UNIVERSITAT
ROVIRA I VIRGILI

Universitat Rovira i Virgili
Department of Computer Engineering and Maths

Flexible and Realistic Client-centric Evaluation of Personal Clouds via User Stereotypes

A thesis submitted for the degree of

Master in Computer Engineering and Security

Author:
Chenglong Zou

Advised by:
Dr. Raúl Gracia-Tinedo
Dr. Pedro García López
Department of Computer
Engineering and Maths
Universitat Rovira i Virgili

September 2016

Abstract

The performance evaluation of Personal Clouds (e.g., Dropbox, Google Drive, Box) is challenging due to their *fat-client architecture* and the absence of realistic workload generators that reproduce the disparate types of *user behavior*.

In this thesis, we propose a novel storage benchmarking methodology in which the workload is generated by emulating groups of users that exhibit a similar behavior, namely, *user stereotypes*. We instantiate this methodology in `BenchBox`, a distributed performance analysis tool that integrates: (i) A *practical yet accurate model* that locally reproduces the aspects of user behavior that matter to the performance of desktop clients (e.g., types of activity, data contents); (ii) A *centralized management and real-time monitoring* system of Personal Cloud desktop clients. (iii) `BenchBox` also uses compact files to represent user stereotypes, enabling researchers to *share only the behavior of users* without disclosing private traces.

We show the benefits of `BenchBox` by exercising public and private Personal Clouds. We show how distinct types of users impact on the performance and efficiency of these systems, which may guide their optimization.

Keywords: Personal Cloud Storage, Performance Analysis, Workload Generator, Stereotype.

Agradecimientos (Acknowledgements)

I want to thank my research director Raul.G.T. and Marc.S.A. for their kind guidance and hints, Pedro.L.G. granting me the great opportunity to join the AST research group and supporting my thesis. All fellow companions that have shared the labs with me during the last year, with their expertise, help and great moments. And special thank to, Anna S., Ramon.S.H and Jordi.C.R provinding the research labs resources for my experiments. Without the facilities provided the experimental result wouldn't have been possible. And finally I want to thank my family and friends, with their encourage to enrole the studies and each of the teachers that have though me during this 6 enriching years of my life at SPSP.

Contents

List of Figures

1

Introduction

In the last decade, the concept of Personal Cloud has been materialized by several successful commercial services, ranging from *public vendors* to *on-premise private systems*. To wit, companies like Dropbox, Box or ownCloud, provide online storage, file synchronization, sharing, as well as accessibility from a variety of mobile devices and the Web [1, 2]. Personal Clouds are also becoming a popular platform to deploy external applications, such as photo viewers or document editors, that give added value to the personal storage service itself. According to market reports, these services are meeting well users needs; for instance, Dropbox has reached 500 million users in 2016 [3].

In a nutshell, most Personal Clouds exhibit a 3-tier architecture consisting of *metadata service*, *data store* and *desktop clients* [1, 2]. At the server side, their design explicitly decouples the management of file contents (*data*) and their logical representation (*metadata*); such a clean design separates the concern of metadata management, which may be stored on premise, from storage management that can be delegated to a third-party cloud service (e.g., Amazon S3). At the client side, Personal Cloud *desktop clients* provide users with automatic file synchronization across several devices, as well as intuitive data sharing capabilities. These clients transparently communicate with back-end servers via a storage protocol for mirroring local changes that users perform in their synchronized folders.

However, the growing popularity and user basis of Personal Clouds also entails stringent system design requirements towards efficiency, performance and scalability.

An attractive innovation path to optimize these systems relies on integrating of data management techniques within the system's workflow [4, 5]. Concretely, embedding storage *optimization mechanisms in the client-side* has proven to be a feasible strategy given the fat client design of Personal Clouds. The objective of client-centric data management is to minimize resource consumption at the server-side by trading-off higher load at client machines. As previous works have recently demonstrated, desktop clients may adopt a mix of compression, deduplication and advanced syncing techniques on users' requests [6, 7, 8].

In this sense, a key aspect for the design refinement of desktop clients is their *performance evaluation*, which can validate whether a new optimization technique is beneficial or not to the system via real-world experiments [9]. Nevertheless, extracting categorical conclusions from simple synthetic experiments may be risky due to the central role that the *workload of users* plays on performance [6]. In particular, the activity of users and the types of contents managed may significantly influence the operation of desktop clients.

Unfortunately, generating *realistic workloads* for evaluating a Personal Cloud is a non-trivial problem that may lead to bad practices. To inform this argument, Red

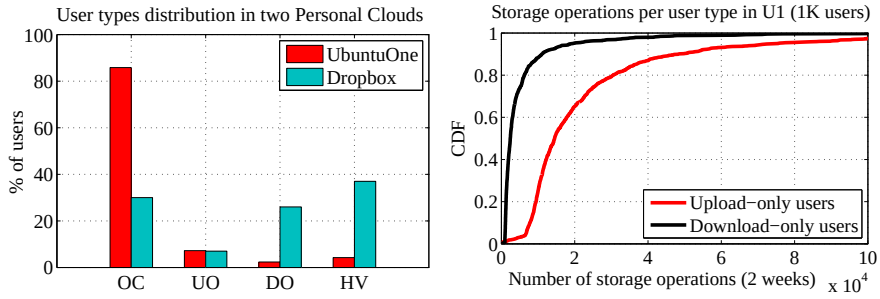

Figure 1.1: In the left, we show the classification of users in DropBox and UbuntuOne populations (*occasional* (OC), *download/upload only* (UO/DO) and *heavy* (HV)). The right figure shows the distribution of storage operations of 1K active U1 users belonging to UO/DO types (2 weeks).

Hat has recently reported a performance evaluation of an ownCloud[1] deployment, in which the system is claimed to support 25K users [10]. However, the synthetic workload used to sustain such a claim was an extreme simplification of the reality: homogeneous user behavior, regular operations (2 uploads/1 download per hour), etc. Clearly, the workload employed does not represent the *behavior of users*, which poses doubts about the performance of the system under test. Moreover, this evaluation was *exclusively server-oriented*; the behavior and resource consumption of desktop clients was omitted from the conclusions of this study, despite of being an important part of the architecture. This paradigmatic case exemplifies the problems that we address in this work.

1.1 Motivation

To motivate the importance of user behavior to generate realistic workloads, we inspect the role of users in the workloads supported by Dropbox [1] and UbuntuOne (U1) [2]. Specifically, in Fig. 1.1 we grouped users by resorting to the classification proposed by Drago et al. in [1]. 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.

We draw the following observations from Fig. 1.1:

- **System-dependent user behaviors**: Fig. 1.1 confirms the existence of different types of users in a Personal Cloud and their different distributions depending on the system at hand. This yields that replaying a workload of one system to evaluate another may be inaccurate, leading to biased insights. This calls for a *flexible workload generation* in terms of user behavior.

- **Impact of user behavior on desktop clients**: The distinct types of users in Fig. 1.1 exhibit *heterogeneous local behavior*, in terms of types of issued operations,

[1] http://owncloud.org

activity intensity or the stored files sizes/contents [1, 2], to name a few. This may impact on the *performance and overhead of desktop clients* and the interactions with the back-end in various ways [4, 6].

These observations raise the need for *client-centric* performance evaluations in Personal Clouds that cover: i) *classifying user behavior*, ii) *locally reproducing user interactions* with desktop clients, and iii) *monitoring client resources*. Unluckily, today's benchmarks are mostly server-oriented and do not reproduce user behavior in such fine grained manner.

1.2 Problem Statement

Most of today's standard storage benchmarks generate synthetic workloads to stress the system under configurable requests rates [6, 11, 12, 13]. They enable practitioners to perform *what-if* testing, but not to understand the system's operation under realistic conditions. There exist another family of advanced benchmarks that focus on modeling system-wide workload properties instead of emulating the behavior of users; for instance in the case of file systems [14], cloud services [15], graph databases [16], or recently in Personal Clouds [17].

However, these tools fall short when modeling and reproducing the behavior of Personal Cloud users in a distributed, fine grained manner. This poses several drawbacks: i) *The workload generation is not client-centric*, as scaling the intensity of a workload is done in a coarse, system-wide manner; i) *Lack of flexibility*, as it is unfeasible to generate workloads with different distributions of user types; iii) Moreover, most tools *lack from client-side monitoring metrics* to assess the effectiveness of data management techniques.

In summary, in this thesis we face the problem of generating user-driven storage workloads in a client-centric manner to evaluate Personal Cloud desktop clients. Clearly, solving this problem calls for a general methodology that enables researchers to design and integrate a new client-centric workload models in a performance evaluation, as well as providing them with means to analyze the results of such evaluation.

1.3 Contributions of this Thesis

The contributions of the present thesis are the following:

- We present a *client-centric benchmarking* methodology based on *user stereotypes*. An stereotype describes the behavior of a group of users with similar storage activity, which enables flexible and realistic performance evaluation of Personal Clouds.

- We contribute a practical yet accurate generative model that *reproduces the behavior* of user stereotypes. Technically, the model captures a set of dimensions that matter to the performance of desktop clients (type of requests, file system contents, etc.).

- We design BenchBox: A client-centric benchmarking tool. BenchBox enables practitioners to generate workloads consisting of variable rates of user stereotypes, as well as to monitor the activity of desktop clients. BenchBox also gives practitioners the ability to share compact *stereotype recipes* that describe the behavior of users without disclosing private traces.
- We demonstrate the potential of BenchBox by comparing public Personal Clouds. BenchBox sheds light on the performance effects induced by disparate user behaviors, which can help designers to optimize the system.

1.4 Thesis Structure

The remaining part of this thesis is structured as follows. In Chapter 2, we present the necessary background and concepts used through the remainder of the thesis. Moreover, in the last part of this Chapter we overview the related work.

In general terms, Chapter 3 describes the client-centric performance evaluation methodology proposed in this thesis. We instantiate our methodology with a practical workload generation model build upon an analysis of real users of UbuntuOne, the Personal Cloud of Canonical. Moreover, we present the design and architecture of our benchmarking tool, BenchBox.

Chapter 4 presents a battery of real performance evaluations carried out with BenchBox against several public Personal Cloud systems. The objective of such evaluation is to demonstrate how the local behavior of users impact on Personal Cloud desktop clients, which can guide their optimization.

Finally, we draw some conclusions and future research directions in Chapter 5.

1.5 Publications

The results of our work are reflected in the following publications:

- Raúl Gracia-Tinedo, **Chenglong Zou**, Pedro García López, Marc Sánchez Artigas. *"BenchBox: Client-centric Evaluation of Personal Clouds via User Stereotypes"*. (To be submitted to *IEEE Transactions of Parallel and Distributed Systems*).

2

Background and Related Work

In this Chapter, we aim at providing an overview of Personal Clouds, as well as the necessary related concepts and definitions to facilitate the understanding of the rest of this thesis.

2.1 Personal Clouds: Concept & Landscape

Personal Clouds services are cloud storage services that provide users with three main functionalities: storage, file synchronization and sharing (3S). Moreover, today's Personal Cloud also provide users with desktop, mobile and Web clients that allow them to manage their data in a wide variety of Operating Systems (OSes) and platforms [18].

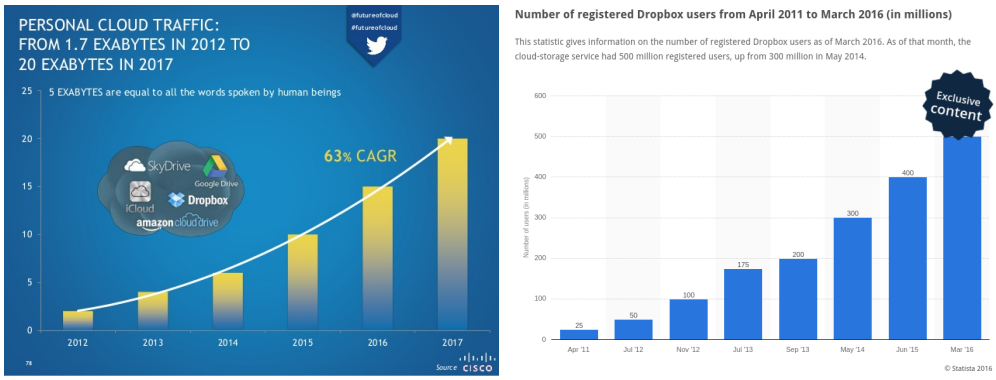

Figure 2.1: The left figure shows the increasing share of personal cloud traffic, and the right figure shows the increasing amount of users of Dropbox

This concept has revolutionized the way users manage their data. Nowadays, Personal Cloud services like Dropbox, Google Drive or Box support millions of users over the globe, and their user populations are still growing. The amount of traffic per year of these services has to handle is estimated over 15 EBytes ($15 * 10^{18}$ Bytes), which is equivalent to 41,095TB everyday. See figure 2.1. This milestone has been satisfied with the improvement of the overall Internet connection, and the cloud providers infrastructure, which aim at improving their performance in parallel with cost reduction.

There are many reasons that have contributed to the success of Personal Cloud storage services, with the emerging of countless personal cloud services, leaded by Dropbox, OneDrive, Google Drive, Box, Amazon Cloud Drive. Overall, facts such as the ongoing trend where people own many devices, their need to access their data from several spots, or even to share them in an effective way are the primary rea-

sons. This is possible greatly because the reduction of resource cost, such as network bandwidth, storage fees and the availability granted by the cloud providers with wide accessibility.

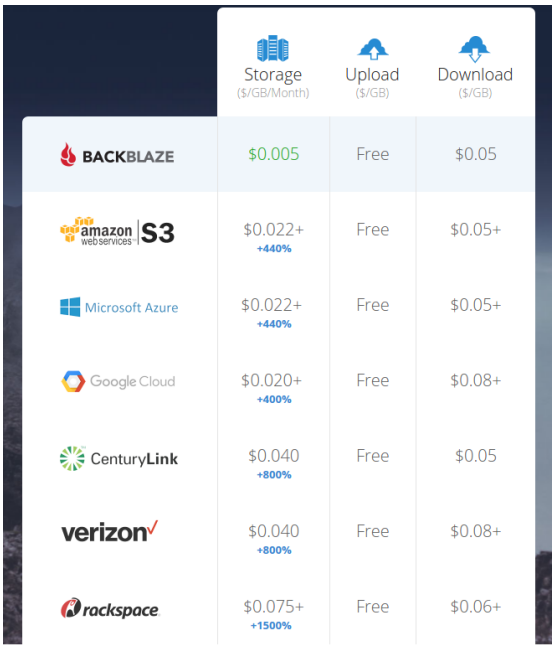

Figure 2.2: The table displays a comparison of some public cloud data store plans.

Comparing figures 2.2 and 2.3 we can notice that there are over two order of magnitude of benefit margin from the subscribed client, and the cost to maintain free user accounts is marginal.

2.2 Personal Cloud Architecture

The architecture of Personal Cloud has changed over time. From its origins with an architecture of single server that contains all the service dependencies (web server, app server, mySQL database, sync server) serving few clients. As visible in Fig. 2.4, the architecture of Personal Clouds nowadays contains additional services such as load balancers and caching to speedup the performance and improve the user experience. The current trend is to replicate several instances horizontally to handle the amount of simultanious connections and requests.

An overview of the current personal clouds services core components from the client and server perspective is depicted in 2.5.

- **Front-end: Desktop Client**. The desktop clients are software applications that run at the client machine and acts as a manager of the file synchronization process of Personal Cloud storage services. This process is hold in background and submits automatically once the clients goes online, if the sync folder has been modified, the client might push the editions to all the clients to whom it share the content with. Additional features allow users to selectively bind folders to sync them automatically with the cloud storage service data store. It might also index

Cloud storage comparison

	OneDrive	Dropbox	Google Drive	Box	Amazon Cloud Drive
File size restrictions?	10GB	10GB with website, none with Dropbox apps	5TB	250MB for free plan, 5GB for paid personal plan	2GB*
Free storage?	5GB**	2GB	15GB	10GB	No***
Can I earn extra free storage?	No**	Yes	No	No	No
Paid plans	$2/month for 50GB**	$10/month for 1TB	$2/month 100GB, $10/month for 1TB	$10/month for 100GB	$12/year for unlimited photos, $60/year for unlimited files
OSes supported	Windows, Mac, Android, iOS, Windows Phone	Windows, Mac, Linux, Android, iOS, Windows Phone, BlackBerry, Kindle Fire	Windows, Mac, Android, iOS	Windows, Mac, Android, iOS, Windows Phone, BlackBerry	Windows, Mac, Android, iOS, Kindle Fire

Figure 2.3: The figure displays a comparison of the current cloud storage services pricing for additional storage.

locally the remote cloud storage container, allowing the client to download files, or share them generating a direct access link. In addition to the basic features, desktop client can implement, file compression, file encryption, file deduplication, file versioning and it may also notify remote file updates or simultaneous file edit conflicts.

- **Back-end: Storage Back-end and Data Synchronization Service**. Desktop client interacts with the synchronization server to submit the events that take place in the synchronization folders. This events use to be communicated through a middleware that handles the events and stores the client current state as meta-data. The meta-data generated are processed by the synchronization service and converted into operations at the data store back-end as file system operations.

  On the one hand, the main role of the synchronization server is to provide the distributed file synchronization. It forwards the notification events through other clients that share the file and secures the consistency of the shared data within a container.

  On the other hand, the task of data store back-end is to store the content of the Desktop client, this service infrastructure might be outsourced using public storage (public cloud), it might be hosted within local facilities (private) or using both approaches (hybrid cloud).

2.3 Personal Cloud Data Management

Data management techniques attend to improve the performance or efficiency of Personal Cloud services. Fig. 2.6 illustrates the currently trending data management techniques implemented in desktop clients of several providers.

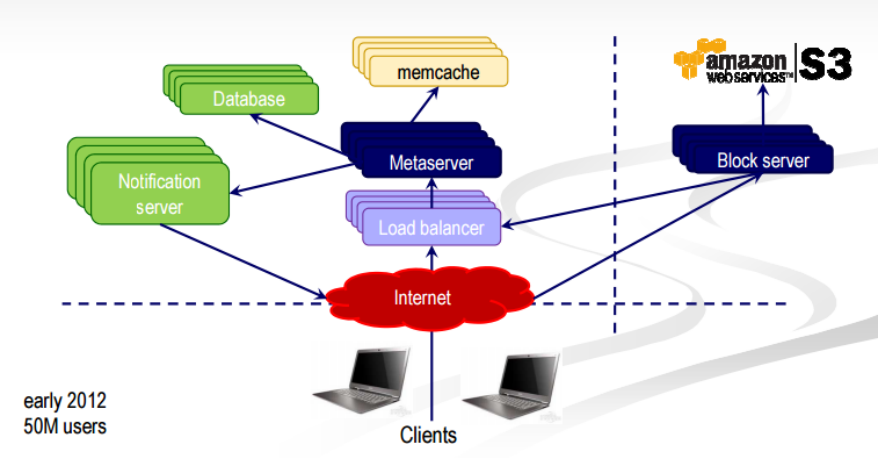

Figure 2.4: The figure displays the Dropbox deployment at 2008 which was able to handle up to 50 million users, the three tier Personal Cloud architecture: Cloud Storage, Desktop Client and API server with Metadata store.

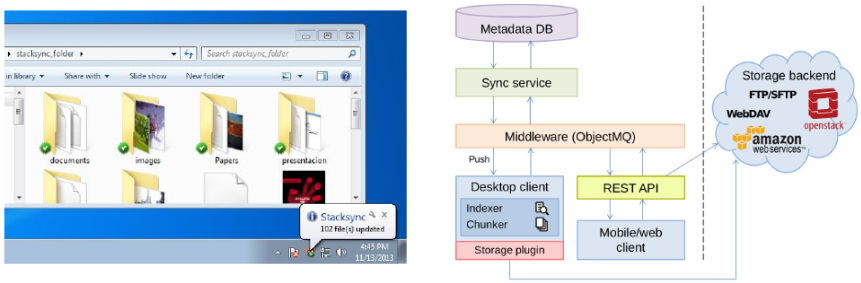

Figure 2.5: In the left figure shows the front-end Desktop Client notification icon and back-end components of StackSync.

- *Chunking*: Split large files into fixed size data units, where the unit refers to the unit of deduplication, delta encoding and compression. This feature simplifies fault recovery, which allows partial transmissions of missing or corrupted chunk but each chunk submission is delimited by a pause, introducing delay to the delivery.

- *Bundling*: Send multiple small files as one chunk, this might improve the throughput, needing less connections and sending more compact data packets.

- *Client side deduplication*: This feature avoid the client sending chunks that are already stored on servers, this feature spare the client to upload once same files located at different folders.

- *Delta encoding*: When a files is modified the client will send only the diff of old and new chunks, this feature might spare network traffic uploading only the modified chunks instead of the whole file.

- *Data compression*: The client compress the file per chunk before sending them to the server, this process might reduce the file size to be uploaded, thus reducing

	Dropbox	SkyDrive	Wuala	Google Drive	Cloud Drive
Chunking	4 MB	variable	variable	8 MB	✗
Bundling	✓	✗	✗	✗	✗
Deduplication	✓	✗	✓	✗	✗
Delta encoding	✓	✗	✗	✗	✗
Compression	always	never	never	smart	never

Figure 2.6: The table shows the supported features of current most trending public personal clouds services.

the synchronization time and network traffic, but it imply processing to compress as drawback, which may drain the battery live if it is a portable device.

2.4 Personal Cloud Storage Developer API

The personal cloud storage APIs(Application programming interfaces) provide access to the personal storage content, some common feature is providing a RESTful Web service API, which enable service customers to create applications to collect or serve their personal cloud data.

During our experiments we use these APIs at the workload generator for pushing files to the personal cloud container to simulate Download behaviors of the desktop client under experiment.

2.5 Related Work

Benchmarking tools. The performance evaluation of cloud and storage systems has been an active research topic in the last years [19, 20]. An excellent survey of Traegger et al. [9] provides a comprehensive taxonomy of a decade in benchmarking literature (see references thereof).

Roughly speaking, we can distinguish between *non-informed* and *informed* benchmarking tools; non-informed benchmarking tools provide practitioners with means to perform *what-if* tests based on "turning knobs" [15, 21, 22, 23, 24, 25, 26, 27, 28]. In other words, these tools facilitate to synthetically generate workloads guided by definable parameters (i.e., operations/second, file size) for exploring the system performance under controlled conditions. Non-informed benchmarks represent a necessary first step to evaluate a system, but they are not devised to reproduce realistic workload conditions.

To overcome this limitation, informed benchmarking tools guide their workload generation based on trace replays or models (e.g., Markov chains) that approximate workloads found in production environments [16, 29, 30]. Seminal works in the field identified structural aspects of workloads to enable their synthetic generation, such as self-similarity and burstiness [31, 32, 33]. Recent efforts advocate to build a control loop to automatically infer workload models for their subsequent execution [34]. Clearly, informed benchmark models represent a significant advance towards better

understanding the performance of a system in the wild. In fact, `BenchBox` enables practitioners to *easily deploy new workload models* that reproduce the behavior of users.

However, to our knowledge, most informed benchmarks are exclusively *server-side oriented*; that is, a set of generation instances generate a synthetic workload that matches a real one in *aggregated terms*. Despite this model is suitable to many thin client services (e.g., Web servers, K/V stores), the fat client architecture of Personal Clouds calls for enacting *clients as the subject of the benchmark* [35]. Otherwise, the system's evaluation would be partial and incomplete [36].

Closer to this work, there are only few efforts in the literature that targeted the evaluation of Personal Clouds. On the one hand, Bocchi et al. [6, 8] and Li et al. [7] presented deep analyses of various desktop clients based on synthetic non-informed benchmarks. Similarly to `BenchBox`, these works measured performance metrics and traffic overheads at desktop clients; in fact, `BenchBox` extends these efforts by also monitoring resources at client machines (CPU, RAM, etc.). However, a key difference between these works and the present one lies on the evaluation methodology; [6, 7, 8] resorted to artificial workload patterns to infer the behavior of desktop clients, but providing no answers on their performance under user-driven activity.

On the other hand, the only informed Personal Cloud benchmarking tool is that of Gonçalvez et al. [17]. Authors propose CloudGen, a server-oriented benchmark that generates Dropbox-like workloads based on hierarchical model. Hence, despite fitting the aggregated workload of Dropbox users, authors in [17] do not involve desktop clients in their benchmarks, making it impossible to assess client-side data management optimizations. Conversely to CloudGen, `BenchBox` enables to orchestrate and monitor a set of desktop clients driven by different behaviors (i.e., *stereotypes*), paying special attention to the fidelity of contents generated during the evaluation process (data compression, deduplication, folder structure, etc.) [37, 38].

Personal Cloud measurements. `BenchBox` is designed to facilitate practitioners measuring the performance of Personal Cloud desktop clients under user-driven activity. In this sense, several efforts have been devoted to analyze these system, which can be divided into *active* and *passive* measurements. In general, active measurements try to infer the interactions of desktop clients and the server as well as the different data management techniques embedded on them [4, 6, 7, 8] —despite some works also actively evaluate server side components [39]. The first work in this sense is [40], in which authors evaluated simple transfer desktop client backup/restore times in several vendors depending on types of backup contents. More extensive efforts [4, 6, 7, 8] revealed that Personal Cloud desktop clients currently integrate combinations of techniques like compression, deduplication or file bundling, to name a few. Thus, depending on the workload intensity, file contents and even the size of sharing groups, desktop clients exhibit different performance trade-offs, which can impact on the user quality of experience (QoE) [41]. Therefore, our work aims at *complementing existing efforts* with a flexible way of reproducing the behavior of users at desktop clients.

Regarding passive measurements, Drago et al. [1] 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. [42] extended that measurement by modeling the behavior of Dropbox users. Similarly, Mager et al. [43] uncovered the architecture and data management of Wuala, a peer-assisted Personal Cloud. More recently, a back-

end measurement of UbuntuOne was presented in [2], unveiling relevant aspects of the service that are impractical to measure from outside (e.g., global population dynamics, attacks). In our view, these works provide capital insights about how users behave in a Personal Cloud. We aim at *exploiting such information* to extract different user stereotypes, which can then be *plugged-in and reproduced* by our benchmarking tool.

User behavior & workload modeling. In contrast to benchmarks that reproduce system-wide models, few prior works focused on generating workloads at the user granularity [22, 29, 44]. In this regard, most of these works target to generate realistic HTTP traffic for Web applications. For instance, authors in [29] presented SURGE, a synthetic workload generator for testing request-based systems. They are the first to present the notion of *user equivalent*, which is defined as a single process in an endless loop that alternates between making requests for Web files, and lying idle. Authors in [22] extend the notion of user equivalent by taking care of dependencies among user operations, which is achieved by replaying user sessions with similar sequences of operations. Similarly, Walty [44] is a benchmarking tool that resorts to user-level workload modeling and aggregates similar users into *profiles* to generate Web traffic.

Conceptually, the notion of user equivalents or profiles resemble the idea of *stereotypes*, as they aim at grouping user behaviors together under some criterion of similarity [45]. However, we find key differences with prior works: i) User stereotypes instances are executed *individually per client machine* to analyze the performance of desktop clients, instead of being a executed as a collection of processes loading the server. ii) User stereotypes do not only reproduce the types of user activity, but also the *contents related to user types*. iii) Finally, our methodology enables stereotypes to be *expressed as a recipe*, enabling researchers to share only the behavior of users, instead of large and potentially private traces.

3

BenchBox: Personal Cloud Benchmarking via User Stereotypes

3.1 User Stereotype Benchmarking

In sociology, a stereotype is defined as a "set of simplistic generalizations about a group that allows others to categorize them and treat them accordingly" [46]. Similarly, in terms of storage, this work we focuses on identifying *stereotypes* that describe the behavior of users that share common workload characteristics in a Personal Cloud. As we show later on, this approach enables practical and realistic workload generation, as well as flexibility to evaluate the system under different user populations.

3.1.1 Goals

As occurs in other fields [35], today's Personal Cloud benchmarks advocate either to exercise desktop clients in a simplistic manner [6, 7, 8] or to generate aggregated synthetic workloads for the server-side [17]. The unintended consequence of this situation is that it leaves an important gap: Practitioners lack from a methodology for *identifying, reproducing and sharing* user behavior at fine grain to evaluate its repercussions on desktop clients. We interpret the term "repercussions" in a broad sense, ranging from performance effects to the perceived user QoE.

A benchmarking tool implementing such methodology may benefit practitioners and researchers in several ways. For instance, it would represent a client-centric approach for *exercising and monitoring desktop clients* under user-driven workloads. Second, it can be seen as a general way to *model and share user behavior*; that is, researchers can design new *workload generators that take as input concise recipes* of user stereotype behavior, thus avoiding to share large and potentially private traces. Even more, considering on premise Personal Cloud installations, administrators may also evaluate the overhead that *certain types of users incur in the server side*, which may lead to the enforcement of specific storage policies or even help in capacity provisioning.

3.1.2 Benchmarking Methodology

Next, we describe our benchmarking methodology based on user stereotypes. It consists of the following stages:

Stereotype identification: The first stage in our methodology is to groups users into a stereotype according to some criterion, such as the similarity of their *activity or data sharing relationships*. Naturally, the grouping criterion may depend on the practitioner,

the technique used to cluster users, or even on the evaluation's goals. Ideally, stereotypes should have clearly differentiated behaviors in terms of storage to evaluate desktop clients under disparate situations.

Sharable workload model: Once groups of users are classified, the next decision to take is *selecting the aspects of their behavior that matter for an evaluation*. Naturally, there is not a universal model; for instance, one practitioner might be interested on modeling in high precision the inter-operation times of users and the types of files being managed, whereas another may aim at capturing their connectivity patterns (e.g., non-stationarity). Overall, the behavioral aspects extracted of each user stereotype should be described in a *recipe*; it is a quantitative representation of the aspects of user behavior to be reproduced (e.g., fittings, activity rates).

Stereotype workload generation: At each desktop client, workload generation processes take as input stereotype recipes to execute an evaluation. The objective is to build a generation process able to behave similarly to the stereotype built from real users. Thus, the decision of "what" dimensions to model is inherently related with "how" they are modeled, despite multiple implementations are possible. In our scenario, we also differentiate two layers involved in workload generation: i) the *activity layer*, which dictates the operations to be executed and when these events occur, and ii) the *data layer* that mimics the management of user files and the structure of synchronized folders.

Client-centric performance evaluation: The last step of our methodology is to execute one or many workload generation processes, one per each desktop client. This stage also embraces the monitoring of metrics at the client side, such as resource consumption, performance of storage operation or even traffic overheads. To this end, a *benchmarking tool* should provide practitioners with means of orchestrating workload generators to be executed on desktop clients, managing stereotypes recipes to feed generators and performing extensive client-side measurements.

Next, we proceed to instantiate this methodology.

3.2 Towards Personal Cloud Stereotypes

Perhaps, the first question to formulate in our methodology is how to define user stereotypes within a system. As happens with user identification in other domains [45, 47], it is hard to have absolute definitions of user types.

To inform this argument, we analyzed the behavior of users in U1. Specifically, in Fig. 3.1 we resorted to two different clustering algorithms —K-means (distance-based) and DBScan (density-based)— in order to automatically classify users into stereotypes. To this end, we picked two aspects of user behavior that impact on desktop clients: *upload* and *download* traffic. Fig. 3.1 confirms our intuition: There is *not a universal way of classifying users*. That is, K-means groups users depending on the *types of operations* (more upload or download oriented), whereas DBScan forms clusters depending on the intensity of user activity (data transferred). Naturally, other algorithms may exhibit different results, specially if we consider more dimensions to cluster users.

Overall, despite these algorithms may not provide universal classification rules, they can tell how to group users based on specific aspects of their behavior. To wit, Fig. 3.1 yields that we can create meaningful user groups based on the *type of traffic*

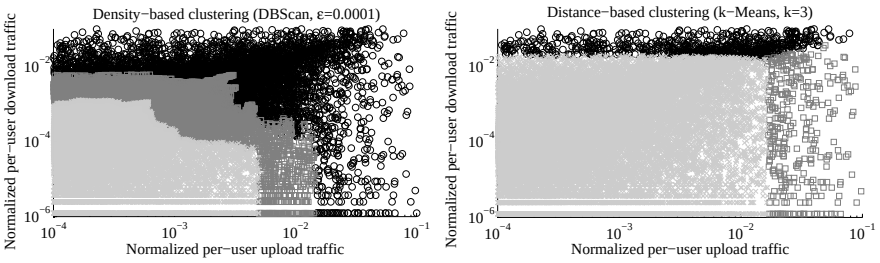

Figure 3.1: Clustering U1 users with *DBScan* (density-based) and *k-means* (distance-based). DBScan groups users depending on their workload intensity, whereas K-means clusters users based on their workload type.

and the *intensity of user activity*. Thus, we only need to define quantitative criteria to generate these groups. Regarding the former aspect, we classified U1 users into the following stereotypes:

Back-up users: Fig. 3.1 shows that there are users that exhibit a much higher volume of upload traffic compared to download traffic (i.e., backup activity). Our criterion to classify a user as a backup user is that he/she exhibits more than two orders of magnitude of difference between upload and download operations.

Sync users: We detected that there are users exhibiting backup activity that is mainly due to *file updates*. As updating file involves complex synchronization activity of interest for a performance evaluation, we aim at classifying such users within a separate stereotype. Thus, we defined that a use exhibiting a number of file updates more than twice the number of unique files is a sync user.

Content distribution users: Due to desktop client syncing or sharing activity, there are users that exhibit high amounts of download traffic. In particular, we classify a user whose download operations are two orders of magnitude higher than his upload operations as content distribution.

As pointed out by Fig. 3.1, we observed that the activity of users in a Personal Cloud is highly skewed, showing a form of Pareto principle [2] (i.e., 80/20 law). That is, there is an small amount of users exhibiting high activity, whereas the activity of most users is modest. We detected that this phenomenon is orthogonal to any user type. Thus, we augmented our stereotypes with **heavy** or **occasional** sub-classes; heavy stands for the 20% of most active users in a sterotype, whereas the remaining 80% are occasional.

Moreover, the concept of user stereotype may not only refer to individual user characteristics, but also to ones related to group dynamics. This is the case of classifying sharing groups in a Personal Cloud. That is, considering a data sharing interaction between two users as a link, we recently found that users are organized into sharing groups exhibiting different topologies and degrees of connectivity [48]. To capture this phenomenon in our evaluation, we propose to form groups in which users are highly interconnected (**clustered**) and groups in which a single hub user capitalizes most of the sharing links (**non-clustered**).

Although our methodology is open to more complex user classification, our objective is to show how the proposed stereotypes impact on the operation of desktop

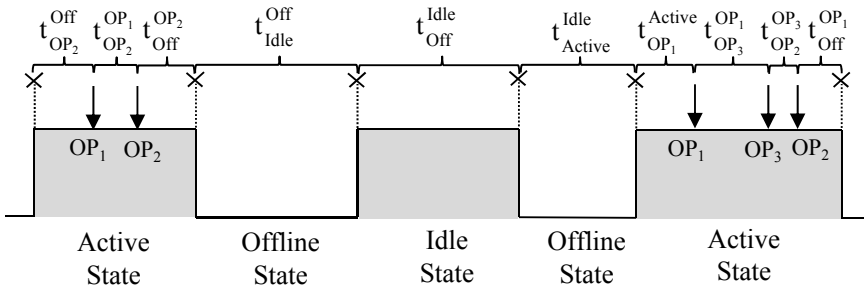

Figure 3.2: Activity states of a user in U1.

clients. To this end, we show next our workload generation model that translates users stereotypes into storage activity.

3.3 Activity Layer: Modeling Active Users

In this section, we present a generative workload model to evaluate the impact of user behavior on Personal Cloud desktop clients. As defined in our methodology, the workload model takes as input user stereotype recipes in order to emulate different types of user behavior.

Our workload model captures several aspects of user behavior that we deemed most relevant for the performance of desktop clients: *type and intensity* of user operations. This demonstrates how our benchmarking tool allows researchers to design and deploy new workload models.

3.3.1 Workload Model State Definition

To model the activity of users, it is first required to define the states that we aim at emulating from users [42]. In this sense, Fig. 3.2 illustrates how a user may interact with the system in U1. Particularly, we distinguish 3 states: i) *Active* (a user is online and executes storage operations against the system), ii) *Idle* (a user is online but he/she does not exhibit storage activity), and iii) *Offline* (a user is disconnected).

In this work, we are interested on modeling the behavior of a user when she/he is online to execute short/medium term experiments. For this reason, we focus on modeling *Idle* and *Active* states, thus discarding offline sessions[1]. As we show later on, a model considering *Idle*/*Active* states can satisfy the requirements imposed by our stereotypes: First, it enables to generate different types of storage operations while in *Active* state to emulate distinct types of stereotype activity (e.g., backup, sync); and second, it allows to capture the activity rate of users (heavy, occasional).

As visible in Fig. 3.2, in *Active* state a Personal Cloud user may perform several operations on his desktop client. Thus, similarly to the interactions with a file system, we consider the following operations in our model: `Upload`, `Download`, `Remove`, `Move` and `Sync` (i.e., update). The execution of such operations on files/directories within

[1]For long term evaluations, our model can be extended by considering disconnection periods, but this goes beyond the scope of this work.

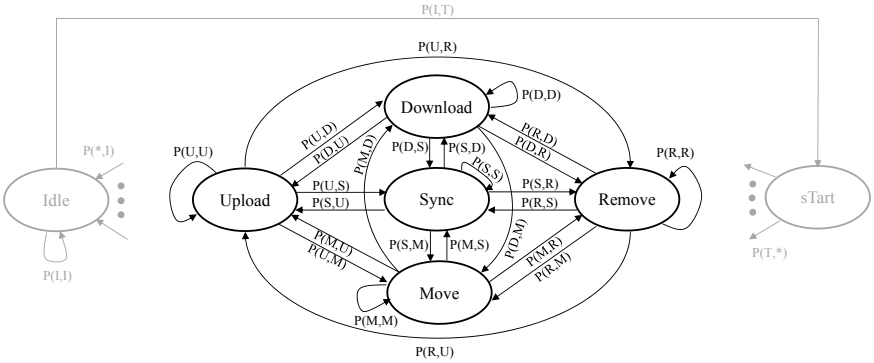

Figure 3.3: Markov model for handling user operations. State transitions are denoted tuples formed by the first letters of the source and destination states. To wit, (S,M) denotes a `Sync`→`Move` transition and P(S,M) is the transition's probability. These probabilities build the transition matrix.

a sync folder trigger management tasks at the desktop client. Thus, we need a workload generator process to model these storage operations as well as for transitioning between *Idle*/*Active* states within the context of a desktop client.

3.3.2 Semi-Markov Process for Modeling User Activity

We resort to a Semi-Markov Process (SMP) to guide the execution of a workload generator [49] (see Fig. 3.3). Formally, let us define $S = \{s_i\}$ as a discrete state space in which a user may transition. Let us also consider a vector $\mathbf{p} = [p_i : i, i \in S]$ that defines an initial distribution of the Markov renewal process. Given that, we define a renewal matrix $\mathbf{Q}(t) = \left[Q_{ij}(t) : i, j \in S\right]$ that describes the probabilities of transitioning from a state i to a state j at time $t \in \mathbb{R}_+$. Therefore, for each state $i \in S$, $\lim_{t\to\infty} \sum_{j\in S} Q_{ij}(t) = 1$.

From definition of the renewal matrix it follows that:

$$\mathbf{P}(t) = \left[p_{ij} : i, j \in S\right]$$

$$p_{ij} = \lim_{t\to\infty} Q_{ij}(t)$$

is a stochastic matrix. It means that for each pair $(i, j) \in S \times S$, where $p_{ij} > 0$ and for each $i \in S$, $\sum_{j\in S} p_{ij} = 1$. Besides, in a SMP we define a function:

$$\mathbf{F}(t) = \frac{Q_{ij}(t)}{p_{ij}} \quad i, j \in S, t \geq 0$$

as a CDF of some random variable, which is denoted by T_{ij} and it is called a *transition or sojourn time* [49] in state i, if the next state will be j. Therefore, a triple $(\mathbf{p}, \mathbf{Q}(t), \mathbf{F}(t))$ determines the homogeneous SMP with the state space S.

In practice, modeling user behavior as a SMP involves three tasks: i) to estimate initial state probabilities ($\mathbf{p}$), ii) to estimate the probabilities of transitioning among states ($\mathbf{Q}(t)$), and iii) the time spent on each state prior the transition ($\mathbf{F}(t)$) (see Section 3.3.3). These sources of information are expressed in a compact way in our stereotype recipes.

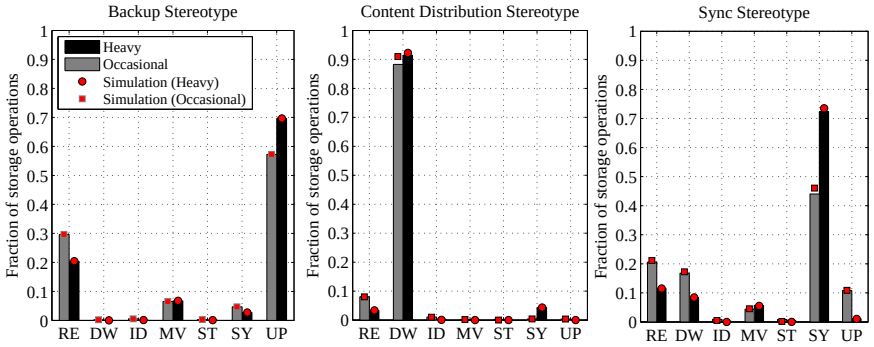

Figure 3.4: Simulation results of our workload model against a U1 trace replay of 1K users per stereotype during 2 weeks.

Regarding the first two aspects, Fig. 3.3 provides a representation of the possible states and transitions that we define in our SMP. That is, while the user is active, she/he may perform any of the proposed storage operations (e.g., store, sync). Moreover, our model also considers to transition between *Idle* and *Active* states. Thus, we first estimate $\mathbf{p}$ by computing the probabilities of a U1 user to start at a certain state i in our traces. Similarly, for building $\mathbf{Q}(t)$ in our recipes, we estimate a user's transition probabilities from state i to state j as the observed number of $i \rightarrow j$ transitions divided by the total number of visits to state j.

Next, we aim at validating that our model emulates the fraction of storage operations exhibited by different stereotypes. To this end, Fig. 3.4 shows the fraction of storage operations of 1K users per stereotype selected in U1 for 2 weeks. Fig. 3.4 also illustrates average results of 50 simulations consisting of 100 simulated users running our model for the same period of time. The model was fed with the estimated SMP depending on the target stereotype. Appreciably, our simulation results accurately fit the storage activity of real users; this yields that our model emulates the popularity of operations across stereotypes. In particular, for popular stereotype operations, our simulations do not exhibit a deviation higher than $\pm 5.8\%$, which is suitable for benchmarking purposes. We also observed that for unpopular operations, such deviation becomes higher. For instance, for `Backup-Heavy` and `Backup-Occasional` users, `Download` operations represent 0.02% and 0.24% of the total amount of operations. In this case, our simulations exhibit an error of 23.08% and 11.75%, respectively.

3.3.3 Distributions of Transition Times

In our SMP workload model, apart from deciding "which" will be the next operation to execute, we should also decide "when" such operation will be executed. Naturally, this aspect inherently dictates the *intensity of user activity*.

In our workload model, we emulate inter-operation or transition times by estimating $\mathbf{F}(t)$. Given a set of U1 users of a certain stereotype, we capture all the times related to a certain state transition (e.g., from state `Upload` to state `Download`). Then, we compare the empirical transition time distribution against a battery of well-known distributions with maximum likelihood estimated parameters [17, 42]. In our stereotype recipes, each transition is associated with the best fitting distribution tested.

To validate the actual accuracy of this approximation, we compare results from

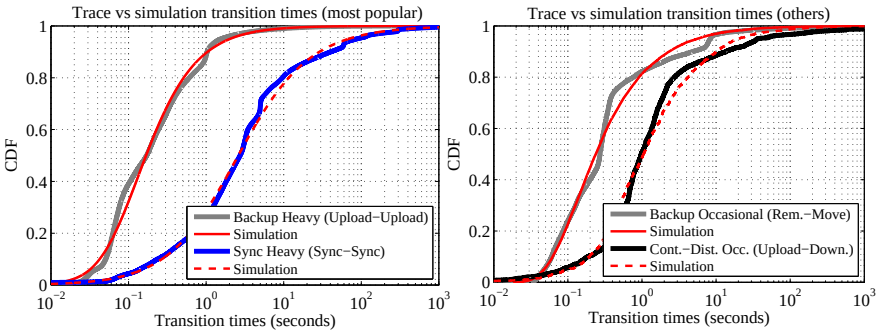

Figure 3.5: Empirical versus simulated transition times.

our selected U1 users with simulated users running our workload model. Concretely, Fig. 3.5 (left) shows the times between consecutive uploads and file update operations for `Backup-Heavy` and `Sync-Heavy` users, respectively. In this case, these transitions are the most frequent ones regarding these stereotypes. First, we observe that inter-operation times in our simulations closely fit empirical distributions. This occurs even for disparate inter-operation distributions. That is, times between uploads for `Backup-Heavy` users are much shorter than times between `Sync` operations. This may be due to the fact that consecutive uploads may be automatically triggered by storing entire directories, whereas file updates are mostly human interactions with files [2]. These observations are consistent with Fig. 3.5 (right), where our simulations present a similar shape compared to less popular empirical state transitions of `Backup-Occasional` and `Content Distribution-Occasional` users.

3.4 Data Layer: Mimicking Sync Folders

The second building block of our modeling framework focuses on generating realistic file contents for users. At the data layer, our model targets the following aspects: i) Directory structure of sync folders, ii) generation of realistic files in terms of size and contents, and iii) mimic file edition.

3.4.1 Sync Folder Structure

To evaluate Personal Cloud desktop clients under realistic conditions it is important to model how the *structure of sync folders*. That is, desktop clients normally run *watcher processes* that monitor one (or multiple) sync folders to detect changes between their local and remote states, which may trigger synchronization. Thus, mimicking the structure of a sync folder can be relevant, as it may influence the performance of watcher processes (e.g., file search, indexing [14]) and the metadata management on the server side.

We found significant differences among user stereotypes in terms of the structure of sync folders. To inform this argument, we analyzed the number of directories to which 1K U1 users of each stereotype worked with for 2 weeks. That is, we found that `Backup-Heavy` users are related in average with 280 directories, whereas for `Backup-Occasional` users this value is 46 (for `Content Distribution` and `Sync`

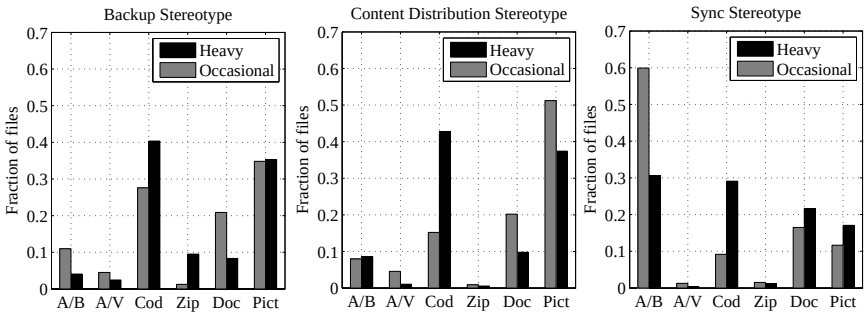

Figure 3.6: Fraction of files of each category per user stereotype.

heavy/occasional users these numbers are 125/8 and 17/9, respectively). Despite this observation does not provide the exact number of directories per sync folder, it offers a worst-case value to generate a realistic sync folder structure. In our recipes, we use these values as an upper limit to initialize the number of directories in a sync folder.

Given the number of directories per sync folder, we then generate an initial snapshot of a sync folder's structure that emulates important aspects observed in users' file systems, such as the directory size (subdirs) or the directory tree depth. To this end, we resort to an existing approximation proposed by Agrawal et al. [14] to generate an initial directory tree "snapshot". This snapshot provides the sync folder with a realistic structure that will be mutated at each directory-level operation during the experiment execution.

3.4.2 File Generation

Upon a file operation desktop clients apply data reduction techniques, such as compression or deduplication, to optimize their operation [7]. Clearly, file sizes and contents (e.g., text, random data) greatly influence the performance of data reduction techniques [38].

To address the generation of files, we propose classification of files based on the different *file categories* found in a Personal Cloud. A category encompasses types of files (extensions) that share similar contents and/or usage patterns. We propose the following categories [2]: `Pictures` (e.g., .jpg, .png), `Audio/Video` (e.g., .mp3, .avi), `Docs` (e.g., .doc, .pdf), `Code` (e.g., .py, .c), `Compressed` (e.g., .zip, .gz) and `Application/Binary` (e.g., .exe, .jar).

Once defined the desired file categories, our model enables to specify for each stereotype recipe the fraction of files of each category, since they may differ among stereotypes. To inform this argument, in Fig. 3.6 we show the fraction of files per category and user stereotype. As a common characteristic, considering `Heavy` stereotypes, `Code` files are quite popular among all user types. However, we found that `Application/Binary` files are common for `Sync` users, but it is not the case for `Backup` and `Content Distribution` users. Also, we observed significant differences among `Heavy` and `Occasional` users within the same stereotype. That is, `Content Distribution-Heavy` users manage 42.8% of `Code` files whereas for `Content Distribution-Occasional` users this value is 15.2%; `Content Distribution` users show the opposite behavior for `Docs`.

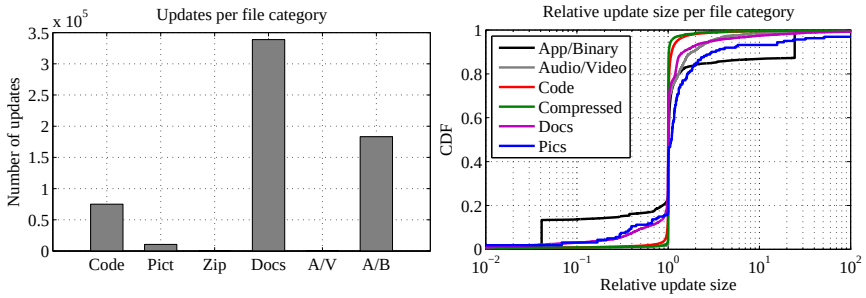

Figure 3.7: Number of edits per file category (left) and relative size of updates (right) per file category.

In our model, each file category is described by the *distribution of file sizes*[1] —that we use to create files of appropriate sizes— and a *content characterization* —that helps us to fill synthetic files with realistic contents. Regarding the latter aspect, we generate realistic file contents in terms of compression and deduplication. First, we resort to our previous work [38] to scan standard datasets that represent the contents of the defined file categories to obtain a dataset characterization file. Such dataset characterization enables us generating similar synthetic content in terms of compression times and ratios. Moreover, we extend this effort by also considering deduplication at the file level. That is, stereotype recipes incorporate a deduplication ratio parameter to allow the workload generator deciding whether to generate a file with existing contents or not. In the case of U1 users, file-level deduplication has been estimated to be 17% [2].

3.4.3 Mimicking File Edition

File edition is one of the most important types of user actions in a Personal Cloud. To provide a practical model of file updates, we augmented the concept of file category with: *edition probability*, *edit size* and *types of edits*. First, as one can infer from Fig. 3.7, not all types of files have the same *chances of being updated*; for instance, in relative terms document files account for 55.7% of file updates, whereas for `Audio/Video`, `Pics` and `Compressed` files together this value is only 1.9%. This behavior is orthogonal to user stereotypes. Hence, upon a file edition operation, our workload generator picks an existing file belonging to a given file category; the file category is chosen in a fitness-proportionate manner based on its edition probability.

Another important aspect of a file update is the *size of the updated content*. In this sense, Fig. 3.7 shows the relative size of updates for each file category. In particular, the relative update size is calculated as the ratio between the updated and original file sizes; a value < 1 means that the update subtracted content to the original file and the opposite represents addition of new content. As visible in Fig. 3.7, the sizes of updates are related to the type of activity executed on them; for instance, `Code` and `Compressed` files are recipient of small updates, whereas `Pics` and `Docs` tend to exhibit

[1] Via a negative log-likelihood test we found that the generalized Pareto distribution provides the best fits with the following estimated parameters: `Application/Binary` ($k = 2.1164$, $\sigma = 4.0991\text{e}{+003}$), `Audio/Video` ($k = 2.4727$, $\sigma = 1.7194\text{e}{+004}$), `Code` ($k = 0.8845$, $\sigma = 2.9578\text{e}{+003}$), `Compressed` ($k = 1.0657$, $\sigma = 5.1760\text{e}{+003}$), `Docs` ($k = 2.0787$, $\sigma = 1.5624\text{e}{+003}$) and `Pictures` ($k = 2.1686$, $\sigma = 2.6006\text{e}{+003}$). In all cases $\theta = -2.2204\text{e}{-015}$.

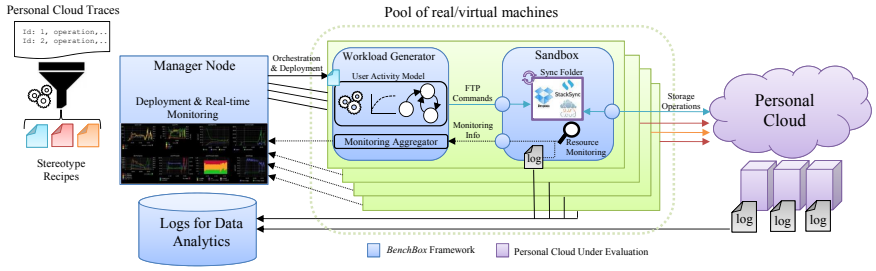

Figure 3.8: Architecture of `BenchBox`.

subtractions/additions of larger amounts of data. To reproduce the size of updates per file category, we fitted the distributions in Fig. 3.7 against a battery of distributions[1].

Finally, it is important to note that there are different types of file edits. According to the model Tarasov et al. [37], files can be updated at the begging (*prepend*), at the end (*append*) or anywhere in the middle. Naturally, these types of updates are performed by users with different probabilities. To this end, we incorporate in our model the probabilities file update types presented at [37].

3.5 BenchBox Design

`BenchBox` is a distributed performance evaluation tool tailored to exercise Personal Cloud desktop clients. `BenchBox` pays special attention to characterize and reproduce the behavior of users for generating client-centric workloads. Next, we show the architecture and operation of `BenchBox`.

3.5.1 Architecture

As visible in Fig. 3.8, the architecture of `BenchBox` has three differentiated components: *manager*, *workload generators* and *sandboxes*. First, the entity that coordinates the benchmarking process is the *manager node*. It enables a practitioner to deploy an arbitrary number of `BenchBox` *instances* that will generate the workload itself, as well as to monitor the activity of these instances in real-time. Moreover, the manager node is able to configure `BenchBox` instances to emulate different types of user behavior, providing controlled flexibility to the workload generation process.

`BenchBox` instances are normally formed by pairs of *workload generators* and *sandboxes* that run on individual *virtual machines*. Workload generators are processes that integrate a user stereotype workload model; as the one we proposed in sections 3.3 and 3.4. These processes take as input an *stereotype recipe* file for generating storage load driven by a particular user behavior. By storage load, we refer to the synthetic files and folders created, as well as the operations performed on them. Workload gen-

[1] Via a negative log-likelihood test we found that the t-location scale distribution provides the best fits with the following estimated parameters: `Application/Binary` ($\mu = 1.0029$, $\sigma = 0.0085$, $\nu = 0.2593$), `Audio/Video` ($\mu = 1.0023$, $\sigma = 7.7649\text{e}{-004}$, $\nu = 0.2737$), `Code` ($\mu = 1.0007$, $\sigma = 0.0026$, $\nu = 0.4424$), `Compressed` ($\mu = 1.0001$, $\sigma = 7.6166\text{e}{-004}$, $\nu = 0.1575$), `Docs` ($\mu = 1.0006$, $\sigma = 0.0011$, $\nu = 0.5574$) and `Pictures` ($\mu = 0.9999$, $\sigma = 0.0116$, $\nu = 0.3808$).

erators translate the workload into FTP commands that are sent via a virtual interface to the sandbox virtual machine.

On the other hand, sandboxes are intended to run Personal Cloud desktop clients in an *isolated environment* for the sake of monitoring accuracy. That is, a set of monitoring tools capture the activity of a Personal Cloud desktop client (CPU, disk, memory and network), in order to track its behavior and detect potential inefficiencies. Similarly to [6], all the network activity that desktop clients produce is monitorized in a primary virtual network interface, whereas a secondary one is used for sending storage commands from the workload generator.

Finally, the sandbox also resorts to the secondary virtual network interface to send monitoring information. In particular, the sandbox sends both real-time monitoring information to the manager node, as well as the experiment logs of each `BenchBox` instance to a data analytics repository for future analysis.

3.5.2 Deployment Lifecycle

Deploying a performance analysis with `BenchBox` consists of the following phases: *Configuration*, *deployment*, *warm-up*, *execution*, *data collection*, and *tear-down*.

In the configuration phase, a practitioner defines at the manager node essential parameters of the experiment, such as the number of `BenchBox` instances to be deployed, the stereotype that each instance belongs to, or the experiment length, to name a few. Once these parameters have been set, the deployment phase distributes `BenchBox` instances across the available machines to execute the experiment.

Once the virtual machines are deployed, `BenchBox` permits to execute a warm-up phase. The objective during this period is to generate a certain amount of storage load to exercise the Personal Cloud prior the actual workload generation (execution phase). For instance, during the warm-up phase `BenchBox` currently creates the directory structure of nodes, to further populate it with realistic contents.

During the execution phase, `BenchBox` instances generate the storage workload and monitor sandbox resources consumed by desktop clients. When the execution phase ends, the manager node orchestrates the `BenchBox` instances to store the collected logs into a data analytics repository, namely, data collection phase. Once the data collection phase ends, the manager node starts the tear-down phase that gracefully terminates the execution of `BenchBox` instances.

3.5.3 Extending `BenchBox`: Models & Recipes

User stereotype recipes are files that capture the necessary parameters of user behavior to feed a workload generator process. Therefore, if a designer aims at introducing a new user model in `BenchBox`, he or she should also consider the necessary metrics to be calculated of the different groups of users. Naturally, there is an interesting trade-off between the recipe complexity and the generation model's accuracy.

Apart from its flexibility, a remarkable advantage of generating user stereotypes recipes from a system's workload is that back-end traces remain private. That is, in `BenchBox` what practitioners share are the recipes that capture the behavior of users and the workload generation model, not the traces themselves [38]. Moreover, stereotype recipes may be very compact and easy to share; for instance, none of the recipes

built from our stereotypes exceeds 7KB in size. This new form of *user behavior sharing* may encourage today's Personal Clouds to contribute user stereotype recipes from private traces for research purposes.

4

Evaluating Personal Clouds with BenchBox

In this Chapter, we aim at launching an extensive battery of experiments with BenchBox to evaluate Personal Cloud services. To this end, we exercise Personal Cloud desktop clients with user-driven workloads guided by the user stereotype recipes obtained from the UbuntuOne trace (see Appendix B).

4.1 Setup

Prototype. The source code of BenchBox consists of 7K LoC for the workload model (Python) and over 3K LoC for the management dashboard (HTML, JavaScript). To automate the deployment of BenchBox instances on nodes with the required software libraries, we resort to vagrant[1]/puppet[2]. BenchBox sandbox instances send monitoring information to InfluxDB[3] via RabbitMQ for further analysis and for real-time monitoring inspection (Grafana[4]). The source code of BenchBox is publicly available[5]. For more information, please refer to Appendix A.

Scenario. We execute BenchBox against public Personal Clouds as they are are the most popular personal storage services among users. In particular, in our experiments we make use of Dropbox (client version 8.4.20) and Google Drive (client version 1.31.2873.2758).

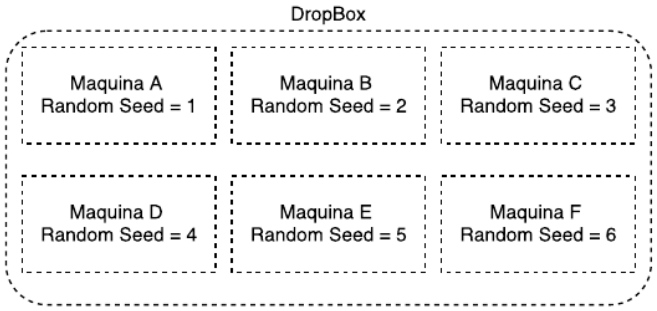

Figure 4.1: Experiment rounds sample

To exercise these clients, we executed several BenchBox instances that generate the desired workload within the Personal Cloud sync folder. Each experiment lasts around 4 hours and consists of 6 machines. As visible in Fig. 4.1, each BenchBox

[1] https://www.vagrantup.com/
[2] https://puppet.com/
[3] https://influxdata.com/
[4] http://grafana.org/
[5] https://github.com/cloudspaces/benchbox

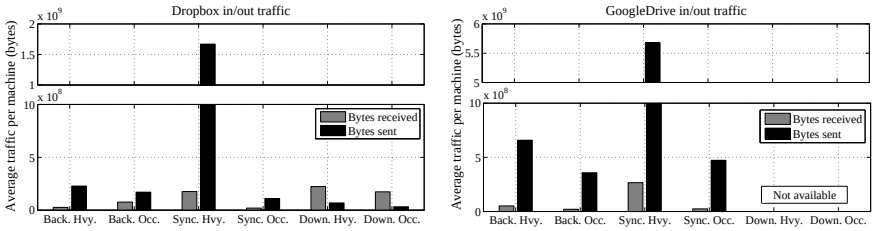

Figure 4.2: Average upload/download traffic per machine depending on the stereotype/provider.

instance executed in a machine is initialized with a unique random seed within the group to introduce some behavior variance at each stereotype workload. However, seeds are consistent across groups of desktop clients for different experiments, thus ensuring that we exercise different Personal Clouds with the same workloads.

In particular, to emulate a file download from a client viewpoint, we resorted to the REST APIs provided by these services to enable simplified data management in user accounts [39] (PUT/GET). That is, in the event of a client download, the `BenchBox` workload generator uploads a file via the REST API to the Personal Cloud account under test, which makes the desktop client to trigger a file download. However, due the important limitations of Google Drive REST API, we could not generate download oriented workloads in this provider (*Download Heavy* and *Download Occasional* stereotypes). Such limitation may be due to prevent service abuses, such the storage leeching problem [50].

In our evaluation, we aim at understanding the impact of different types of user behavior on desktop clients. Therefore, we account for metrics that are heavily influenced by such behavior (e.g., CPU, network traffic, hard disk, memory). Most of these metrics have not been reported before, so our evaluation complements existing measurements of Personal Cloud desktop clients [6, 7].

Regarding sharing groups, we have reported that disparate sharing topologies exist within the same system, specially in terms of connectivity or clustering [48]. To address this aspect, we built two sharing groups that present totally opposite topologies: a *Clustered* group (all users are linked by a common shared folder) and a *Non-clustered* group (all nodes share individual folders with a hub node).

Platform. We execute `BenchBox` and the Personal Cloud desktop clients in machines that incorporate an i5 processor, 4GB DDR2 of RAM and 500GB of hard-disk storage running a Debian 3.2 distribution. The execution of desktop clients is performed in VM with Windows 7 Enterprise (64 bits) as operating system (`BenchBox` sandbox).

4.2 Impact of User Behavior on Provider Resources

In this section, we analyze the impact of user behavior on desktop clients for both Dropbox and Google Drive Personal Clouds. In particular, we aim at understanding how different types of user stereotypes are related to the consumption of valuable cloud resources, such as in/out bandwidth and storage.

Fig. 4.2 depicts the average amount of in/out traffic per machine, depending on

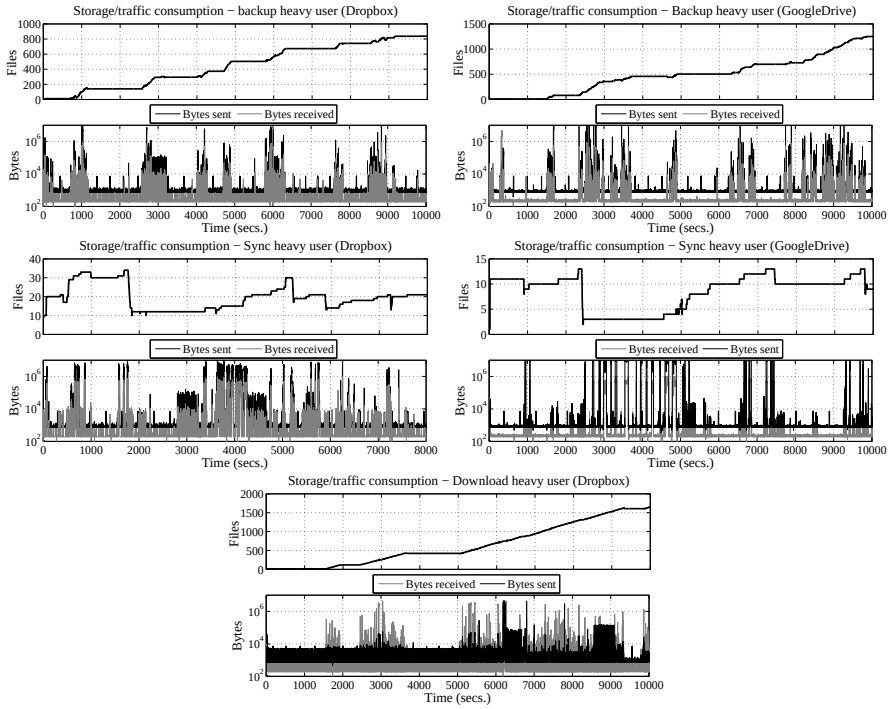

Figure 4.3: Relationship between storage and traffic consumption for individual stereotype users.

the provider and the stereotype under test. As expected, for both providers the stereotypes that consume more upload traffic (bytes sent) are those more focused on storage, such as *Backup-** stereotype. Conversely, we observe that the desktop clients exercised with a *Download-** stereotype exhibited a much higher rate of incoming traffic (bytes received).

Surprisingly, we also note that in both Dropbox and Google Drive the *Sync-Heavy* stereotype exhibits the highest average outgoing traffic values per machine. In fact, for *Sync-Heavy* users this value is x7.31 and x8.62 times higher than for *Backup-Heavy* users in Dropbox and Google Drive, respectively. This suggests that file updates are specially complex operations to handle by desktop clients, involving high amount of metadata communication with the server (other systems have also reported efficiency problems under file updates [2]).

If we compare both providers, we observe that in most cases Dropbox desktop clients consume less traffic to handle the same workload that Google Drive ones. This is specially evident for upload or outgoing traffic. This demonstrates that the number of data management techniques embedded into Dropbox clients [6, 7, 8] reports significant traffic savings for Dropbox, for a wide range of user activity. Therefore, Dropbox saves up costs in traffic by delegating optimization tasks to desktop clients.

Fig. 4.3 depicts the evolution of a random user's files of each **-Heavy* stereotype, and the repercussions on network traffic. In this sense, we observe that the most storage intensive users are *Backup-Heavy* and *Download-Heavy*; in particular, Fig. 4.3 shows that at the end of the experiment users belonging to these stereotypes store more than 1K files. For both stereotypes, we observe a clear correlation between the increase of the number of files and the spikes of network traffic. Naturally, in the case of the

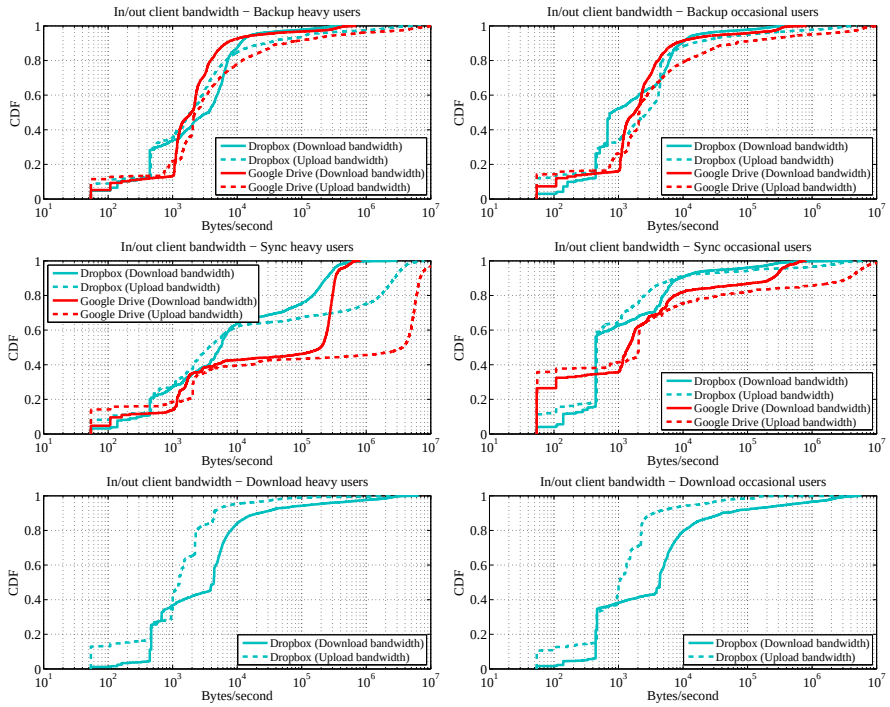

Figure 4.4: Aggregated upload/download bandwidth distribution of desktop clients depending on the stereotype/Personal Cloud.

Backup-Heavy user outgoing traffic dominates as files are created by the user itself; on the other hand, the *Download-Heavy* user exhibits higher spikes of incoming traffic as files are downloaded from the cloud.

Conversely, *Sync-Heavy* users only generate a small set of files in both Dropbox and Google Drive, indicating that this type of users are not intensive in the consumption of storage capacity. Overall, in line with our previous observations, the network traffic generated generated by these users is the highest. Moreover, the traffic generated by these users is not correlated with a growth in the number of files; to wit, one can observe important traffic spikes while the number of files remain constant. Clearly, these spikes are due a high number of update operation on existing files.

In conclusion, we can state that the different stereotypes tested are more demanding at different types of cloud resources; this yields that the data management techniques that are effective for some users, may be of little help for other ones. Therefore, these observations may motivate researchers and Personal Cloud providers to investigate intelligent and adaptive data reduction techniques in desktop clients based on the type of activity of users.

Fig. 4.4 depicts upload/download bandwidth of Dropbox and Google Drive desktop clients for our user stereotypes. More precisely, Fig. 4.4 draws a Cumulative Distribution Function (CDF) of all bandwidth values gathered at all desktop clients per stereotype/provider case.

At first glance, we see that the highest bandwidth values belong to the most used type of traffic (e.g., upload bandwidth for *Backup* users). This observation applies for all stereotypes and Personal Cloud providers. In particular, *Sync Heavy* users are the ones exhibiting highest bandwidth values, specially for upload traffic. Thus, transfer-

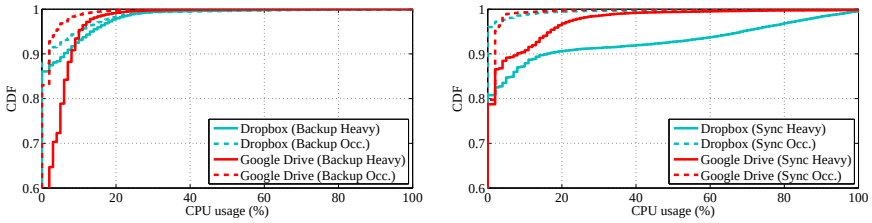

Figure 4.5: CPU consumption CDF of Dropbox and Google Drive for *Backup* and *Sync* users.

ring more data to the cloud —either upload or download— enables desktop clients to achieve higher transfer speed (e.g., TCP connections can be better utilized). Such observation is in line with prior measurements of Personal Clouds that reported higher transfer speed rates for larger files [39].

In this sense, we also observe that Google Drive tends to exhibit higher bandwidth values than Dropbox. For instance, the 90 percentile of upload bandwidth for *Backup Heavy* users is 28.21KBps and 66.90KBps for Dropbox and Google Drive, respectively. For *Sync Heavy* users, the 90 percentile of download bandwidth is 240.87KBps and 333.04KBps for Dropbox and Google Drive. In our view, the fact that a relatively new Personal Cloud player such as Google Drive is already delivering higher bandwidth performance than Dropbox on free accounts gives a sense on the intense competition among providers in this market.

4.3 Resource Usage of Desktop Clients on User's Machines

In this battery of experiments, we are going to analyze a quite unexplored facet of these services: The resources that desktop clients consume on user machines. In particular, we focus on the CPU, RAM and disk usage of Personal Cloud desktop clients depending on the stereotype at hand. We believe that these metrics are particularly important as they may impact on the quality of experience of users [41].

Fig. 4.5 shows the CDF of CPU consumption for users of various stereotypes in Dropbox and Google Drive. Naturally, as appreciable in Fig. 4.5 **-Heavy* users exhibit a much higher CPU consumption that **-Occasional* users in both providers. Such consumption is particularly intense for *Sync-Heavy* users, which can reach a CPU utilization of 70% for the 5% of the experiment time in the case of Dropbox. In our view, this means that frequent file updates in desktop clients involve high CPU consumption that may impact on the execution of other applications within a user's machine.

If we compare Dropbox and Google Drive in Fig. 4.5, we observe different behaviors regarding the CPU consumption. On the one and, Google Drive desktop clients tend to use the CPU a higher amount of time; that is, these clients use the CPU 45% of the time under *Backup-Heavy* workloads, whereas for Dropbox ones this value is only 14%. On the other hand, Dropbox exhibits a more aggressive CPU usage pattern; for instance, the 99 percentile of CPU usage for *Backup-Heavy* workloads is 25.6% and 19% for Dropbox and Google Drive, respectively. A similar situation occurs for *Backup-Occasional* users.

This emphasizes our proposal of adapting data management techniques embed-

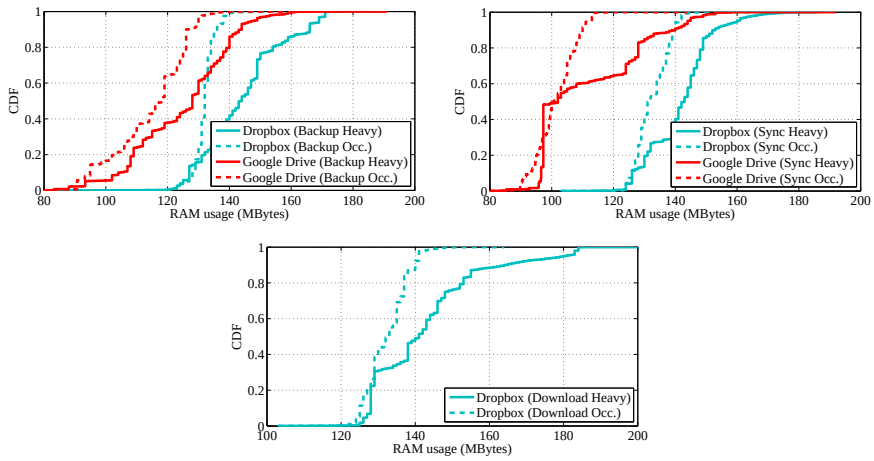

Figure 4.6: RAM usage CDF of Dropbox and Google Drive desktop clients for all the stereotypes.

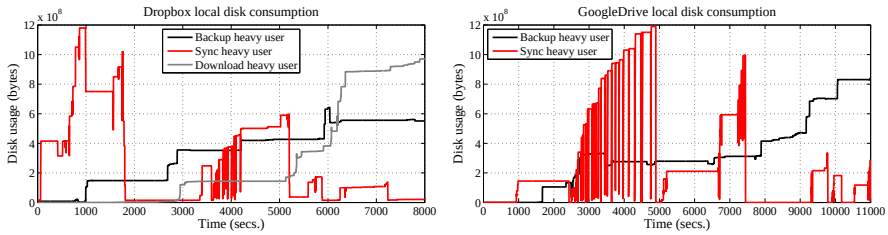

Figure 4.7: Time-series evolution of the size of sync folders exhibited by individual stereotype users in Dropbox and Google Drive.

ded into desktop clients to the type of user activity. By doing this, designers can reduce unnecessary data processing, making desktop clients less resource consuming.

Fig. 4.6 depicts the distribution of RAM usage of desktop clients depending on the user stereotype and the Personal Cloud selected. This figure is consistent with our previous observations on CPU consumption; the intensity of user activity is an important aspect to model in a workload, as **-Heavy* users exhibit a much higher RAM usage than **-Occasional* users in both Personal Clouds. In general, we see that the RAM usage distributions of **-Heavy* and **-Occasional* are similar for all stereotypes. Perhaps, *Download-Heavy* users exhibit the highest amount of consumed RAM, which suggests that Dropbox desktop clients make significant use of memory for reconstructing data chunks of downloaded files.

Moreover, Fig. 4.6 also shows that Dropbox tends to consume a higher amount of memory than Google Drive given the same stereotype. For instance, the median RAM usage for *Sync-Heavy* in Dropbox is 143MB, whereas for Google Drive this value is 102MB. This can indicate that the more complex data management implemented in Dropbox (see Fig. 2.6) makes the desktop client highly resource consuming also in terms of RAM.

The last metric that we analyze from a user's machine viewpoint is the usage of disk. That is, Fig. 4.7 shows the disk usage of individual users of each tested stereotypes in Dropbox and Google Drive. Concretely, the disk usage metric depicts the size

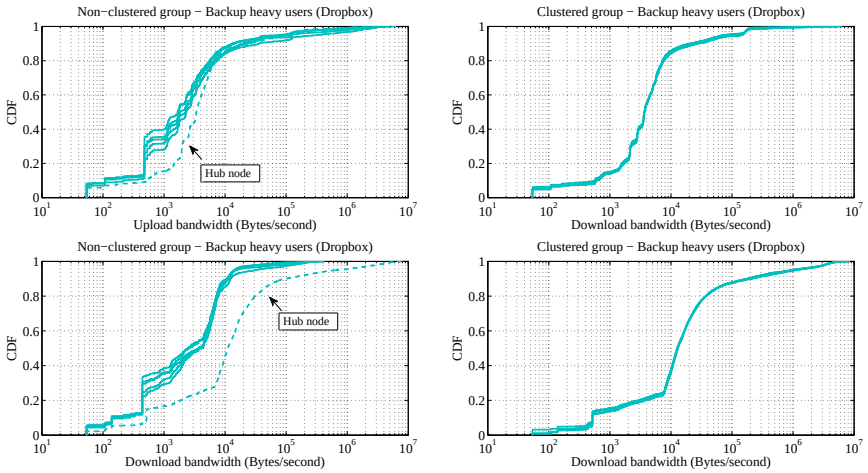

Figure 4.8: Upload/download bandwidth of individual users in Dropbox depending on the data sharing topology.

on disk of contents written to the Personal Cloud sync folder (user content or desktop client internal data). As expected, the usage patterns of *Backup* and *Download* users present an increasing utilization of disk along the experiments. This is natural, as both types of users tend to accumulate a large number of local files, either via upload or download operations (see Fig. 4.3).

However, it is very interesting to observe that *Sync* users tend to exhibit the highest disk utilization in some parts of the experiment, despite the fact that these users are not storage intensive. The explanation for the growth of the sync folder for *Sync* may be due file caching on file updates; that is, we detected that Dropbox desktop clients store recently modified or deleted files in a hidden folder within the sync folder (`//Dropbox//.dropbox.cache`). This is probably done to exploit temporal locality of file management and avoid extra communication with the server. As shown in Fig. 4.7, the consequence of this optimization is that under intensive file update workloads the cache can grow dramatically.

4.4 The Role of Sharing Group Topologies

In this section, we aim at analyzing the resources consumed by groups of Dropbox users sharing content. We are particularly interested in the role that the sharing topology —i.e., how users are interconnected via sharing folders— impacts on the consumption of cloud and local resources. To this end, we built two groups: A *Non-Clustered* group, in which a single *hub node* has shared folder with N other nodes; and a *Clustered* group, in which all the nodes in the group share a folder with the remaining $N-1$ ones. In fact, both types of topologies are common in real environments [48]. For this battery of experiments, we focused only on *Backup-Heavy* users.

Fig. 4.8 shows the distribution of bandwidth values for all nodes in both groups. As can be observed, regular nodes in the *Non-Clustered* group present lower upload and download bandwidth than the hub node. The reason for this is that the hub node supports a workload much higher than the rest of nodes in this topology, as the rest of nodes share data with it. This is specially evident for download bandwidth, as most

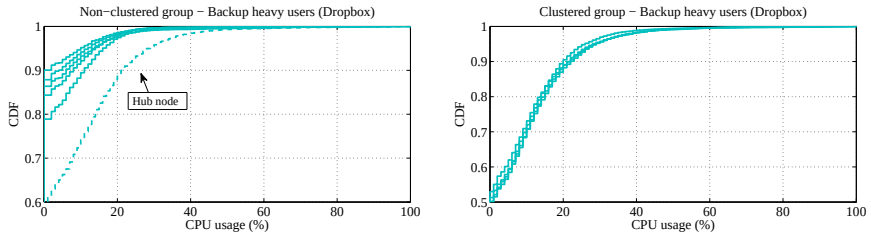

Figure 4.9: CPU consumption CDF of Dropbox desktop clients for *Backup-Heavy* users grouped in *Non-clustered*/*Clustered* topologies.

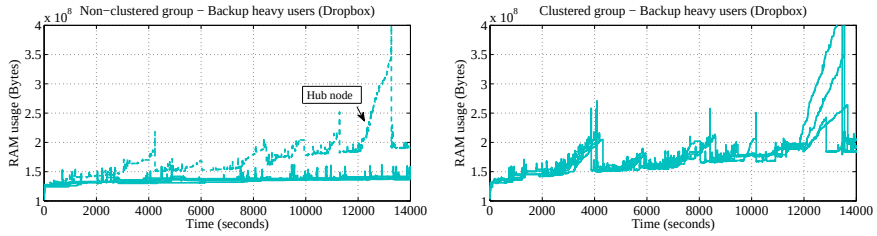

Figure 4.10: RAM usage time series of Dropbox desktop clients for *Backup-Heavy* users grouped in *Non-clustered*/*Clustered* topologies.

of the time the hub node should download data coming from the rest of nodes.

On the other hand, we can see that in the *Clustered* topology all the nodes exhibit similar bandwidth numbers to the hub node in the *Non-Clustered* group. In this case, all the nodes are supporting high load coming from the rest of participants, and they are also sending content to the rest of participants. Moreover, this makes highly clustered groups to be potentially much more bandwidth consuming from a provider's perspective than non-clustered ones. This may lead Personal Clouds to investigate novel content distribution techniques based on the topology of sharing groups.

Fig. 4.9 demonstrates that our observations on cloud bandwidth consumption are extensible to local machine resources. In this sense, the figure shows that for the *Non-Clustered* groups most of the nodes exhibit low to moderate CPU usage; that is, these desktop clients use the CPU between 10% to 20% of the time. However, we can observe that the desktop client at the hub node uses the CPU over 40% of the time. This is mainly due to the constant activity of processing shared files from other users.

On the other hand, we observe that all the nodes in the *Clustered* topology exhibit a CPU usage that is even higher than the hub node in the *Non-Clustered* one (desktop clients use the CPU 50% of the time). This indicates that the desktop client should work in parallel for both storing files to nodes and downloading them from different users, which inherently requires higher compute resources.

In terms of memory, Fig. 4.10 shows a time-series view of the memory consumption of desktop clients belonging to both network topologies. Similarly to what happens to the consumption of CPU, we observe that the hub node supports high memory usage in the *Non-Clustered* sharing group. In some cases, the spikes of memory consumption may reach more than 350MB, which is over 2.6 times more than the rest of machines in the group. This can significantly impact user experience for hub nodes, as the local machine has much less memory to accommodate other concurrent applications. Fig. 4.10 also shows a similar problem to the hub node exists across all

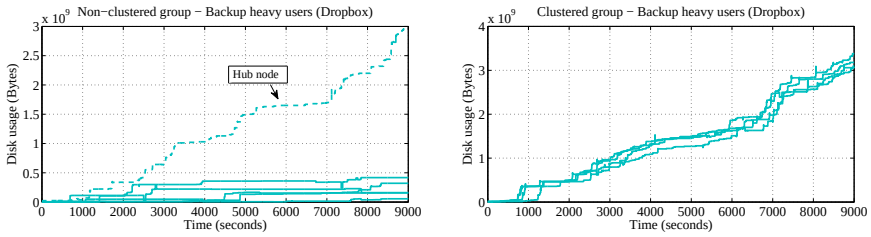

Figure 4.11: Disk usage time-series Dropbox desktop clients for *Backup-Heavy* users grouped in *Non-clustered*/*Clustered* topologies.

the nodes forming the *Clustered* topology, with some desktop clients consuming more than 400MB of RAM.

Finally, in Fig. 4.11 we focus on analyzing the disk usage of desktop clients for both topologies. Similarly to what we observed for CPU and RAM, we clearly see that the sharing activity of group members against the hub node also impacts on the size of its sync folder. That is, after 2.5 hours of experiment, the hub node recorded a sync folder size of 3GB; conversely, none of the rest of desktop clients stored more than 0.5GB. This yields that a hub node establishing several data sharing link with storage intensive users (e.g., *Backup-Heavy*) may rapidly limit its own local storage resources. This observation applies in the case of users connected in a *Clustered* topologies; that is, after 2.5 hours of experiments all the nodes exhibited a sync folder size $>$ 3GB.

4.5 Discussion of Experimental Results

Next, we aim at summarizing in a comprehensive way the main observations and conclusions that we draw from our experiments:

- **The behavior of users matter to the consumed cloud resources**: We observed that the different stereotypes modeled impact on distinct types of cloud resources. For instance, both *Backup* and *Download* are active consumers of cloud storage space via upload and download traffic, respectively. On the other hand, we pointed out that *Sync* users exhibit modest storage consumption, but they are the type of users that consume most upload bandwidth.

  Opportunity: Thanks to our benchmarking methodology, we can replay these types of users, and other that can be modeled in the future. This can be used by researchers and practitioners to design tailored data management techniques that best fit to each type of user activity.

- **Desktop clients consume significant local resources**: We observed that Personal Cloud desktop clients consume significant resources of user machines during their operation (i.e., CPU, disk, RAM). In this sense, a key aspect to take into account is the intensity of a user's activity (e.g., **-Heavy*, **-Occasional*); a higher intensity yields more resources consumed on data management tasks. The type of user activity is also relevant; to wit, in Dropbox *Sync-Heavy* consume the highest amount of CPU and disk resources, whereas *Download-Heavy* users tend to consume more RAM. Clearly, the lack of efficiency in the operation of desktop

clients may impact on the quality of experience of users, as less resources are left for the execution of other applications.

Opportunity: `BenchBox` provides researchers and practitioners with golden opportunities of monitoring and analyzing the resources consumed by desktop clients under different types of user-driven workloads. This may help to improve the efficiency of Personal Cloud desktop clients.

- **No single Personal Cloud winner**: Comparing Dropbox and Google Drive we found that there is no clear winner in terms of resource consumption depending on the type user activity. For instance, Dropbox generates less traffic among desktop clients handling the same workload than Google Drive. However, Google Drive utilizes less RAM and exhibits a less aggressive CPU usage than Dropbox.

  Opportunity: `BenchBox` provides a solid ground to evaluate different Personal Cloud provider under realistic and reproducible conditions.

- **Data sharing topologies are important**: Apart from the individual activity of a user with his own desktop client, we observed that the global activity of users within a sharing group is very important to understand the behavior of these systems. On the one hand, we found that in *Non-Clustered* topologies the hub node is highly overloaded compared to the rest of group members; on the other hand, *Clustered* groups present a more homogeneous but very high resource utilization. In our view, clustered sharing groups are specially important to handle by Personal Cloud providers, as they may represent an important source of wasted cloud and local resources.

  Opportunity: `BenchBox` supports the execution of groups of users to analyze the repercussions of their topology on the consumption of cloud and local resources. This may be exploited by researchers to devise new data management techniques that also consider groups of users, instead of only individual desktop clients (e.g., peer-assisted content distribution [51]).

5

Conclusions and Future Work

In this work, we presented a novel client-centric benchmarking methodology based on identifying and modeling groups of users with similar local storage behavior — *user stereotypes*— for generating user-driven storage workloads. To this end, we first identified archetypal user behaviors in real traces (UbuntuOne). Then, we designed a practical workload model that reproduces both the activity and storage contents of users in sync folders. We instantiated such methodology in a performance evaluation tool (`BenchBox`) to exercise several Personal Clouds.

Our experiments showed that the different behaviors of users significantly impact on the consumption of both provider and local machine resources. Furthermore, given the different combinations data management techniques embedded at desktop clients, we found that the efficiency of Personal Clouds varies depending on the type of user activity supported. This proves that `BenchBox` can help researchers and practitioners to optimize Personal Clouds via realistic user-driven workloads. We foresee that `BenchBox` can be used to evaluate many other fat client storage systems in which user behavior plays a key role.

Future Research Lines

The ideas and contributions behind this thesis enable us to devise new research lines that may follow the present work:

New applications of our benchmarking methodology: Despite being an attractive use-case, we believe that the evaluation of fat client systems with user-driven workloads may go far beyond Personal Clouds. For instance, our methodology may be applied to content distribution services with syncing capabilities, such as BittorentSync[1]. There also exist a myriad of contributory storage systems with fat client designs that heavily depend on the behavior of users, such as friend-to-friend, peer-to-peer and cooperative caching systems, among others [52, 53, 54]; these systems are a suitable target for `BenchBox` in order to evaluate their operation under realistic user-driven workloads. Similarly, Distributed Online Social Networks (DOSN)[2] [55] and Social Clouds [56] are built upon the storage resources of users, which are not only subject to their individual behavior but also to their social relationships [57]. As we demonstrated in this thesis, `BenchBox` allows to emulate workloads of user groups that are connected via storage relationships. Our benchmarking methodology perfectly fits this gap as it enables researchers to devise new workload models to reproduce the stereotypes that best represent users in such disparate systems.

[1] `https://getsync.com/`

[2] `https://joindiaspora.com/`

Evaluate on-premise providers: In this thesis, we presented an extensive evaluation of public Personal Cloud providers. However, it is increasingly common to find companies and organizations that resort to on-premise providers, such as `ownCloud`[1] or `StackSync` [5]. The advantage of such on-premise systems is that they enable to build a Personal Cloud service within the domains of an organization, thus keeping the data under control. We foresee to exercise on-premise Personal Clouds to evaluate their desktop clients, as well as the impact of local user behavior on the server side.

Devise new workload models: We proposed a practical workload model in this thesis to emulate the activity and contents of Personal Cloud users in short-term workloads. However, `BenchBox` enables to easily deploy and execute alternative workload models that capture other workload characteristics. For instance, one can easily infer that to reproduce long-term workloads (e.g., one or more days) in a realistic way it would be necessary to capture the non-stationary connectivity of users; that is, the connection/disconnection patterns of users during days and nights. In this sense, we also plan to work in the dissemination of the tool for encouraging researchers and practitioners using `BenchBox` as a tool to develop and share their own workload models.

[1] `https://owncloud.org/`

A

Implementation Notes of BenchBox

In the previous sections we have introduced BenchBox design plan and all the requirements of the framework. In this section we are going to have a detailed analysis of the implementation assertions with each component of BenchBox. During the implementation we had several issues and required several workarounds, until the current version.

Our initial idea with BenchBox was to be able to batch a set of hosts to run a certain personal cloud. At that time, we focused the development of BenchBox for StackSync the personal cloud developed within our research group. The source code of BenchBox can be found in [Github].

The Implementation is distributed into four modules, each responsible for a specific function: manager, workload generator, vagrant and the monitor.

A.1 BenchBox Manager Front-end

The front-end implementation is contained within the manager module, this is the core of the BenchBox framework, it is a server instance that delivers practitioners a user interface to setup and deploy experiments with BenchBox, it provides a real time dashboard to track the state of experiments powered with Grafana. This module is composed with several components and need to meet several requirements.

The module itself is contained within a vagrant project. It is provided within a virtual machine instance that has all the dependencies required, this way practitioners would not have to manage dependencies on their local machine.

Having the manager within a virtual machine allows practitioners to launch the manager from different platforms (Windows, Linux and Macintosh). They only need Vagrant and Virtualbox and a web browser to access the front-end interface. The manager instance has the following software requirements:

- **NodeJS**: we have used Node server framework for our front-end. We choose NodeJS because it is the framework that we were more familiar and its was light to install and deploy.
- **MongoDB**: it is the database used at the manager to hold information and configuration parameters of the front end, the location of the testing hosts, their credentials and session state.
- **RabbitMQ**: at the early stages of the BenchBox the communication between the manager and the testing hosts were stateless, this induced hard time to control the state flow of the experiments. Therefore we changed the design to route the messages between the hosts through a message queue middle-ware. With this

middle-ware we obtained better communication between the evaluation hosts. The middle-ware provide an event driven flow between the hosts communication, granting better scale-ability, asynchronous communication and fault tolerance because the framework automatically re-sends message that are not Acknowledged by the receiver.

- **ZeroRPC**: is a light-weight, reliable and language agnostic library for distributed communication between server side processes. It is build on top of ZeroMQ with message pack. We use ZeroRPC in the manager to build a reliable RPC engine, to bind the test machines with a permanent channel to the manager.
- **InfluxDB**: this database is used to store the statistics generated during experiments. The sandBox machine instances reports metrics related with the desktop clients. The benchBox machine instances reports metrics related with the operation performed. The statistics generated from the sandbox instances are forwarded to the manger through RabbitMQ exchange, handled by the NodeJS server, the server then stores the stats to the InfluxDB database.
- **Grafana**: is a framework with front-end for creating real time dashboard complemented with time series database. In this case we have combined Grafana with InfluxDB.
- **Vagrant**: we use this computer software to create and configure virtual development environments of our manager server instance within Virtualbox, this configuration management tool complemented with a provisioning library such as Puppet or raw bash scripts enables the automatic setup and deployment of the manager server instance.

How to launch the manager: with a console terminal in the project manager directory. We can iniciate the manager by launching, "vagrant up". With this command we are booting the manager Virtualbox instance, the specifications of the instance is described within Vagrantfile.

The bash code contained within script block variable tells the Virtualbox instance to start the manager server after the operating system is booted. The manager node comes along with 4096MB of RAM memory and 2 CPU core. The Virtualbox instance forwards several ports, so the manager service will be visible for local or corporate network. The port forwarding allows experimental machines to send metric messages to the manager. The manager instance is provided with all the dependent software pre-installed. The manager image can be found at atlas.hashicorp with the box name chenglong/manager.

A.2 BenchBox Instances Vagrant or Windows

The Vagrant or Windows modules are Vagrant projects, each of the module contains a Vagrant file, required to setup the testing environment at each experimental host. It consist to load two virtual machines: sandBox and benchBox. The Vagrant module defines the provisioning of Linux version of sandBox while the Windows module defines the provisioning of Windows version of testing environment, the two allows

```
$script = <<SCRIPT
echo "I am provisioning... Manager node"
echo `pwd`
date > /etc/vagrant_provisioned_at
echo "Start manager services"
# nohup /bin/bash /vagrant/manager.start.sh  &> /vagrant/nohup_manager.out&
/vagrant/manager.start.sh
echo "[http://localhost:8888/load#/l"
SCRIPT
box_memory = "4096"
box_cpu = "2"
Vagrant.configure(2) do |config|
    # The most common configuration options are documented and commented below.
    # For a complete reference, please see the online documentation at
    # https://docs.vagrantup.com.
    # Every Vagrant development environment requires a box. You can search for
    # boxes at https://atlas.hashicorp.com/search.
    config.vm.box = "chenglong/manager"
    config.vm.hostname = "manager"
    config.vm.network "forwarded_port", guest: 8888, host: 8888 # manager web interface
    config.vm.network "forwarded_port", guest: 4242, host: 4242 # zerorpc server
    config.vm.network "forwarded_port", guest: 5672, host: 5672 # rabbitmq
    config.vm.network "forwarded_port", guest: 15672, host: 15672 # webinterface rabbitmq
    config.vm.network "forwarded_port", guest: 3000, host: 3000 # grafana server
    config.vm.network "forwarded_port", guest: 8083, host: 18083 # webinterface influxdb
    config.vm.network "forwarded_port", guest: 8086, host: 8086 # listens in port 8083 (web) and 8086 (HTTP API).
    config.vm.network :forwarded_port, guest: 22, host: 12225, id: "ssh"
    config.vm.provider :virtualbox do |vm|
        vm.customize [
            "modifyvm", :id,
            "--memory", box_memory.to_s,
            "--cpus", box_cpu.to_s
            ]
    end
    config.vm.provision "shell", inline: $script
end
```

Figure A.1: This figure shows a sample vagrant file, based over our manager

targeting desktop clients under different platform. Both linux/windows share the same benchBox instance which loads the module workload generator.

- *sandBox*: The sandBox Virtualbox instance is defined within chenglong/sandbox hosted at atlas.hasicorp, this virtual machine comes with the python dependencies to run our target personal cloud desktop clients, and is provided with pre-installed desktop clients.
- *benchBox*: The benchBox virtual instance is defined within chenglong/benchbox hosted at atlas.hashicorp this virtual machine comes with dependencies for workload_generator module, with pre-installed python-dev, python-libraries and java to run jar applications.

A.3 BenchBox Monitor

The monitoring module holds the procedures that will be executed withing the vagrant sandBox machine, this create a windows instance with Virtualbox, the provisioning of sandBox contains the dependencies required for the monitoring module with our target desktop client installed. During an experiment the monitor module startup the target desktop client, and monitors the following metrics:

1. the network bandwidth
2. the shared folder files
3. the shared folder size
4. the CPU usage

5. the RAM usage

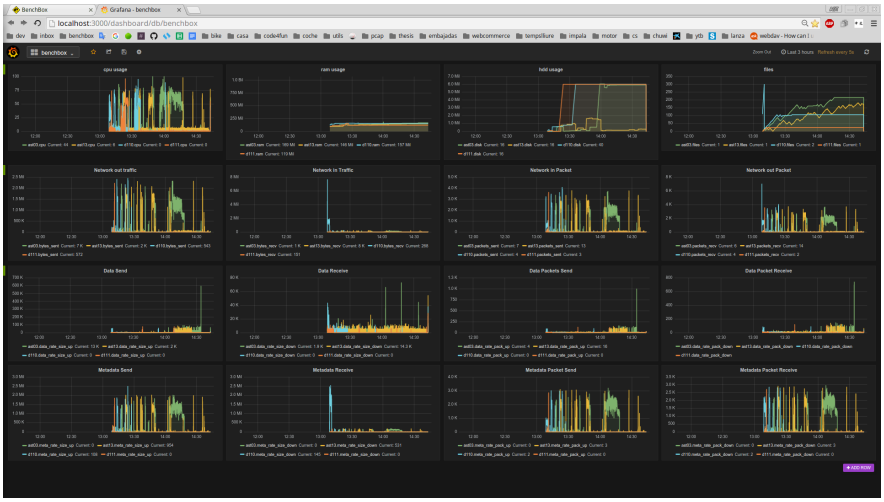

Figure A.2: This figure shows a sample dashboard during an experiment with the monitoring

The metrics are captured using the following libraries: the network bandwidth is analyzed at packet level using pcapy, this library allows live capture of all the network traffic from the testing host. During the capture, only network traffic through SSL (income or outgoing) are considered applying network filters mask to the packet capture process. The CPU consumption and the RAM consumption are obtained using python psutil library, this library allows metrics capturing by process. At last we analyze the folder size and count the amount of files within it. This task is performed with python library to count the amount of files contained within the shared folders (recursive os.path.getsize() of the shared folder content).

Each of the sandBox machine has a direct data channel with the manager instance using RabbitMQ message queue, the metrics tracked are forwarded from sandBox to the manager using message queue and later it is stored to InfluxDB database at the manager. Additional dependencies are the FTP-server installed at the sandBox that channels the workload-generator files to the personal cloud desktop client folder.

Inside the module there is monitor_rmq.py this is a RabbitMQ peer that handles events such as starting or stopping the monitoring procedures. This procedure is composed with py_sniffer and rmq_monitor, the first analyzes the network packets depending its target personal cloud, for instance each target will have a particular call to start the desktop client and filter the network packets, the second reports the metrics captured to the manager node through RabbitMQ.

A.4 BenchBox Workload Generator

The workload generator module contains applications and procedures necessary to generate workload in the form of file system operations. This module has the following external dependencies:

1. SDGen: allows synthetic data generation for storage benchmarks, this tool enables users to generate realistic data to feed storage bench-marking tools (github repo) https://github.com/iostackproject/SDGen

2. impressions: is a utility that allows realistic file-system generation [14].

The workload generator has the unique role of creating a virtual instance of the personal cloud shared folder content regarding the target stereotype recipe. In order to mirror the synthetic file-system, we use python ftp client paired with the sandBox ftp-server. With this binding each file-system operation is notified with a ftp operation mirroring both file systems.

Inside the module there is executor_rmq.py this is a RabbitMQ peer that handles events such as starting or stopping the workload-generator. This procedure is composed with communication utilities for ftp operations that replicate each personal cloud request execution (put, get, delete, move, update) with an ftp operation to the file-system. There is external utilities folder that contains the source code of impressions. it is called by the workload generator to create files. And at last there is py_publish folder, it reports the workload-generated operations to the manager node through RabbitMQ exchange. The remaining scripts in the module we can encounter sample user stereotype recipe in the user_stereotype directory, and a constats.py which contains constants to setup experiments variables such as the random seed which will directly inflict the pattern of the experiments and make the experiments repeatable.

A.5 Communication of BenchBox Components

1. *Communication from the manager with the experimental hosts*: The benchBox experiments starts, with the assignment of one manager node, this node contains a RabbitMQ message queue server that will listen to metrics requests and status notification requests. By metric requests we refer to sandBox or benchBox metrics generated during experiments, in the case of sandBox the metrics are related with the personal cloud desktop client that run during an experiment, the other case are the file-system operations that the benchBox instance reproduces during experiment. Beside the manager having a direct communication link between the sandBox and benchBox they also require an initial binding with the physical node itself prior the message queue binding. First the manager node connect with each of the physical experimental machines through Open-SSH with their credentials to bootstrap a RabbitMQ peer listener at each physical machine. Later through this RabbitMQ peer we can startup the sandBox and benchBox with vagrant from the manager interface. Once the virtual instances are up provisioned with a RabbitMQ peer listener, unlike the physical peer listener these has the role to boot the workload_generator procedure and the monitoring procedure within the sandBox and benchBox virtual instances.

2. *Communication from sandBox with the benchBox virtual instances*: These virtual instances are complementary pair peers that has only one requisite, that is to have the virtual file-system generated at the benchBox instance mirrored into the desktop client sync folder of the sandBox. This is accomplished through ftp-server hosted at the sandBox and ftp-client connecting from benchBox.

3. *Communication from the sandBox and benchBox to the manager*: The sandBox and benchBox instances have a peer connection to the manager RabbitMQ server, with this setup both can easily forward the metrics generated to the manager node. The manager consumes these metrics as messages from its metrics exchange, afterward it stores the trace to its local InfluxDB database.

4. *Communication from the desktop client to its cloud container*

   During the experiments, the workload_geneartor within benchBox machines may perform upload operations to its container so the file generated is not directly forwarded through ftp server but instead its download by the desktop client from its cloud container. This allows us to generate download workload against the desktop client under test.

5. *Communication from RabbitMQ message to InfluxDB*: When the manager has to handle a metric exchange from RabbitMQ server these are intercepted by a listener running under NodeJS that analyze the content of the metrics and writes the metrics to the InfluxDB database based on its exchange its stored as benchbox trace for monitoring metrics or benchbox_workload trace for the workload_generator operations.

The code of our tool can be accessed at: `https://github.com/cloudspaces/BenchBox/`.

B

Analysis of UbuntuOne Trace

In this appendix, we describe the technology used to analysis the Ubuntu One Traces until the obtaining the Stereotype recipe.

B.1 The dataset

This section will describe the analysis performed to the Ubuntu one traces. This traces were released by Canonical for research, this contribution was done after they shut-down their service[1]. It contains meta-data traces of UbuntuOne during one month of user activity, consisting 3.304.920.868 lines of meta-data operations. An more technical definition of the traces can be found in [2]. In the following, we depict the details are helpful to understand the queries performed to generate the distributions of the stereotype recipes.

B.2 Analysis tools

Cloudera Impala

Cloudera Impala is an open source massively parallel processing SQL query engine for data stored in a computer cluster running Apache Hadoop. We have configured a cluster of 6 hosts in our lab with HDFS partitions to host the data-set to analyze.

B.3 Trace Analysis

This section will explain the procedure and methodology followed to execute the queries over Cloudera and the data filtering applied to the traces, until the obtaining stereotype recipe samples.

Each of the properties from the stereotype recipe and the procedure followed to obtain them are explained as follow:

Operation chain, the operation chain determines the next operation state for the workload_generator during experiment. The current and next operation can be grouped by session states or operation state, by session state we find START and IDLE which belongs to the starting and ending point of a user session in real traces. Operation state are the possible operations that we simulate from the traces. These operation

[1] `http://cloudspaces.eu/results/datasets`

state can be MOVE, SYNC, UPLOAD, DOWNLOAD or DELETE operations to te file-system. The distributions from the recipe are obtained analyzing the successive user operations given an initial state during an active session.

Initial state probability this property determines the probability which the initial state is remained idle or it starts with an operation. This probability is obtained analyzing the distribution of sessions that have operations and sessions without any active operation. This property determines the probability that a session will start with operations or it will remain idle during whole session interval.

Stereotype file type probability from the traces we have categorized the file types into the following six groups: AppBinary, AudioVideo, Pict, Code, Docs and Compressed.

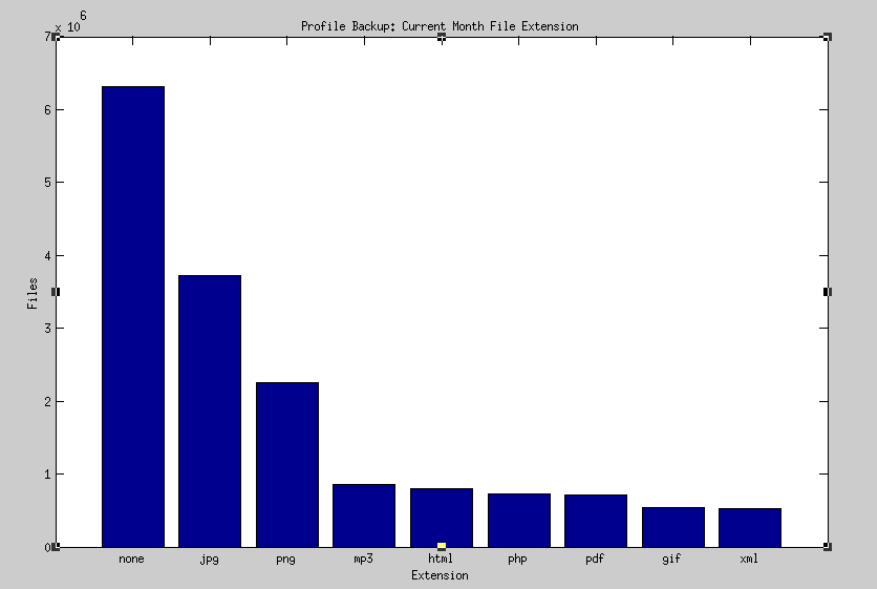

Figure B.1: This displays the most popular file extensions from backup user during the whole month of the UbuntuOne traces.

This set of file category is decided from a related work, to built the distribution of probability we have counted the amount of file of each category from the traces for each target stereotype user group, See figure B.1.

Stereotype file types extensions,

To generalize file type with similar properties when we create files characterizations with SDGen. we have selected the following file properties for each category.

- AppBinary:'.o', '.msf', '.jar', '.dat', '.ini', '.dll', '.log', '.mo', '.lock', '.npy', '.exe'
- AudioVideo: '.mp3', '.ogg', '.wav', '.au'
- Pict:'.jpeg','.png','.gif', '.svg', '.tif', '.bmp'
- Code:'.php', '.html', '.js', '.xml', '.h', '.c', '.java', '.py', '.css', '.htm', '.cpp', '.r', '.hpp', '.json', '.d', '.m'
- Docs:'.pdf', '.txt', '.tmp', '.doc', '.odt', '.docx', '.xls', '.csv', '.tex', '.po'

- Compressed: '.gz', '.zip'

This categorization of group of file extensions involves a generalization of the files we create during the experiments. More variety of characterization may raise the precision of the file properties, but for simplicity we consider the previous category with those extensions so we follow the scheme of categories from a related work.

File types sizes, the file size is calculated by file category, for each category we have grouped its file size (with their extensions) and build a file size array, which is later processed to obtain the best fitting distribution function to generate the file size from the given category. The fitting function is obtained using Matlab which computes the best fitting distribution function for each of the category considered. The script used can be found at the source inner the workload generator module as allfitdist.m

File update location probabilities, this distribution is obtained from related work, currently we use random file edit locations when we perform a SYNC operation.

Directory count distribution, this distribution is obtained by counting the amount of files and folder owned by each group of users.

File level deduplication, This property used durng the file creation, it influece the compression ratio of the generated file. This parameter is ontained from related work [1, 2]

References

[1] Idilio Drago, Marco Mellia, Maurizio M Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: understanding personal cloud storage services. In *ACM IMC'12*, pages 481–494, 2012.

[2] Raúl Gracia-Tinedo, Yongchao Tian, Josep Sampé, Hamza Harkous, John Lenton, Pedro García-López, Marc Sánchez-Artigas, and Marko Vukolic. Dissecting UbuntuOne: Autopsy of a global-scale personal cloud back-end. In *ACM IMC'15*, page In press, 2015.

[3] ZDNet. Dropbox hits half a billion users. `http://www.zdnet.com/article/dropbox-hits-half-a-billion-users/`, 2016.

[4] Zhenhua Li, Christo Wilson, Zhefu Jiang, Yao Liu, Ben Y Zhao, Cheng Jin, Zhi-Li Zhang, and Yafei Dai. Efficient batched synchronization in dropbox-like cloud storage services. In *ACM/IFIP/USENIX Middleware'13*, pages 307–327. 2013.

[5] Pedro García-López, Marc Sánchez-Artigas, Sergi Toda, Cristian Cotes, and John Lenton. Stacksync: Bringing elasticity to dropbox-like file synchronization. In *ACM Middleware'14*, pages 49–60, 2014.

[6] Idilio Drago, Enrico Bocchi, Marco Mellia, Herman Slatman, and Aiko Pras. Benchmarking personal cloud storage. In *ACM IMC'13*, pages 205–212, 2013.

[7] Zhenhua Li, Cheng Jin, Tianyin Xu, Christo Wilson, Yao Liu, Linsong Cheng, Yunhao Liu, Yafei Dai, and Zhi-Li Zhang. Towards network-level efficiency for cloud storage services. In *ACM IMC'14*, pages 115–128, 2014.

[8] E. Bocchi, I. Drago, and M. Mellia. Personal cloud storage benchmarks and comparison. *IEEE Transactions on Cloud Computing*, 2015.

[9] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P Wright. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage*, 4(2):5, 2008.

[10] Scale-out file sync & share with red hat storage and owncloud. https://owncloud.com/wp-content/uploads/RHS-ownCloud-Performance.pdf, 2014.

[11] William D Norcott and Don Capps. IOzone filesystem benchmark. http://www.iozone.org.

[12] Fio. http://freecode.com/projects/fio, 2005.

[13] Filebench. http://sourceforge.net/projects/filebench/, 2008.

[14] Nitin Agrawal, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. In *USENIX FAST '09*, pages 125–138, 2009.

[15] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In *ACM SoCC'10*, pages 143–154, 2010.

[16] Timothy G Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. LinkBench: a database benchmark based on the facebook social graph. In *ACM SIGMOD'13*, pages 1185–1196, 2013.

[17] Glauber D Gonçalves, Idilio Drago, Alex B Vieira, Ana Paula Couto da Silva, Jussara M. Almeida, and Marco Mellia. Workload models and performance evaluation of cloud storage services. *Computer Networks*, 2016.

[18] Raúl Gracia-Tinedo. *On Personal Storage Systems: Architecture and Design Considerations*. PhD thesis, Departament d'Enginyeria Informàtica i Matemàtiques (Universitat Rovira i Virgili), 2015.

[19] Dror G Feitelson. *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[20] Vasily Tarasov, Saumitra Bhanage, Erez Zadok, and Margo Seltzer. Benchmarking file system benchmarking: It *IS* rocket science. *USENIX HotOS'11*, 2011.

[21] Eric Anderson, Mahesh Kallahalla, Mustafa Uysal, and Ram Swaminathan. Buttress: A toolkit for flexible and high fidelity i/o benchmarking. In *USENIX FAST'04*, pages 4–4, 2004.

[22] Diwakar Krishnamurthy, Jerome Rolia, Shikharesh Majumdar, et al. A synthetic workload generation technique for stress testing session-based systems. *IEEE Transactions on Software Engineering*, 32(11):868–882, 2006.

[23] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP^2Bench: a SPARQL performance benchmark. In *IEEE ICDE'09*, pages 222–233, 2009.

[24] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: comparing public cloud providers. In *ACM IMC'10*, pages 1–14, 2010.

[25] Bonnie++. http://www.coker.com.au/bonnie++/, 2001.

[26] Qing Zheng, Haopeng Chen, Yaguang Wang, Jiangang Duan, and Zhiteng Huang. COSBench: A benchmark tool for cloud object storage services. In *IEEE CLOUD'12*, pages 998–999, 2012.

[27] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. OLTP-Bench: An extensible testbed for benchmarking relational databases. *VLDB Endowment*, 7(4), 2013.

[28] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent Zhan, Xiaona Li, and Bizhu Qiu. BigDataBench: A big data benchmark suite from internet services. In *IEEE HPCA'14*, pages 488–499, 2014.

[29] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 26(1):151–160, 1998.

[30] Vasily Tarasov, Santhosh Kumar, Jack Ma, Dean Hildebrand, Anna Povzner, Geoff Kuenning, and Erez Zadok. Extracting flexible, replayable models from large block traces. In *USENIX FAST'12*, page 22, 2012.

[31] Jan Beran, Robert Sherman, Murad S Taqqu, and Walter Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 43(2/3/4):1566–1579, 1995.

[32] Martin F Arlitt and Carey L Williamson. Web server workload characterization: The search for invariants. In *ACM SIGMETRICS Performance Evaluation Review*, volume 24, pages 126–137, 1996.

[33] Mark E Crovella and Azer Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[34] Ye Jin, Mingliang Liu, Xiaosong Ma, Qing Liu, Jeremy Logan, Norbert Podhorszki, Jong Youl Choi, and Scott Klasky. Combining phase identification and statistic modeling for automated parallel benchmark generation. In *ACM SIGMETRICS'15*, page In press, 2015.

[35] Wojciech Golab, M Rizwanur Rahman, Alvin AuYoung, Kimberly Keeton, and Indarchand Gupta. Client-centric benchmarking of eventual consistency for cloud storage systems. In *IEEE ICDCS'14*, pages 493–502, 2014.

[36] Margo Seltzer, David Krinsky, Keith Smith, and Xiaolan Zhang. The case for application-specific benchmarking. In *USENIX HotOS'99*, pages 102–107, 1999.

[37] Vasily Tarasov, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok. Generating realistic datasets for deduplication analysis. In *USENIX ATC'12*, pages 1–12, 2012.

[38] Raúl Gracia-Tinedo, Danny Harnik, Dalit Naor, Dmitry Sotnikov, Sivan Toledo, and Aviad Zuck. SDGen: Mimicking datasets for content generation in storage benchmarks. In *USENIX FAST'15*, pages 317–330, 2015.

[39] Raúl Gracia-Tinedo, Marc Sanchez Artigas, Adrian Moreno-Martinez, Cristian Cotes, and Pedro Garcia Lopez. Actively measuring personal cloud storage. In *IEEE CLOUD'13*, pages 301–308, 2013.

[40] W. Hu, T. Yang, and J.N. Matthews. The good, the bad and the ugly of consumer cloud storage. *ACM SIGOPS Operating Systems Review*, 44(3):110–115, 2010.

[41] Pedro Casas, Hans Ronald Fischer, Stefan Suette, and Roland Schatz. A first look at quality of experience in personal cloud storage services. In *IEEE ICC'13*, pages 733–737, 2013.

[42] Gil Goncalves, Idilio Drago, Ana Paula Couto da Silva, Alex Borges Vieira, and Jussara M Almeida. Modeling the dropbox client behavior. In *IEEE ICC'13*, pages 1332–1337, 2014.

[43] Thomas Mager, Ernst Biersack, and Pietro Michiardi. A measurement study of the wuala on-line storage service. In *IEEE P2P'12*, pages 237–248, 2012.

[44] Giancarlo Ruffo, Rossano Schifanella, Matteo Sereno, and Roberto Politi. Walty: A user behavior tailored tool for evaluating web application performance. In *Network Computing and Applications, 2004.(NCA 2004). Proceedings. Third IEEE International Symposium on*, pages 77–86. IEEE.

[45] Marcelo Maia, Jussara Almeida, and Virgílio Almeida. Identifying user behavior in online social networks. In *ACM SocialNets'08*, pages 1–6, 2008.

[46] Collins dictionary. http://www.collinsdictionary.com/dictionary/english/stereotyping.

[47] Danyel Fisher, Marc Smith, and Howard T Welser. You are who you talk to: Detecting roles in usenet newsgroups. In *IEEE HICSS'06*, volume 3, pages 59b–59b, 2006.

[48] Raúl Gracia-Tinedo, Pedro García-López, Alberto Gómez, and Anastasio Illana. Understanding data sharing in private personal clouds. In *IEEE CLOUD'16*, 2016.

[49] Franciszek Grabski. *Semi-Markov processes: Applications in system reliability and maintenance*. Elsevier, 2014.

[50] Raúl Gracia-Tinedo, Marc Sánchez Artigas, and Pedro García López. Cloud-as-a-gift: Effectively exploiting personal cloud free accounts via rest apis. In *IEEE 6th International Conference on Cloud Computing (CLOUD)*, pages 621–628, 2013.

[51] Rahma Chaabouni, Marc Sánchez-Artigas, and Pedro García-López. Reducing costs in the personal cloud: Is bittorrent a better bet? In *IEEE P2P'14*, pages 1–10, 2014.

[52] Jinyang Li and Frank Dabek. F2f: Reliable storage in open networks. In *IPTPS'06*, 2006.

[53] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201, 2001.

[54] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 202–215, 2001.

[55] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p so-

cial networking: early experiences and insights. In *SNS@EuroSys'09*, pages 46–52, 2009.

[56] Kyle Chard, Simon Caton, Omer Rana, and Kris Bubendorfer. Social cloud: Cloud computing in social networks. In *IEEE CLOUD'10*, pages 99–106, 2010.

[57] Raúl Gracia-Tinedo, Marc Sánchez-Artigas, Aleix Ramírez, Adrián Moreno-Martínez, Xavier León, and Pedro García-López. Giving form to social cloud storage through experimentation: Issues and insights. *Future Generation Computer Systems*, 40:1–16, 2014.