offering at least three key services: *file storage*, *synchronization* and *sharing* [11]. Numerous services such as Dropbox, U1 and Box fall under this definition.

From an architectural viewpoint, a Personal Cloud exhibits a 3-tier architecture consisting of: *(i)* clients, *(ii)* synchronization or metadata service and *(iii)* data store [2, 10]. Thus, these systems explicitly decouple the management of file contents (data) and their logical representation (metadata). Companies like Dropbox and Canonical only own the infrastructure for the metadata service, which processes requests that affect the virtual organization of files in user volumes. The contents of file transfers are stored separately in Amazon S3. An advantage of this model is that the Personal Cloud can easily scale out storage capacity thanks to the "pay-as-you-go" cloud payment model, avoiding costly investments in storage resources.

In general, Personal Clouds provide clients with 3 main types of access to their service: Web/mobile access, Representational State Transfer (REST) APIs [7, 12] and desktop

<i>API Operation</i>	<i>Related RPC</i>	<i>Description</i>
<code>ListVolumes</code>	<code>dal.list_volumes</code>	This operation is normally performed at the beginning of a session and lists all the volumes of a user (root, user-defined, shared).
<code>ListShares</code>	<code>dal.list_shares</code>	This operation lists all the volumes of a user that are of type <i>shared</i> . In this operation, the field <i>shared by</i> is the owner of the volume and <i>shared to</i> is the user to which that volume was shared with. In this operation, the field <i>shares</i> represents the number of volumes type <i>shared</i> of this user.
(Put/Get)Content	see appendix A	These operations 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. Since the upload management in U1 is complex, we refer the reader to appendix A for a description in depth of upload transfers.
<code>Make</code>	<code>dal.make_dir</code> <code>dal.make_file</code>	This operation is equivalent to a “touch” operation in the U1 back-end. Basically, it creates a file node entry in the metadata store and normally precedes a file upload.
<code>Unlink</code>	<code>dal.unlink_node</code>	Delete a file or a directory from a volume.
<code>Move</code>	<code>dal.move</code>	Moves a file from one directory to another.
<code>CreateUDF</code>	<code>dal.create_udf</code>	Creates a user-defined volume.
<code>DeleteVolume</code>	<code>dal.delete_volume</code>	Deletes a volume and the contained nodes.
<code>GetDelta</code>	<code>dal.get_delta</code>	Get the differences between the server volume and the local one (generations).
<code>Authenticate</code>	<code>auth.get_user_id_from_token</code>	Operations managed by the servers to create sessions for users.

Table 2: Description of the most relevant U1 API operations.

clients. Our measurements in this paper focus on the desktop client interactions with U1. Personal Cloud desktop clients are very popular among users since they provide automatic synchronization of user files across several devices (see Section 3.3). To achieve this, desktop clients and the server-side infrastructure communicate via a *storage protocol*. In most popular Personal Cloud services (e.g., Dropbox), such protocols are proprietary.

U1 Personal Cloud was a suite of online services offered by Canonical that enabled users to store and sync files online and between computers, as well as to share files/folders with others using file synchronization. Until the service was discontinued in July 2014, U1 provided desktop and mobile clients and a Web front-end. U1 was integrated with other Ubuntu services, like Tomboy for notes and U1 Music Store for music streaming.

3. THE U1 PERSONAL CLOUD

In this section, we first describe the U1 storage protocol used for communication between clients and the server-side infrastructure (Sec. 3.1). This will facilitate the understanding of the system architecture (Sec. 3.2). We then discuss the details of a U1 desktop client (Sec. 3.3). Finally, we give details behind the core component of U1, its metadata back-end (Sec. 3.4).

3.1 U1 Storage Protocol

U1 uses its own protocol (`ubuntuone-storageprotocol`) based on TCP and Google Protocol Buffers¹. In contrast to most commercial solutions, the protocol specifications and client-side implementation are publicly available². Here, we describe the protocol in the context of its *entities* and *operations*. Operations can be seen as end-user actions intended to manage one/many entities, such as a file or a directory.

3.1.1 Protocol Entities

In the following, we define the main entities in the protocol. Note that in our analysis, we characterize and identify the role of these entities in the operation of U1.

Node: Files and directories are *nodes* in U1. The protocol supports CRUD operations on nodes (e.g. list, delete, etc.). The protocol assigns Universal Unique Identifiers (UUIDs) to both node objects and their contents, which are generated in the back-end.

¹<https://wiki.ubuntu.com/UbuntuOne>

²<https://launchpad.net/ubuntuone-storage-protocol>

Volume: A volume is a container of node objects. 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) *user defined folder* (UDF), which is a volume created by the user, and iii) *shared* (sub-volume of another user to which the current user has access).

Session: The U1 desktop client establishes a TCP connection with the server and obtains U1 storage protocol session (not HTTP or any other session type). This session is used to identify a user’s requests during the session lifetime. Usually, sessions do not expire automatically. A client may disconnect, or a server process may go down, and that will end the session. To create a new session, an OAuth [13] token is used to authenticate clients against U1. Tokens are stored separately in the Canonical authentication service (see § 3.4.1).

3.1.2 API Operations

The U1 storage protocol offers an API consisting of the *data management* and *metadata operations* that can be executed by a client. Metadata operations are those operations that do not involve transfers to/from the data store (i.e., Amazon S3), such as listing or deleting files, and are entirely managed by the synchronization service. On the contrary, uploads and downloads are, for instance, typical examples of data management operations.

In Table 2 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.

3.2 Architecture Overview

As mentioned before, U1 has a 3-tier architecture consisting of *clients*, *synchronization service* and the *data/metadata store*. Similarly to Dropbox [2], U1 decouples the storage of *file contents* (data) and *their logical representation* (metadata). Canonical only owns the infrastructure for the metadata service, which processes requests that affect the virtual organization of files in user volumes. The actual contents of file transfers are stored separately in Amazon S3.

However, U1 treats *client requests* differently from Dropbox. Namely, Dropbox enables clients to send requests *either to the metadata or storage service* depending on the request type. Therefore, the Dropbox infrastructure only processes metadata/control operations. The cloud storage service manages data transfers, which are normally orchestrated by computing instances (e.g. EC2).

In contrast, U1 receives *both metadata requests and data transfers* of clients. Internally, the U1 service discriminates client requests and contacts either the metadata store or the storage service. For each upload and download request, a new back-end process is instantiated to manage the data transfer between the client and S3 (see appendix A). Therefore, the U1 model is simpler from a design perspective, yet this comes at the cost of delegating the responsibility of processing data transfers to the metadata back-end.

U1 Operation Workflow. Imagine a user that initiates the U1 desktop client (§ 3.3). At this point, the client sends an **Authenticate** API call (see Table 2) to U1, in order to establish a new session. An API server receives the request and contacts to the Canonical authentication service to verify the validity of that client (§ 3.4.1). Once the client has been authenticated, a persistent TCP connection is established between the client and U1. Then, the client may send other management requests on user files and directories.

To understand the synchronization workflow, let us assume that two clients are online and work on a shared folder. Then, a client sends an **Unlink** API call to delete a file from the shared folder. Again, an API server receives this request, which is forwarded in form of RPC call to a RPC server (§ 3.4). As we will see, RPC servers translate RPC calls into database query statements to access the correct metadata store shard (PostgreSQL cluster). Thus, the RPC server deletes the entry for that file from the metadata store.

When the query finishes, the result is sent back from the RPC server to the API server that responds to the client that performed the request. Moreover, the API server that handled the **Unlink** notifies the other API servers about this event that, in turn, is detected by the API server to which the second user is connected. This API server notifies via push to the second client, which deletes that file locally. The API server finishes by deleting the file also from Amazon S3.

Next, we describe in depth the different elements involved in this example of operation: The desktop client, the U1 back-end infrastructure and other key back-end services to the operation of U1 (authentication and notifications).

3.3 U1 Desktop Client

U1 provides a user friendly desktop client, implemented in Python (GPLv3), with a graphical interface that enables users to manage files. It runs a daemon in the background that exposes a message bus (DBus) interface to handle events in U1 folders and make server notifications visible to the user through OS desktop. This daemon also does the work of deciding what to synchronize and in which direction to do so.

By default, one folder labeled `~/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 changes, the client acts on the incoming unsolicited notification (push) sent by U1 service and starts the download. Push notifications are possible since clients establish a TCP connection with the metadata service that remains open while online.

In terms of data management, Dropbox desktop clients deduplicate data at chunk level [2]. In contrast, U1 resorts to file-based cross-user deduplication to reduce the waste of storing repeated files [5]. Thus, to detect duplicated files, U1 desktop clients provide to the server the SHA-1 hash of a file prior to the content upload. Subsequently, the system checks if the file to be uploaded already exists or not. In the

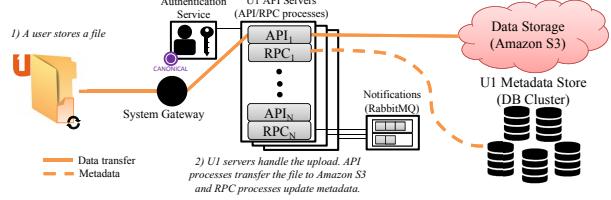


Figure 1: Architecture and workflow of U1 back-end.

affirmative case, the new file is *logically linked* to the existing content, and the client does not need to transfer data.

Finally, as observed in [5], the U1 client applies compression to uploaded files to optimize transfers. However, the desktop client does not perform techniques such as file bundling³, delta updates and sync deferment in order to simplify its design, which may lead to inefficiencies.

3.4 U1 Metadata Back-end

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

System gateway. 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 of two racked servers.

Metadata store. U1 stores metadata in a PostgreSQL database cluster composed of 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.

API/RPC servers. Beyond the load balancer we find the API and RPC database processes that run on 6 separate racked servers. API servers receive commands from the user, perform authentication, and translate the commands into RPC calls. In turn, RPC database workers translate these RPC calls into database queries and route queries to the appropriate database shards. API/RPC processes are more numerous than physical machines (normally 8 – 16 processes per physical machine), so that they can migrate among machines for load balancing. Internally, API and RPC servers, the load balancer and the metadata store are connected through a switched 1Gbit Ethernet network.

Data storage. Like other popular Personal Clouds, such as Dropbox or SugarSync, U1 stores 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 latest months of operation, U1 had a $\approx 20,000\$$ monthly bill in storage resources, being the most important Amazon S3 client in Europe.

³Li et al. [5] suggest that U1 may group small files together for upload (i.e. bundling), since they observed high efficiency uploading sets of small files. However, U1 does not bundle small files together. Instead, clients establish a TCP connection with the server that remains open during the session, avoiding the overhead of creating new connections.

With this infrastructure, U1 provided service to 1.29 million users traced in this measurement for one month.

3.4.1 Authentication Service

The authentication service of U1 is shared with other Canonical services within the same datacenter and it is based on OAuth [13]. The first time a user interacts with U1, the desktop client requires him to introduce his credentials (email, password). The API server that handles the authentication request contacts the authentication service to generate a new token for this client. The created token is associated in the authentication service with a new user identifier. The desktop client also stores this token locally in order to avoid exposing user credentials in the future.

In the subsequent connections of that user, the authentication procedure is easier. Basically, the desktop client sends a connection request with the token to be authenticated. The U1 API server responsible for that requests asks the authentication service if the token does exist and has not expired. In the affirmative case, the authentication service retrieves the associated user identifier, and a new session is established. During the session, the token of that client is cached to avoid overloading the authentication service.

The authentication infrastructure consists of 1 database server with hot failover and 2 application servers configured with crossed stacks of Apache/Squid/HAProxy.

3.4.2 Notifications

Clients detect changes in their volumes by comparing their local state with the server side on every connection (generation point). However, if two related clients are online and their changes affect each other (e.g. updates to shares, new shares), API servers notify them directly (push). To this end, API servers resort to the TCP connection that clients establish with U1 in every session.

Internally, the system needs a way of notifying changes to API servers that are relevant to simultaneously connected clients. Concretely, U1 resorts to RabbitMQ (1 server) for communicating events between API servers⁴, which are subscribed in the queue system to send and receive new events to be communicated to clients.

Next, we describe our measurement methodology to create the dataset used in our analysis.

4. DATA COLLECTION

We present a study of the U1 service back-end. In contrast to other Personal Cloud measurements [2, 7, 5], we did not deploy vantage points to analyze the service externally. Instead, we inspected directly the U1 metadata servers to measure the system. This has been done in collaboration with Canonical in the context of the FP7 CloudSpaces⁵ project. Canonical anonymized sensitive information to build the trace (user ids, file names, etc.).

The traces are taken at both *API* and *RPC* server stages. In the former stage we collected important information about the storage workload and user behavior, whereas the second stage provided us with valuable information about the requests' life-cycle and the metadata store performance.

We built the trace capturing a series of service *logfiles*. Each logfile corresponds to the entire activity of a single

⁴If connected clients are handled by the same API process, their notifications are sent immediately, i.e. there is no need for inter-process communication with RabbitMQ.

⁵<http://cloudspaces.eu>

Trace duration	30 days (01/11 - 02/10)
Trace size	758GB
Back-end servers traced	6 servers (all)
Unique user IDs	1, 294, 794
Unique files	137.63M
User sessions	42.5M
Transfer operations	194.3M
Total upload traffic	105TB
Total download traffic	120TB

Table 3: Summary of the trace.

API/RPC process in a machine for a period of time. Each logfile is within itself strictly *sequential and timestamped*. Thus, 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).

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 (`whitecurrant`), followed by the number of the server process (23) and the date. The mapping between services and servers is dynamic within the time frame of analyzed logs, since they can migrate between servers to balance load. In any case, the identifier of the process is unique within a machine. After that is the date the logfile was "cut" (there is one log file per server/service and day).

Database sharding is in the metadata store back-end, so it is behind the point where traces were 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 (one session/connection per desktop client). 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.

Approximately 1% of traces are not analyzed due to failures parsing of the logs.

4.1 Dataset

The trace is the result of merging all the logfiles (758GB of .csv text) of the U1 servers for 30 days (see Table 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 real-world cloud service (users, files, operations). Since we captured file properties such as file size and hash, we can study the storage system in high detail (contents are not disclosed).

Dataset limitations. The dataset only includes events originating from desktop clients. Other sources of user requests (e.g., the web front-end, mobile clients) are handled by different software stacks that were not logged. Also, a small number of apparently malfunctioning clients seems to continuously upload files hundreds of times —these artifacts have been removed for this analysis. Finally, we detected that sharing among users is limited.

5. STORAGE WORKLOAD

First, we quantify the storage workload supported by U1 for one month. Moreover, we pay special attention to the behavior of files in the system, to infer potential improvements. We also unveil attacks perpetrated to the U1 service.

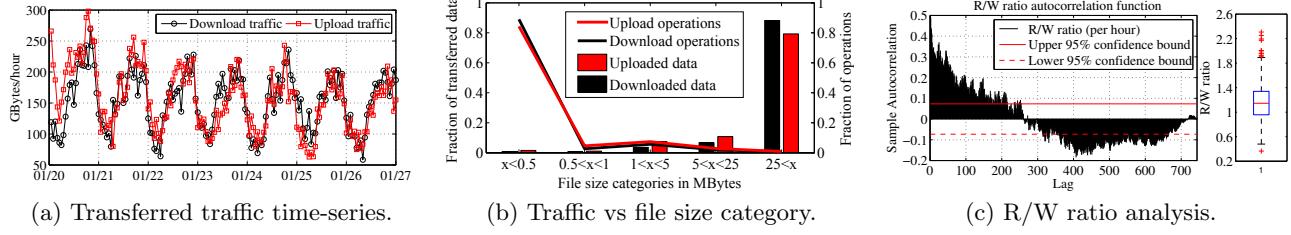


Figure 2: Macroscopic storage workload metrics of U1.

5.1 Macroscopic Daily Usage

Storage traffic and operations. Fig. 2(a) provides a time-series view of the upload/download traffic of U1 for one week. We observe in Fig. 2(a) that U1 exhibits important *daily patterns*. To wit, the volume of uploaded GBytes per hour can be up to 10x higher in the central day hours compared to the nights. This observation is aligned with previous works, that detected time-based variability in both the usage and performance of Personal Cloud services [2, 7]. This effect is probably related to the working habits of users, since U1 desktop clients are by default initiated automatically when users turn on their machines.

Another aspect to explore is the relationship between file size and its impact in terms of upload/download traffic. To do so, in Fig. 2(b), we depict in relative terms the fraction of transferred data and storage operations for distinct file sizes. As can be observed, a very small amount of large files ($> 25\text{MBytes}$) consumes 79.3% and 88.2% of upload and download traffic, respectively. Conversely, 84.3% and 89.0% of upload and download operations are related to small files ($< 0.5\text{MBytes}$). As reported in other domains [14, 15, 16], we conclude that in U1 the workload in terms of *storage operations* is dominated by small files, whereas a small number of large files generate most of the *network traffic*.

For uploads, we found that 10.05% of total upload operations are *updates*, that is, an upload of an existing file that has distinct hash/size. However, in terms of traffic, file updates represent 18.47% of the U1 upload traffic. This can be partly explained by the lack of delta updates in the U1 client and the heavy file-editing usage that many users exhibited (e.g., code developers). Particularly for media files, U1 engineers found that applications that modify the metadata of files (e.g., tagging .mp3 songs) induced high upload traffic since the U1 client uploads again files upon metadata changes, as they are interpreted as regular updates.

To summarize, Personal Clouds tend to exhibit daily traffic patterns, and most of this traffic is caused by a small number of large files. Moreover, desktop clients should efficiently handle file updates to minimize traffic overhead.

R/W ratio. The read/write (R/W) ratio represents the relationship between the downloaded and uploaded data in the system for a certain period of time. Here we examine the variability of the R/W ratio in U1 (1-hour bins). The boxplot in Fig. 2(c) shows that the R/W ratio *variability can be important*, exhibiting differences of 8x within the same day. Moreover, the median (1.14) and mean (1.17) values of the R/W ratio distribution point out that the U1 workload is *slightly read-dominated*, but not as much as it has been observed in Dropbox [2]. One of the reasons for this is that the sharing activity in U1 was much lower than for Dropbox.

We also want to explore if the R/W ratios present patterns or dependencies along time due to the working habits of users. To verify whether R/W ratios are independent along time, we calculated the autocorrelation function (ACF) for

each 1-hour sample (see Fig. 2(c)). To interpret Fig. 2(c), if R/W ratios are completely uncorrelated, the sample ACF is approximately normally distributed with mean 0 and variance $1/N$, where N is the number of samples. The 95% confidence limits for ACF can then be approximated to $\pm 2/\sqrt{N}$.

As shown in Fig. 2(c), R/W ratios are not independent, since most lags are outside 95% confidence intervals, which indicates long-term correlation with alternating positive and negative ACF trends. This evidences that the R/W ratios of U1 workload are not random and follow a pattern also guided by the working habits of users.

Concretely, averaging R/W ratios for the same hour along the whole trace, we found that from 6am to 3pm the R/W ratio shows a linear decay. This means that users download more content when they start the U1 client, whereas uploads are more frequent during the common working hours. For evenings and nights we found no clear R/W ratio trends.

We conclude that different Personal Clouds may exhibit disparate R/W ratios, mainly depending on the purpose and strengths of the service (e.g., sharing, content distribution). Moreover, R/W ratios exhibit patterns along time, which can be predicted in the server-side to optimize the service.

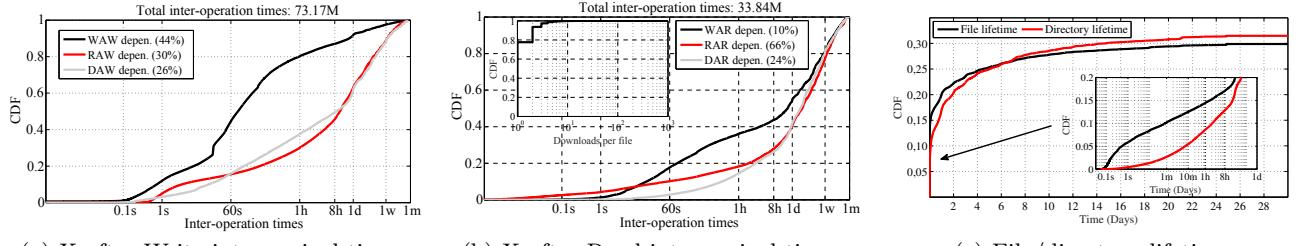
5.2 File-based Workload Analysis

File operation dependencies. Essentially, in U1 a file can be *downloaded (or read)* and *uploaded (or written)* multiple times, until it is eventually *deleted*. Next, we aim at inspecting the dependencies among file operations [17, 18], which can be *RAW* (Read-after-Write), *WAW* (Write-after-Write) or *DAW* (Delete-after-Write). Analogously, we have *WAR*, *RAR* and *DAR* for operations executed after a read.

First, we inspect file operations that occur after a write (Fig. 3(a)). We see that *WAW* dependencies are the most common ones (30.1% of 170.01M in total). This can be due to the fact that users *regularly update synchronized files*, such as documents of code files. This result is consistent with the results in [17] for personal workstations where block updates are common, but differs from other organizational storage systems in which files are almost immutable [18]. Furthermore, the 80% of *WAW* times are shorter than 1 hour, which seems reasonable since users may update a single text-like file various times within a short time lapse.

In this sense, Fig. 3(a) shows that *RAW* dependencies are also relevant. Two events can lead to this situation: (i) the system synchronizes a file to another device right after its creation, and (ii) downloads that occur after every file update. For the latter case, reads after successive writes can be optimized with sync deferment to reduce network overhead caused by synchronizing intermediate versions to multiple devices [5]. This has not been implemented in U1.

Second, we inspect the behavior of X-after-Read dependencies (Fig. 3(b)). As a consequence of active update patterns (i.e., write-to-write) and the absence of sync deferment, we see in Fig. 3(b) that *WAR* transitions also occur



(a) X-after-Write inter-arrival times.

(b) X-after-Read inter-arrival times.

(c) File/directory lifetime.

Figure 3: Usage and behavior of files in U1.

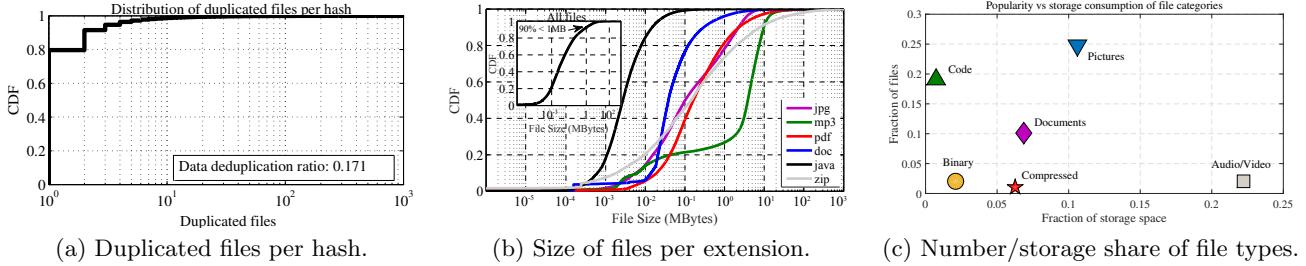


Figure 4: Characterization of files in U1.

within reduced time frames compared to other transitions. Anyway, this dependency is the least popular one yielding that files that are *read tend not to be updated again*.

In Fig. 3(b), 40% of RAR times fall within 1 day. RAR times are shorter than the ones reported in [18], which can motivate the introduction of *caching mechanisms* in the U1 back-end. Caching seems specially interesting observing the inner plot of Fig. 3(b) that reveals a long tail in the distributions of reads per file. This means that a small fraction of files is very popular and may be effectively cached.

By inspecting the Delete-after-X dependencies, we detected that around 12.5M files in U1 were *completely unused* for more than 1 day before their deletion (9.1% of all files). This simple observation on dying files evidences that *warm and/or cold data exists* in a Personal Cloud, which may motivate the involvement of warm/cold data systems in these services (e.g., Amazon Glacier, f4 [19]). To efficiently managing warm files in these services is object of current work.

Node lifetime. Now we focus on the lifetime of user files and directories (i.e., nodes). As shown in Fig. 3(c), 28.9% of the new files and 31.5% of the recently created directories are deleted within one month. We also note that the lifetime distributions of *files and directories are very similar*, which can be explained by the fact that deleting a directory in U1 triggers the deletion of all the files it contains.

This figure also unveils that a large fraction of nodes are deleted within *a few hours after their creation*, especially for files. Concretely, users delete 17.1% of files and 12.9% of directories within 8 hours after their creation time.

All in all, in U1 files exhibit similar lifetimes than files in local file systems. For instance, Agrawal et al. in [15] analyzed the lifetimes of files in corporative desktop computers for five years. They reported that around 20% to 30% of files (depending on the year) in desktop computers present a lifetime of one month, which agrees with our observations. This suggests that *users behave similarly deleting files either in synchronized or local folders*.

5.3 File Deduplication, Sizes and Types

File-based deduplication. The deduplication ratio (dr) is a metric to quantify the proportion of duplicated data. It

takes real values in the interval $[0, 1]$, with 0 signaling no file deduplication at all, and 1 meaning full deduplication. It is expressed as $dr = 1 - (D_{unique}/D_{total})$, where D_{unique} is the amount of unique data, and D_{total} is equal to the total storage consumption.

We detected a dr of 0.171, meaning that 17% of files' data in the trace can be deduplicated, which is similar (18%) to that given by the recent work of Li et al. [5]. This suggests that file-based cross-user deduplication could be a practical approach to reduce storage costs in U1.

Moreover, Fig. 4(a) demonstrates that the distribution of file objects w.r.t unique contents exhibits a long tail. This means that a small number of files accounts for a very large number of duplicates (e.g., popular songs), whereas 80% files present no duplicates. Hence, files with many duplicates represent a *hot spot* for the deduplication system, since a large number of logical links point to a single content.

File size distribution. The inner plot of Fig. 4(b) illustrates the file size distribution of transferred files in the system. At first glance, we realize that the *vast majority of files are small* [14, 15, 16]. 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 store, which has not been designed for storing very small files [8].

In this sense, Fig. 4(b) shows the file size distribution of the most popular file extensions in U1. Non-surprisingly, the distributions are very disparate, which can be used to model realistic workloads in Personal Cloud benchmarks [3]. It is worth noting that in general, incompressible files like zipped files or compressed media are larger than compressible files (docs, code). This observation indicates that compressing files *does not provide much benefits* in many cases.

File types: number vs storage space. We classified files belonging to the 55 most popular file extensions into 7 categories: Pics (.jpg, .png, .gif, etc.), Code (.php, .c, .js, etc.), Docs (.pdf, .txt, .doc, etc.), Audio/Video (.mp3, .wav, .ogg, etc.), Application/Binary (.o, .msf, .jar, etc.) and Compressed (.gz, .zip, etc.). Then, for each cat-

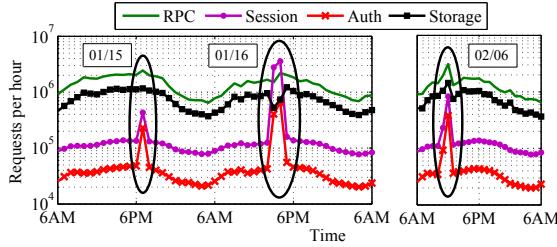


Figure 5: DDoS attacks detected in our trace.

egory, we calculated the ratio of the number of files to the total in the system. We did the same for the storage space. This captures the relative importance of each content type.

Fig. 4(c) reveals that Audio/Video category is one of the most relevant types of files regarding the share of consumed storage, despite the fraction of files belonging to this class is low. The reason is that U1 users stored .mp3 files, which are usually larger than other popular text-based file types.

Further, the Code category contains the highest fraction of files, indicating that many U1 users are code developers who frequently update such files, despite the storage space required for this category is minimal. Docs are also popular (10.1%), subject to updates and hold 6.9% of the storage share. Since the U1 desktop client lacks delta updates and deferred sync, such frequent updates pose a high stress for desktop clients and induce significant network overhead [5].

5.4 DDoS and Abuse of Personal Clouds

A Distributed Denial of Service (DDoS) can be defined as the attempt to disrupt the legitimate use of a service [20]. Normally, a DDoS attack is normally accompanied by some form of fraudulent resource consumption in the victim’s side.

Surprisingly, we found that DDoS attacks to U1 are *more frequent* than one can reasonably expect. Specifically, we found evidence of three such attacks in our traces (January 15, 16 and February 6)⁶. These DDoS attacks had as objective to *share illegal content* through the U1 infrastructure.

As visible in Fig. 5, all the attacks resulted in a dramatic increase of the number of session and authentication requests per hour —both events related to the management of user sessions. Actually, the authentication activity under attack was 5 to 15 times higher than usual, which directly impacts the Canonical’s *authentication subsystem*.

The situation for API servers was even worse: during the second attack (01/16) API servers received an activity 245x higher than usual, whereas during the first (01/15) and last (02/06) attacks the activity was 4.6x and 6.7x higher than normal, respectively. Therefore, the most affected components were the API servers, as they serviced both *session and storage operations*.

We found that these attacks consisted on sharing a *single user id* and its credentials to distribute content across thousands of desktop clients. The nature of this attack is similar to the *storage leeching problem* reported in [12], which consists of exploiting the freemium business model of Personal Clouds to illicitly consume bandwidth and storage resources.

Also, the reaction to these attacks was not automatic. U1 engineers manually handled DDoS by means of deleting fraudulent users and the content to be shared. This can be easily seen on the storage activity for the second and

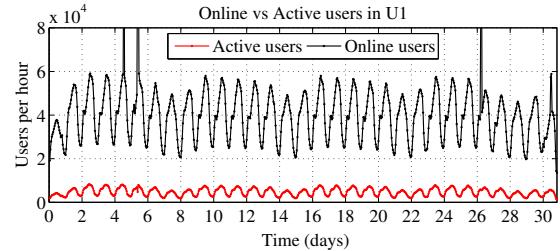


Figure 6: Online vs active users per hour.

third attack, which decays within one hour after engineers detected and responded to the attack.

These observations confirm that Personal Clouds are a suitable target for attack as other Internet systems, and that these situations are indeed common. We believe that further research is needed to build and apply secure storage protocols to these systems, as well as new countermeasures to automatically react to this kind of threats.

6. UNDERSTANDING 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 Distinguishing Online from Active Users

Online and active users. We consider a user as *online* if his desktop client exhibits any form of interaction with the server. This includes automatic client requests involved in maintenance or notification tasks, for which the user is not responsible for. Moreover, we consider a user as *active* if he performs data management operations on his volumes, such as uploading a file or creating a new directory.

Fig. 6 offers a time-series view of the number of online and active users in the system per hour. Clearly, *online users are more numerous than active users*: The percentage of active users ranges from 3.49% to 16.25% at any moment in the trace. This observation reveals that the actual storage workload that U1 supports is light compared to the potential usage of its user population, and gives a sense on the scale and costs of these services with respect to their popularity.

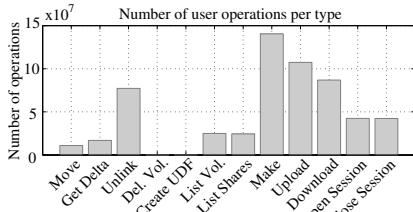
Frequency of user operations. Here we examine how frequent the protocol operations are in order to identify the hottest ones. Fig. 7(a) depicts the absolute number of each operation type. As shown in this figure, the most frequent operations correspond to *data management operations*, and in particular, those operations that relate to the download, upload and deletion of files.

Given that active users are a minority, it proves that the U1 protocol does not impose high overhead to the server-side, since the operations that users issue to manage their sessions and are typically part of the session start up (e.g., *ListVolumes*) are not dominant. And consequently, the major part of the processing burden comes from active users as desired. This is essentially explained by the fact that the U1 desktop client does not need to regularly poll the server during idle times, thereby limiting the number of requests not linked to data management.

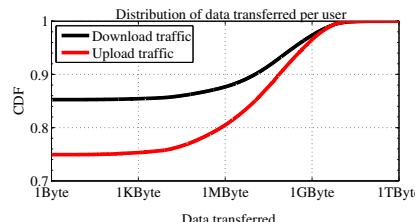
As we will see in § 7, the frequency of API operations will have an immediate impact on the back-end performance.

Traffic distribution across users. Now, we turn our attention to the distribution of consumed traffic across users. In Fig. 7(b) we observe an interesting fact: in one month, only 14% of users downloaded data from U1, while uploads

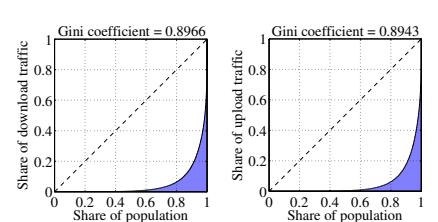
⁶Our interviews with Canonical engineers confirmed that these activity spikes correspond to DDoS attacks, instead of a software release or any other legitimate event.



(a) Total number of requests per type.



(b) CDF of traffic across users.



(c) Lorenz curve of traffic vs active users.

Figure 7: User requests and consumed traffic in U1 for one month.

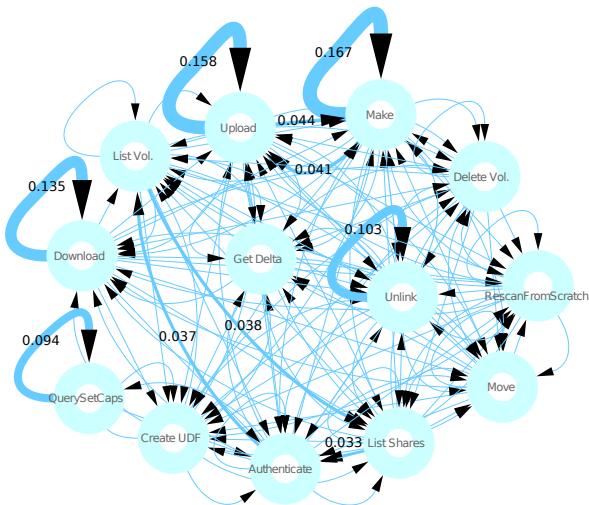


Figure 8: Desktop client transition graph through API operations. Global transition probabilities are provided for main edges.

represented 25%. This indicates that a minority of users are responsible for the storage workload of U1.

To better understand this, we measure how (un)equal the traffic distribution across active users is. To do so, we resort to the Lorenz curve and the Gini coefficient⁷ as indicators of inequality. The Gini coefficient varies between 0, which reflects complete equality, and 1, which indicates complete inequality (i.e., only one user consumes all the traffic). The Lorenz curve plots the proportion of the total income of the population (y axis) that is cumulatively earned by the bottom $x\%$ of the population. The line at 45 degrees thus represents perfect equality of incomes.

Fig. 7(c) reports that the consumed traffic across active users is very unequal. That is, the Lorenz curve is very far from the diagonal line and the Gini coefficient is close to 1. The reason for this inequality is clear: 1% of active users account for the 65.6% of the total traffic (147.52TB). Providers may benefit from this fact by identifying and treating these users more efficiently.

Types of user activity. To study the activity of users, we used the same user classification than Drago et al. in [2]. So we distinguished among *occasional*, *download/upload only* and *heavy* users. A user is *occasional* if he transfers less than 10KB of data. Users that exhibit more than three orders of magnitude of difference between upload and download (e.g., 1GB versus 1MB) traffic are classified as either *download-only* or *upload-only*. The rest of users are in the *heavy* group.

Given that, we found that 85.82% of *all* users are *occasional* (mainly online users), 7.22% *upload-only*, 2.34%

download-only and 4.62% are *heavy* users. Our results clearly differ from the ones reported in [2], where users are 30% *occasional*, 7% *upload-only*, 26% *download-only* and 37% *heavy*. This may be explained by two reasons: (i) the usage of Dropbox is more extensive than the usage of U1, and (ii) users in a university campus are more active and share more files than other user types captured in our trace.

6.2 Characterizing User Interactions

User-centric request graph. To analyze how users interact with U1, Fig. 8 shows the sequence of operations that desktop clients issue to the server in form of a graph. Nodes represent the different protocol operations executed. And edges describe the transitions from one operation to another. The width of edges denotes the global frequency of a given transition. Note that this graph is user-centric, as it aggregates the different sequence of commands that every user executes, not the sequence of operations as they arrive to the metadata service.

Interestingly, we found that the *repetition of certain operations* becomes really frequent across clients. For instance, it is highly probable that when a client transfers a file, the next operation that he will issue is also another transfer —either upload or download. This phenomenon can be partially explained by the fact that many times users synchronize data at *directory granularity*, which involves repeating several data management operations in cascade. File editing can be also a source of recurrent transfer operations. This behavior can be exploited by predictive data management techniques in the server side (e.g., download prefetching).

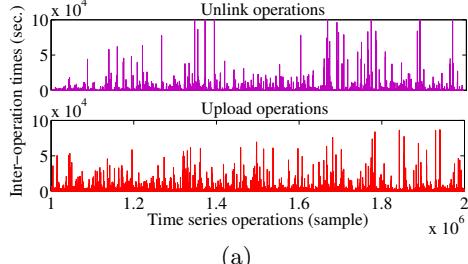
Other sequences of operations are also highlighted in the graph. For instance, once a user is authenticated, he usually performs a *ListVolumes* and *ListShares* operations. This is a regular initialization flow for desktop clients. We also observe that *Make* and *Upload* operations are quite mixed, evidencing that for uploading a file the client first needs to create the metadata entry for this file in U1.

Burstiness in user operations. Next, we analyze inter-arrival times between consecutive operations of the same user. We want to verify whether inter-operation times are Poisson or not, which may have important implications to the back-end performance. To this end, we followed the same methodology proposed in [21, 22], and obtained a time-series view of *Unlink* and *Upload* inter-operation times and their approximation to a power-law distribution in Fig. 9.

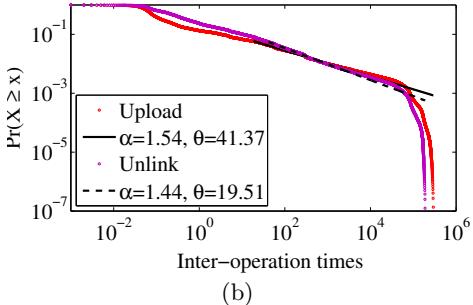
Fig. 9(a) exhibits large spikes for both *Unlink* and *Upload* operations, corresponding to *very long inter-operation times*. This is far from an exponential distribution, where long inter-operation times are negligible. This shows that the interactions of users with U1 are not Poisson [21].

Now, we study if the *Unlink* and *Upload* inter-operation times exhibit *high variance*, which indicates *burstiness*. In all cases, while not strictly linear, these distributions show a downward trend over almost six orders of magnitude. This

⁷http://en.wikipedia.org/wiki/Gini_coefficient



(a)



(b)

Figure 9: Time-series user operations inter-arrival times and their approximation to a power-law.

suggests that high variance of user inter-arrival operations is present in time scales ranging from seconds to several hours. Hence, users issue requests in a *bursty non-Poisson way*: during a short period a user sends several operations in quick succession, followed by long periods of inactivity. A possible explanation to this is that users manage data at the *directory granularity*, thereby triggering multiples operations to keep the files inside each directory in sync.

Nevertheless, we cannot confirm the hypothesis that these distributions are heavy-tailed. Clearly, Fig. 9(b) visually confirms that the empirical distributions of user **Unlink** and **Upload** inter-arrivals can be only approximated with $P(x) \approx x^{-\alpha}, \forall x > \theta, 1 < \alpha < 2$, for a central region of the domain.

We also found that metadata operations follow more closely a power-law distribution than data operations. The reason is that the behavior of metadata inter-operation times are not affected by the actual data transfers.

In conclusion, we can see that user operations are bursty, which has strong implications to the operation of the back-end servers (§ 7).

6.3 Inspecting User Volumes

Volume contents. Fig. 10 illustrates the relationship between files and directories within user volumes. As usual, files are much more numerous than directories. And we have that over 60% of volumes have been associated with at least one file. For directories, this percentage is only of 32%, but there is a strong correlation between the number of files and directories within a volume: Pearson correlation coefficient is 0.998. What is relevant is, however, that a small fraction of volumes is heavily loaded: 5% of user volumes contain more than 1,000 files.

Shared and user-defined volumes. At this point, we study the distribution of user-defined/shared volumes across users. As pointed out by Canonical engineers, sharing is not a popular feature of U1. Fig. 11 shows that only 1.8% of users exhibits at least one shared volume. On the contrary, we observe that user-defined volumes are much more popular; we detected user-defined volumes in 58% of users —the

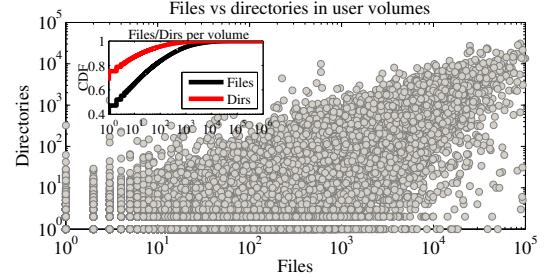


Figure 10: Files and directories per volume.

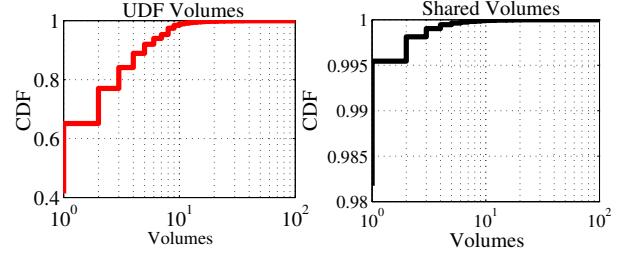


Figure 11: Distribution of shared/user-defined volumes across users.

rest of users only use the root volume. This shows that the majority of users have some degree of expertise using U1.

Overall, these observations reveal that U1 was used more as a storage service rather than for collaborative work.

7. METADATA BACK-END ANALYSIS

In this section, we focus on the interactions of RPC servers against the metadata store. We also quantify the role of the Canonical authentication service in U1.

7.1 Performance of Metadata Operations

Here we analyze the performance of RPC operations that involve contacting the metadata store.

Fig. 12 illustrates the distribution of service times of the different RPC operations. As shown in the figure, all RPCs exhibit *long tails of service time distributions*: from 7% to 22% of RPC service times are very far from the median value. This issue can be caused by several factors, ranging from interference of background processes to CPU power saving mechanisms, as recently argued by Li et al. in [9].

Also useful is to understand the relationship between the service time and the frequency of each RPC operation. Fig. 13 presents a scatter plot relating RPC median service times with their frequency, depending upon whether RPCs are of type *read*, *write/update/delete* or *cascade*, i.e., whether other operations are involved. This figure confirms that the type of an RPC strongly determines its performance. First, *cascade* operations (*delete_volume* and *get_from_scratch*) are the slowest type of RPC —more than one order of magnitude slower compared to the fastest operation. Fortunately, they are relatively infrequent. Conversely, *read* RPCs, such as *list_volumes*, are the fastest ones. Basically, this is because *read* RPCs can exploit lockless and parallel access to the pairs of servers that form database shards.

Write/update/delete operations (e.g. *make_content*, or *make_file*) are slower than most *read* operations, but exhibiting comparable frequencies. This may represent a performance barrier for the metadata store in scenarios where users massively update metadata in their volumes or files.

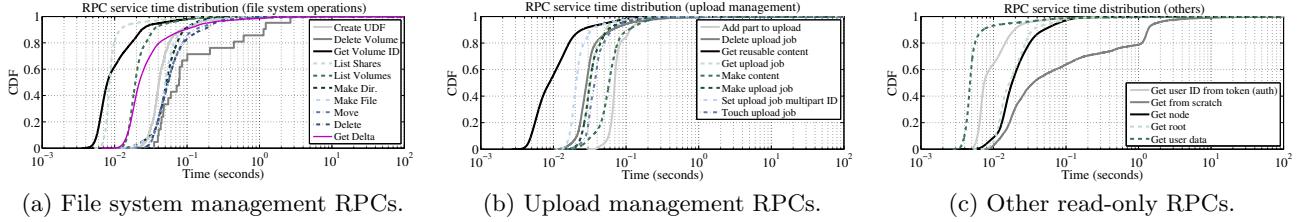


Figure 12: Distribution of RPC service times accessing to the metadata store.

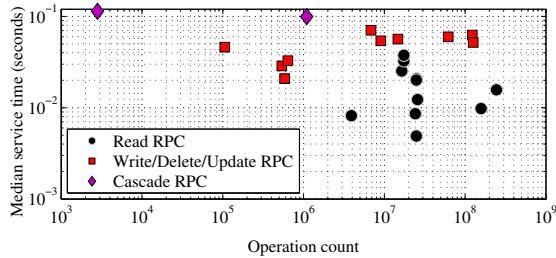


Figure 13: We classified all the U1 RPC calls into 3 categories, and every point in the plot represents a single RPC call. We show the median service time vs frequency of each RPC (1 month).

7.2 Load Balancing in U1 Back-end

We are interested in analyzing the internal load balancing of both API servers and shards in the metadata store. In the former case, we grouped the processed API operations by physical machine. In the latter, we distributed the RPC calls contacting the metadata store across 10 shards based on the user id, as U1 actually does. Results appear in Fig. 14, where bars are mean load values and error lines represent the standard deviation of load values across API servers and shards per hour and minute, respectively.

Fig. 14 shows that server load presents a *high variance across servers*, which is symptom of bad load balancing. This effect is present irrespective of the hour of the day and is more accentuated for the metadata store, for which the time granularity used is smaller. Thus, this phenomenon is visible in short or moderate periods of time. In the long term, the load balancing is adequate; the standard deviation across shards is only of 4.9% when the whole trace is taken.

Three particularities should be understood to explain the poor load balancing. First, user load is *uneven*, i.e., a small fraction of users is very active whereas most of them present low activity. Second, the cost of operations is *asymmetric*; for instance, there are metadata operations whose median service time is 10x higher than others. Third, users display a *bursty* behavior when interacting with the servers; for instance, they can synchronize an entire folder. So, operations arrive in a correlated manner.

We conclude that the load balancing in the U1 back-end can be significantly improved, which is object of future work.

7.3 Authentication Activity & User Sessions

Time-series analysis. Users accessing the U1 service should be authenticated prior to the establishment of a new session. To this end, U1 API servers communicate with a separate and shared authentication service of Canonical.

Fig. 15 depicts a time-series view of the session management load that API servers support to create and destroy sessions, along with the corresponding activity of the authentication subsystem. In this figure, we clearly observe that the authentication and session management activity is

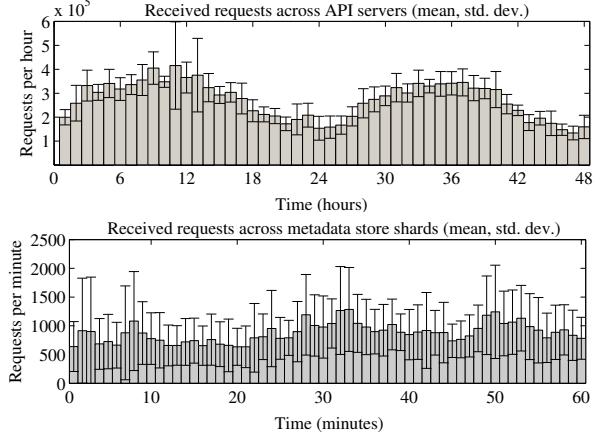


Figure 14: Load balancing of U1 API servers and metadata store shards.

closely related to the habits of users. In fact, daily patterns are evident. The authentication activity is 50% to 60% higher in the central hours of the day than during the night periods. This observation is also valid for week periods: on average, the maximum number of authentication requests is 15% higher on Mondays than on weekends. Moreover, we found that 2.76% of user authentication requests from API servers to the authentication service fail.

Session length. Upon a successful authentication process, a user's desktop client creates a new U1 session.

U1 sessions exhibit a similar behavior to Dropbox *home* users in [2] (Fig. 15). Concretely, 97% of sessions are shorter than 8 hours, which suggests a strong correlation with user working habits. Moreover, we also found that U1 exhibits a high fraction of very short-lived sessions (i.e. 32% shorter than 1s.). This is probably due to the operation of NAT and firewalls that normally mediate between clients and servers, which might be forcing the creation of new sessions by closing TCP connections unexpectedly [2, 23]. Overall, Fig. 15 suggests that *domestic users are more representative* than other specific profiles, such as university communities, for describing the connection habits of an entire Personal Cloud user population.

We are also interested in understanding the data management activity related to U1 sessions. To this end, we differentiate sessions that exhibited any type of data management operation (e.g., upload) during their lifetime (*active sessions*) from sessions that do not (*cold sessions*).

First, we observed that the majority of U1 sessions (and, therefore, TCP connections) do not involve any type of data management. That is, only 5.57% of connections in U1 are active (2.37M out of 42.5M), which, in turn, tend to be much longer than *cold* ones. From a back-end perspective, the unintended consequence is that a fraction of server resources is *wasted keeping alive TCP connections of cold sessions*.

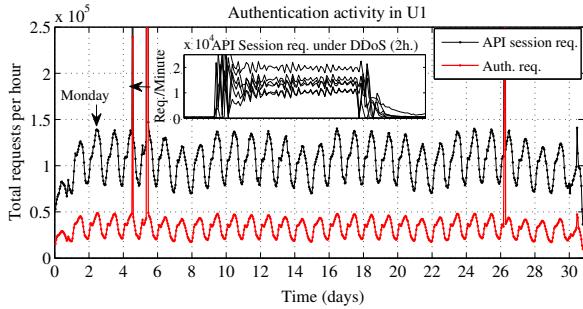


Figure 15: API session management operations and authentication service requests. The inner plot shows API session requests under a DDoS.

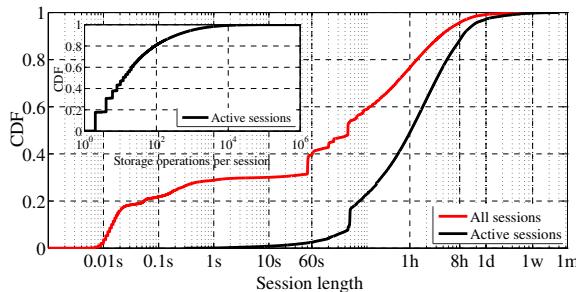


Figure 16: Distribution of session lengths and storage operations per session.

Moreover, similarly to the distribution of user activity, the inner plot of Fig. 15 shows that 80% of active sessions exhibited at most 92 storage operations, whereas the remaining 20% accounted for 96.7% of all data management operations. Therefore, the active sessions much more active than others.

A provider may benefit from these observations to optimize session management. That is, depending on a user's activity, the provider may wisely decide if a desktop client works in a *pull* (cold sessions) or *push* (active sessions) fashion to limit the number of open TCP connections [24].

8. RELATED WORK

The performance evaluation of cloud storage services is an interesting topic inspiring several recent papers. Hill et al. in [25] provide a quantitative analysis of the performance of the Windows Azure Platform, including storage. Palankar et al. in [26] perform an extensive measurement against Amazon S3 to elucidate whether cloud storage is suitable for scientific Grids. Similarly, [27] presents a performance analysis of the Amazon Web Services. Liu et. al. [18] inspected in depth the workload patterns of users in the context of a storage system within a university campus. This work concentrates on macroscopic storage workload metrics and type of requests, as well as the differences in access patterns of personal and shared folders. Unfortunately, these papers provide no insights into Personal Clouds.

Perhaps surprisingly, despite their commercial popularity, only few research works have turned attention to analyze the performance of Personal Cloud storage services [2, 3, 5, 6, 7, 12, 28, 29]. We classify them into two categories:

Active measurements. The first analysis of Personal Cloud storage services we are aware is due to Hu et al. [6] that compared Dropbox, Mozy, Carbonite and CrashPlan storage services. However, their analysis was rather incomplete; the metrics provided in [6] are only backup/restore

times depending on several types of backup contents. They also discuss potential privacy and security issues comparing these vendors. The work of Hu et al. [6] was complemented by Gracia-Tinedo et al. [7] that extensively studied the REST interfaces provided by three big players in the Personal Cloud arena, analyzing important aspects of their QoS and potential exploitability [12].

Recently, Drago et al. [3] presented a complete framework to benchmark Personal Cloud desktop clients. One of their valuable contributions is to set a benchmarking framework addressed to compare the different data reduction techniques implemented in desktop clients (e.g., file bundling).

Passive measurements. Drago et al. [2] presented an external measurement of Dropbox in both a university campus and residential networks. They analyzed and characterized the traffic generated by users, as well as the workflow and architecture of the service. [29] extended that measurement by modeling the behavior of Dropbox users. Similarly, Mager et al. [28] uncovered the architecture and data management of Wuala, a peer-assisted Personal Cloud.

Unfortunately, these works do not provide insights on the provider's metadata back-end, since this is not possible from external vantage points. Moreover, [2, 29] study the storage workload and user behavior on specific communities (e.g., university campus) that may lead to false generalizations. In contrast, we analyze the entire population of U1.

Furthermore, Li et al. [5] analyzed a group of Personal Cloud users in both university and corporate environments. Combined with numerous experiments, they studied the efficiency of file sync protocols, as well as the interplay between data reduction techniques integrated in desktop clients and users' workload (update frequency, file compressibility).

Key differences with prior work. The main difference with previous works is that we study in details the *metadata back-end servers* of a Personal Cloud, instead of simply measuring it from outside. The unique aspect of our work is that most of our insights could have been obtained only by taking a perspective from within a data center; this goes, for example, for the internal infrastructure of the service, or the performance of the metadata store, to name a few.

Also, apart from guiding simulations and experiments, we believe that the present analysis will help researchers to optimize several aspects of these services, such as file synchronization [4, 10] and security [30], among others.

9. DISCUSSION AND CONCLUSIONS

In this paper, we focus on understanding the nature of Personal Cloud services by presenting the internal structure and measurement study of UbuntuOne (U1). The objectives of our work are threefold: (i) to unveil the internal operation and infrastructure of a real-world provider, (ii) to reconfirm, expand and contribute observations on these systems to generalize their characteristics, and (iii) to propose potential improvements for these systems.

This work unveils several aspects that U1 *shares* with other large-scale Personal Clouds. For instance, U1 presents clear similarities with Dropbox regarding the way of decoupling data and metadata of users, which seems to be a standard design for these systems [10]. Also, we found characteristics in the U1 workload that reconfirm observations of prior works [2, 5] regarding the *relevance of file updates*, the *effectiveness of deduplication* or the execution of user operations in *long sequences*, among other aspects. Therefore, our analysis and the resulting dataset will enable researchers to get closer to the nature of a real-world Personal Cloud.

Thanks to the scale and back-end perspective of our study, we expanded and contributed insights on these services. That is, we observed that the distribution of activity across users in U1 is even *more skewed* than in Dropbox [2] or that the behavior of domestic users dominate session lengths in U1 compared to other user types (e.g., university). Among the novelties of this work, we modeled the *burstiness of user operations*, we analyzed the *behavior of files* in U1, we provided evidences of *DDoS attacks* to this service, and we illustrated the performance of the U1 *metadata back-end*.

An orthogonal conclusion that we extract from our study is that understanding *the behavior of users is essential* to adapt the system to its actual demands and reduce costs. In the following, we relate some of our insights to the running costs of U1 as well as potential optimizations, which may be of independent interest for other large-scale systems:

Optimizing storage matters. A key problem to the survival of U1 was the growing costs of outsourcing data storage [31], which is directly related to the data management techniques integrated in the system. For instance, the fact that file updates were responsible for 18.5% of upload traffic in U1, mainly due to the lack of delta updates in the desktop client, gives an idea of the margin of improvement (§ 5.1). Actually, we confirmed that a simple optimization like file-based deduplication could readily save 17% of the storage costs. This calls to further research and the application of advanced data reduction techniques, both at the client and server sides.

Take care of user activity. This observation is actually very important, as we found that 1% of U1 users generated 65% of traffic (§ 6.1), showing a weak form of the Pareto Principle. That is, a very small fraction of the users represented most of the OPEX for U1. A natural response may be to limit the activity of free accounts, or at least to treat active users in a more cost effective way. For instance, distributed caching systems like Memcached, data prefetching techniques, and advanced sync deferment techniques [5] could easily cut the operational costs down. On the other hand, U1 may benefit from cold/warm storage services (e.g., Amazon Glacier, f4 [19]) to limit the costs related to most inactive users.

Security is a big concern. Another source of expense for a Personal Cloud is related to its exploitation by malicious parties. In fact, we found that DDoS attacks aimed at sharing illegal content via U1 are indeed frequent (§ 5.4). The risk that these attacks represent to U1 is in contrast to the limited automation of its countermeasures. We believe that further research is needed to integrate secure storage protocols and automated countermeasures for Personal Clouds. In fact, understanding the common behavior of users in a Personal Cloud (e.g., storage, content distribution) may provide clues to automatically detect anomalous activities.

Acknowledgment

This work has been funded by the Spanish Ministry of Science and Innovation through projects DELFIN (TIN-2010-20140-C03-03) and “Servicios Cloud y Redes Comunitarias” (TIN-2013-47245-C2-2-R) as well as by the European Union through projects FP7 CloudSpaces (FP7-317555) and H2020 IOStack (H2020-644182). We also thank Jordi Pujol-Ahulló for his feedback in the latest versions of this paper.

10. REFERENCES

- [1] F. Research, “The personal cloud: Transforming personal computing, mobile, and web markets.” <http://www.forrester.com>, 2011.
- [2] I. Drago, M. Mellia, M. M Munafò, A. Sperotto, R. Sadre, and A. Pras, “Inside dropbox: understanding personal cloud storage services,” in *ACM IMC’12*, 2012, pp. 481–494.
- [3] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, “Benchmarking personal cloud storage,” in *ACM IMC’13*, 2013, pp. 205–212.
- [4] 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.
- [5] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z.-L. Zhang, “Towards network-level efficiency for cloud storage services,” in *ACM IMC’14*, 2014.
- [6] 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.
- [7] R. Gracia-Tinedo, M. Sánchez-Artigas, A. Moreno-Martínez, C. Cotes, and P. García-López, “Actively measuring personal cloud storage,” in *IEEE CLOUD’13*, 2013, pp. 301–308.
- [8] R. Sears, C. Van Ingen, and J. Gray, “To blob or not to blob: Large object storage in a database or a filesystem?” Microsoft Research, Tech. Rep., 2007.
- [9] J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble, “Tales of the tail: Hardware, os, and application-level sources of tail latency,” in *ACM SoCC’14*, 2014.
- [10] 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, pp. 49–60.
- [11] “FP7 cloudsaces EU project,” <http://cloudspaces.eu>.
- [12] R. Gracia-Tinedo, M. Sánchez-Artigas, and P. García-López, “Cloud-as-a-gift: Effectively exploiting personal cloud free accounts via REST APIs,” in *IEEE CLOUD’13*, 2013, pp. 621–628.
- [13] E. Hammer-Lahav, “The OAuth 1.0 Protocol,” <http://tools.ietf.org/html/rfc5849>, 2010.
- [14] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, “Measurements of a distributed file system,” in *ACM SIGOPS Operating Systems Review*, vol. 25, no. 5, 1991, pp. 198–212.
- [15] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, “A five-year study of file-system metadata,” *ACM Transactions on Storage*, vol. 3, no. 3, p. 9, 2007.
- [16] 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. 5–2.
- [17] W. Hsu and A. Smith, “Characteristics of i/o traffic in personal computer and server workloads,” *IBM Systems Journal*, vol. 42, no. 2, pp. 347–372, 2003.
- [18] S. Liu, X. Huang, H. Fu, and G. Yang, “Understanding data characteristics and access patterns in a cloud storage system,” in *IEEE/ACM CCGrid’13*, 2013, pp. 327–334.
- [19] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, and S. Kumar, “f4: Facebook’s warm blob storage system,” in *USENIX OSDI’14*, 2014, pp. 383–398.

<code>dal.add_part_to_uploadjob</code>	Continues a multipart upload by adding a new chunk.
<code>dal.delete_uploadjob</code>	Garbage-collects the server-side state for a multipart upload, either because of commit or cancellation.
<code>dal.get_reusable_content</code>	Check whether the server already has the content that is being uploaded.
<code>dal.get_uploadjob</code>	Get the server-side state for a multipart upload.
<code>dal.make_content</code>	Make a file entry in the metadata store (the equivalent of an inode).
<code>dal.make_uploadjob</code>	Set up the server-side structure for multipart upload.
<code>dal.set_uploadjob_multipart_id</code>	Set the requested Amazon S3 multipart upload id to the <code>uploadjob</code> .
<code>dal.touch_uploadjob</code>	Check if the client has canceled the multipart upload (garbage collection after a week).

Table 4: Upload related RPC operations that interact with the metadata store.

- [20] J. Mirkovic and P. Reiher, “A taxonomy of DDoS attack and DDoS defense mechanisms,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.
- [21] A.-L. Barabasi, “The origin of bursts and heavy tails in human dynamics,” *Nature*, vol. 435, no. 7039, pp. 207–211, 2005.
- [22] M. E. Crovella and A. Bestavros, “Self-similarity in world wide web traffic: evidence and possible causes,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, 1997.
- [23] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, “An experimental study of home gateway characteristics,” in *ACM IMC’10*, 2010, pp. 260–266.
- [24] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, “Adaptive push-pull: disseminating dynamic web data,” in *ACM WWW’01*, 2001, pp. 265–274.
- [25] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, “Early observations on the performance of windows azure,” in *ACM HPDC’10*, 2010, pp. 367–376.
- [26] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, “Amazon S3 for science grids: a viable solution?” in *ACM DADC’08*, 2008, pp. 55–64.
- [27] 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.
- [28] T. Mager, E. Biersack, and P. Michiardi, “A measurement study of the wuala on-line storage service,” in *IEEE P2P’12*, 2012, pp. 237–248.
- [29] G. Gonçalves, I. Drago, A. P. C. da Silva, A. B. Vieira, and J. M. Almeida, “Modeling the dropbox client behavior,” in *IEEE ICC’14*, vol. 14, 2014.
- [30] M. Mulazzani, S. Schrittweis, 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.
- [31] J. Silber, “Shutting down Ubuntu One file services,” <http://blog.canonical.com/2014/04/02/shutting-down-ubuntu-one-file-services/>, April 2014.

APPENDIX

A. UPLOAD MANAGEMENT IN U1

The management of file uploads is one of the most complex parts in the U1 architecture⁸. Specifically, U1 resorts to the multipart upload API offered by Amazon S3⁹. The lifecycle

⁸Downloads are simpler: API servers only perform a single request to Amazon S3 for forwarding the data to the client.

⁹<http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingRESTAPIMpUpload.html>

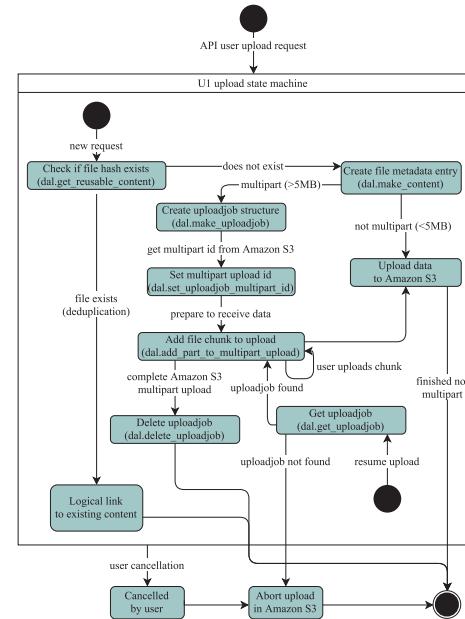


Figure 17: Upload state machine in U1.

of an upload is closely related to this API, where several U1 RPC calls are involved (see Table 4).

Internally, U1 uses a persistent data structure called `uploadjob` that keeps the state of a multipart file transfer between the client and Amazon S3. The main objective of multipart uploads in U1 is to provide user with a way of interrupting/resuming large upload data transfers. `uploadjob` data structures are stored in the metadata store during their life-cycle. RPC operations during the multipart upload process guide the lifecycle of `uploadjobs` (see Fig. 17).

Upon the reception of an upload request, U1 first checks if the file content is already stored in the service, by means of a SHA-1 hash sent by the user. If deduplication is not applicable to the new file, a new upload begins. The API server that handles the upload sends an RPC to create an entry for the new file in the metadata store.

In the case of a multipart upload, the API server creates a new `uploadjob` data structure to track the process. Subsequently, the API process requests a multipart id to Amazon S3 that will identify the current upload until its termination. Once the id is assigned to the `uploadjob`, the API server uploads to Amazon S3 the chunks of the file transferred by the user (5MB), updating the state of the `uploadjob`.

When the upload finishes, the API server deletes the `uploadjob` data structure from the metadata store and notifies Amazon S3 about the completion of the transfer.

Finally, U1 also executes a periodic garbage-collection process on `uploadjob` data structures. U1 checks if an `uploadjob` is older than one week (`dal.touch_uploadjob`). In the affirmative case, U1 assumes that the user has canceled this multipart upload permanently and proceeds to delete the associated `uploadjob` from the metadata store.