BenchBox: A User-driven Benchmarking Framework for Fat-client Storage Systems

Raúl Gracia-Tinedo, Chenglong Zou, Marc Sánchez-Artigas, *Member, IEEE* Pedro García-López, *Member, IEEE*

Abstract—In many online storage services, end-users mainly interact with the system via "fat" storage clients that integrate complex functionality. This means that to obtain a complete performance evaluation of one of such systems we may need to generate workloads on the client side that reproduce the behavior of real users. Unfortunately, this remains as an open research challenge today.
We present `BenchBox`: A distributed performance evaluation framework for fat-client storage systems. On the one hand, `BenchBox` can generate workloads directly in storage clients that mimic users exhibiting a certain behavior, namely, *user stereotypes*. To this end, the framework enables to plug-in *workload models* and feed them with compact recipes that capture the behavior of user stereotypes (e.g., storage activity, type of file contents, data sharing links). On the other hand, `BenchBox` provides researchers with *management and monitoring* facilities to deploy experiments and analyze the performance of groups of storage clients.
To demonstrate our framework, we equipped `BenchBox` with a *2-layer workload model* that reproduces both the *activity* —e.g., types of operations, frequency— and *data* —e.g., file sizes, data types— of users in a Personal Cloud. We used this model to generate workloads based on user stereotypes that we identified in real traces (UbuntuOne). Our experiments with public providers show how distinct types of users impact on the performance and efficiency of Personal Clouds, which may guide their optimization.

Index Terms—Performance evaluation, User modeling, Client-centric Benchmarking, Personal Clouds.

1 Introduction

Online data storage services are becoming massively adopted by end-users [61] and this trend is expected to grow in the next few years [25]. To face this challenge, the design of many storage systems —cloud, decentralized or hybrid— is based on a *fat-client architecture*. In these systems, a user interacts with a software client that communicates with the server side and/or other clients, but it still can execute functionalities without synchronous connections to any other system entity. Indeed, making clients "smarter" seems to be a promising strategy as users benefit from increasingly higher resources and a better infrastructure [26].

But even more importantly, offloading complex storage functionalities to fat clients offers significant *optimization and cost-reduction* opportunities to the storage service. For instance, backup services like CrashPlan integrate into software clients encryption and data reduction techniques upon file operations [17]. As another example, Personal Clouds like Dropbox enable to synchronize files directly across software clients within LAN networks [6], [29]. In a different scenario, decentralized storage systems in the literature (friend-to-friend [43], peer-to-peer [19], [51], or social clouds [15]) and commercial services such as Storj[1] implement complex distributed data management techniques (e.g., caching, redundancy) in their software clients. These techniques can mitigate the impact of having just a few peers to store data, or even mask their low availability within the system [36]. All in all, such different use-cases show how fat-client designs optimize the operation of online data storage services.

• *The authors are with the Department of Computer Engineering and Maths, Universitat Rovira i Virgili, Spain.*
Email: raul.gracia, chenglong.zou, marc.sanchez, pedro.garcia@urv.cat.

Manuscript received April 19, 2005; revised August 26, 2015.

1. https://storj.io/

1.1 Problem: Realistic Evaluation of Fat-client Systems

Fat clients introduce extra complexity when it comes to evaluating the performance of storage systems as a whole [59], which may lead to bad practices. To inform this argument, Red Hat has recently reported a performance evaluation of an ownCloud deployment, in which the system is claimed to support 25K users [5]. 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 individual *behavior of users*, not to mention the collective behavior of user groups. This poses doubts about the performance of the system in a real setting. Moreover, this evaluation was *exclusively server-oriented*; the behavior and resource consumption of desktop clients was omitted from the conclusions of this study, despite being an important part of the architecture.

In our view, practitioners aiming at evaluating fat-client storage systems in realistic conditions require a tool that: i) generates user-driven workloads, ii) executes such workloads at the client side, and iii) orchestrates groups of clients for evaluation and monitoring. Unfortunately, providing a practical solution to these problems is a research challenge.

User-driven workloads. Commonly, end-users heavily interact with fat-client storage systems, which poses the need for generating workloads based on user behavior (e.g., types of operations, intensity, sharing relationships). To give a sense of how users behave in these systems, we inspect the types of users in two real Personal Cloud providers: Dropbox [22] and UbuntuOne (U1) [37]. That is, in Fig. 1 we grouped users according to the classification proposed by Drago et al. in [22]. 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

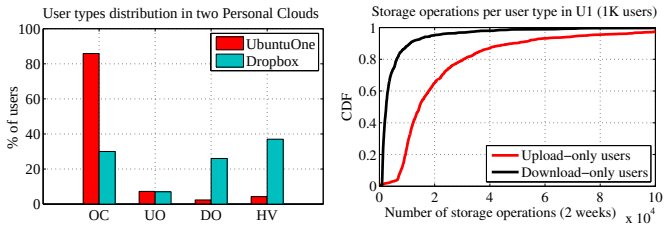


Fig. 1. On the left, we show the classification of users in DropBox and UbuntuOne populations (*occasional* (OC), *(down/up)load only* (UO/DO) and *heavy* (HV)). The right figure shows the distribution of storage operations of 1K active U1 users belonging to UO/DO types (2 weeks).

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.

Fig. 1 (left) illustrates the existence of different types of users in a Personal Cloud and their different distributions depending on the system at hand. Presumably, the distinct types of *user activity* will induce heterogeneous workloads against the system. Even worse, Fig. 1 (right) shows that even users of the same type may greatly differ in terms of *activity intensity*. For this reason, we need flexible mechanisms to model and generate user-driven workloads according to the types of users found in our system.

Client-centric benchmark. A complete performance evaluation of a fat-client storage system requires to generate the workload at software clients, so that client-side functionality can be evaluated as well [27]. Retaking our use-case service, an innovation path to optimize Personal Clouds is to embed data reduction techniques at the client-side [45], [46]. In fact, recent works have demonstrated that desktop clients integrate a mix of compression, deduplication and advanced syncing techniques on users' requests [13], [21], [44].

As can be inferred from Fig. 1, the *efficiency of desktop clients* executing data reduction and management techniques may be very different when supporting a very active upload-only user, compared to handling users mainly focused on downloading files from friends. Even more, the *characteristics of the data at hand*, such as the size of stored files and type of contents [22], [37] are also of capital importance to software clients. Therefore, we require means for executing user-driven workloads directly in fat clients that enable us not only to model storage operations, but also the types of contents generated.

Group orchestration and monitoring. In a fat-client storage system, interactions across users within a group may be relevant [19], [43]. Examples of these interactions are data sharing and collaborative work, among others. In a Personal Cloud, groups of users may share files and folders.

Fig. 1 gives clues on the joint effects of storage relationships and the individual activity of users. That is, from the system's viewpoint, a small and strongly connected group of very active users working on a shared folder may stress the system way more than a larger group of users with weak sharing relationships [36]. For this reason, practitioners require means for deploying *groups of users* that operate on shared data and define their behavior, as well as *monitoring facilities* to infer the impact of such activity on the fat client.

1.2 Lack of User-driven, Client-centric Benchmarks

Most storage benchmarks generate synthetic workloads to stress the server under configurable requests rates [3], [4], [21], [49]. While they enable practitioners to perform *what-if* testing, they fall short understand the system's operation under realistic conditions. Existing benchmarks that aim at reproducing the aggregated workload from real users are server-oriented and do not cope with the needs of fat clients [7], [10], [16]. The few works that focus on client-centric benchmarking [21], [27] generate synthetic workloads and do not provide means to reproduce user interactions with software clients. To our knowledge, there is no tool enabling researchers to generate user-driven workloads directly in software clients, not to mention the management and monitoring of client groups in an evaluation.

A tool filling this gap would be suitable for companies and research groups aiming at evaluating fat-client storage systems in realistic conditions or pre-production stages [46]. Moreover, it would help to answer important questions for the evaluation of these systems: Are software clients working efficiently for all types of users? How does the behavior of users impact on the resource consumption of a fat-client storage system? Does it matter for a fat-client system how users interact among them? To provide researchers with a framework to answer this kind of questions is our core goal.

1.3 Contributions

We introduce `BenchBox`: A distributed benchmarking framework for fat-client storage systems. `BenchBox` allows researchers to plug-in client-centric workload models and feed them with compact and sharable *recipes* that represent user groups exhibiting similar behavior, namely, *user stereotypes*. Workload models in `BenchBox` are flexible enough to model not only user activity, but also data contents as well. `BenchBox` also enables us to deploy experiments with groups of user stereotypes, as well as to monitor the activity of software clients. In summary, our contributions are:

- We designed and implemented `BenchBox`, the first benchmarking framework that allows us to model the behavior of users, generate workloads accordingly directly in fat clients, and monitor their activity.
- We equipped `BenchBox` with a practical yet realistic 2-layer workload model to evaluate storage services like Personal Clouds. Our model reproduces both the *activity* —e.g., types of operations, frequency— and *data* —e.g., file sizes, data types— of users. We used it to generate workloads based on user stereotypes that we identified in a large-scale service (UbuntuOne).
- We demonstrate `BenchBox` by evaluating public Personal Clouds. `BenchBox` sheds light on the performance impact of user behavior on these systems, as well as on the machines where fat clients run, which may help practitioners to optimize their operation.

The rest of the paper is as follows. Section 2 discusses related works. We overview the design of `BenchBox` in Section 3. In Section 4, we define a set of user stereotypes based on traces of a real-world provider (UbuntuOne). We present our workload model to mimic the behavior of user stereotypes in sections 5 and 6. We show the benefits of `BenchBox` by evaluating public Personal Clouds against our stereotypes in Section 7. We conclude in Section 8.

2 Related Work

Benchmarking tools. The performance evaluation of cloud and storage systems has been an active research topic in the last years [23], [56]. An excellent survey of Traegger et al. [59] 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" [2], [8], [16], [20], [41], [42], [54], [60], [62]. In other words, these tools facilitate to synthetically generate workloads guided by definable parameters (i.e., operations/second, file size) for exploring the system's 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 [10], [11], [57]. Seminal works in the field identified structural aspects of workloads to enable their synthetic generation, such as self-similarity and burstiness [9], [12], [18]. Recent efforts advocate to build a control loop to automatically infer workload models for their subsequent execution [39]. Regarding the use case of this work, the only informed Personal Cloud benchmark is that of Gonçalvez et al. [30]. Authors propose CloudGen, a synthetic trace generator of Dropbox-like workloads based on a new hierarchical model. Resulting traces of CloudGen can be used to analyze the behavior of the system at larger scales, or even as input for a trace replay. Given that `BenchBox` enables practitioners to *easily deploy new workload models* that reproduce the behavior of users, we believe that the workload model of CloudGen could be easily added in our framework for testing fat clients as well.

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), the fat-client architecture of many online storage services calls for enacting *clients as the subject of the benchmark* [27]. Otherwise, the system's evaluation would be partial and incomplete [55].

Closer to this work, there are only few works in the literature that targeted the client-centric evaluation of cloud services. Bocchi et al. [13], [21] and Li et al. [44] presented deep analyses of various Personal Cloud clients based on non-informed client-centric 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; [13], [21], [44] resorted to artificial workload patterns to infer the behavior of desktop clients, but providing no answers on their performance under user-driven activity. Moreover, in contrast to these works, our framework enables to plug-in workload models that, for example, pay special attention to generate realistic file contents during the evaluation process (data compression, folder structure, etc.) [33], [58].

Personal Cloud measurements. `BenchBox` is designed to facilitate practitioners measuring the performance of fat storage clients under user-driven activity. Among the various measurements of fat-client systems [40], [50], [53], several efforts have been devoted to analyze our use-case application: Personal Clouds. They 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 —despite some works also actively evaluate server side components [35]. The first work in this sense is [38], in which authors evaluated simple transfer desktop client backup/restore times in several vendors depending on types of backup contents. As mentioned before, other works [13], [21], [44], [45] 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 and/or overhead, which impacts on the user quality of experience (QoE) [14]. 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. [22] 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. [28] extended that measurement by modeling the behavior of Dropbox users. Similarly, Mager et al. [47] uncovered the architecture and data management of Wuala, a peer-assisted Personal Cloud. More recently, a back-end measurement of UbuntuOne was presented in [37], 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 [11], [41], [52]. In this regard, most of these works target to generate realistic HTTP traffic for Web applications. For instance, authors in [11] 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 [41] 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 [52] 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 criteria of similarity [48]. 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 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 framework 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: Framework Design

BenchBox is a distributed performance evaluation framework for fat-client storage systems. BenchBox targets to reproduce different types of client-centric storage workloads in a compact and sharable manner. Moreover, the framework enables to plug-in different models for generating workloads, based on a practitioner's needs. To achieve this, BenchBox separates the abstractions to characterize and reproduce user workloads (Section 3.1) from the mechanisms to deploy and execute them (Section 3.2).

3.1 Abstractions and Lifecycle

User stereotype: 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" [1]. Similarly, in a fat-client storage system, we define a *stereotype* as a set of conditions applied on the workload characteristics of users that results in a user group. The objective of a user stereotype is to represent a group of users that is relevant to the performance evaluation of the system.

Workload model: A workload model is a piece of logic that emulates certain aspects of the behavior of user stereotypes based on some input information. In BenchBox, workload models are client-centric; this means that the generated storage activity is handled by a single desktop client. During a performance evaluation, BenchBox instantiates a workload generation process per desktop client that runs a given workload model. Workload models are also pluggable, so researchers may design and share new approaches for generating storage activity based on the behavior of users.

Stereotype recipe: A stereotype recipe is a quantitative representation of the workload aspects of a user stereotype to be reproduced during the workload generation (e.g., fittings, activity rates). In our framework, stereotype recipes are the input for workload models; this yields that each stereotype recipe should contain the necessary information to feed a workload model according to the group of users that represents. Stereotype recipes can be designed to be compact, which facilitates sharing among researchers.

Working with BenchBox. The first step to use BenchBox is to group users found in real system traces into stereotypes according some criteria, such as their *type of activity or even their social relationships* (see Fig. 2). Naturally, the grouping criteria 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 diverse situations. As a use case, in Section 4 we identify a set of stereotypes in U1 that will allow us to reproduce different types of realistic user behavior in the wild.

Once groups of users are classified into stereotypes, the next decision to make is *how to model the aspects of their behavior that matter for an evaluation*. Intuitively, there is not a universal model; for instance, one practitioner might be interested on modeling with high precision the inter-operation times of users and the types of files being managed, whereas another might aim at capturing their connectivity patterns (e.g., non-stationarity). As a practical example, in this work we propose a storage workload model that encompasses two layers: i) the *activity layer*, which dictates the operations to be executed and when these events occur (Section 5), and ii) the *data layer* that mimics the management of user files and the structure of synchronized folders (Section 6).

Having both a workload model and the stereotypes in place, the next task consists of *building stereotype recipes*. As visible in the code snippet below, this requires a practitioner to create a file for each stereotype capturing the values needed as input to guide the workload model:

```
['Upload Only' stereotype recipe]

user_transitions, DELETE, DELETE, 0.3
user_transitions, DOWNLOAD, DELETE, 0.1
...
file_types_sizes, {Code: (pareto, {params}), ...}
file_update_prob, {Code: 0.12, Docs: 0.01, ...}
...
```

In this simple example, let us assume that the workload model is able of executing storage operations based on some user states, such as DELETE or DOWNLOAD. Visibly, the first two lines of the recipe are dictating to the workload model the probabilities of transitioning between these states for *upload only* users. Similarly, we could assume that the workload model can also generate realistic files. In this case, the properties file_types_sizes and file_update_prob instruct the workload generator about the size of files to be generated, as well as the probabilities of updating a particular type of file. As one can infer, stereotype recipes may be very compact and easy to share; for instance, none of the recipes built in this work exceeds 7KB in size.

The last step is to evaluate fat clients with our workload models and stereotype recipes. Next, we present the architecture of BenchBox that allows practitioners to execute workload generators on desktop clients, manage stereotypes recipes and perform extensive client-side measurements.

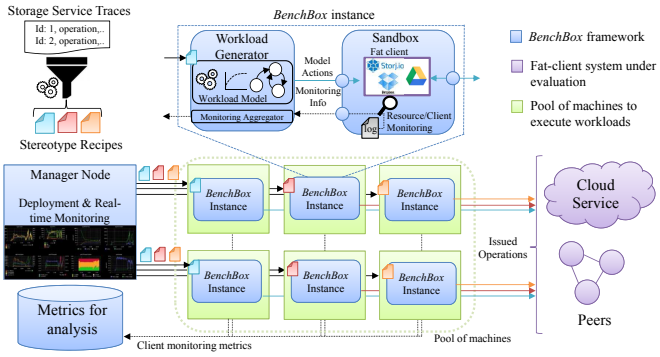


Fig. 2. Architecture, lifecycle and deployment of BenchBox.

3.2 Architecture

As visible in Fig. 2, 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 client-side workloads, 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 execute a workload model. These processes take as input an stereotype recipe file for generating workloads driven by a particular user behavior. What constitutes a workload depends on the storage service under test; it consists of a series of commands issued to the fat client (e.g., CRUD operations on files), even including synthetic files and folders to be managed. Workload generators translate the workload into commands (e.g., FTP, API calls) that are sent via a virtual interface to the fat-client at the sandboxed virtual machine.

On the other hand, sandboxes are intended to run fat clients in an *isolated environment* to obtain accurate monitoring. That is, BenchBox incorporates tools to capture the activity of a software client (CPU, disk, memory and network), in order to track its operation and detect potential inefficiencies. In addition to resource monitoring, other metrics related to the specific operation of clients, such as synchronization time or conflict ratio, can be injected [22], [44]. Similarly to [21], all the network activity that desktop clients produce is monitorized in a primary virtual network interface, whereas a secondary one is used for both receiving storage commands from the workload generator and for sending monitoring information. In particular, the sandbox sends 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.

One may consider that the BenchBox's architecture and lifecycle (e.g., stereotype classification, workload modeling) are more complex than existing benchmarking tools [21], [44]. While this may be true, such tools are mainly devised to infer the operation and worst-case performance of fat clients via "what-if" tests on artificial scenarios. In contrast, the design of BenchBox enables us to analyze the average-case performance of these systems by modeling realistic scenarios for different types of users. All in all, we next demonstrate the practicality of our framework by applying it on a very popular fat-client storage service: Personal Clouds. To this end, we first identify user stereotypes in a real service traces, and then we design a workload model that reproduces the behavior of users in these systems.

4 Defining Stereotypes: UbuntuOne

In this section, we define user groups or stereotypes from a real trace from UbuntuOne (U1) [37]: the Personal Cloud of Canonical, integrated by default in Linux Ubuntu OS. The key feature of this trace is that it has been collected from the service back-end by the provider itself, including the activity of 1.29 million desktop clients for 30 days from (January-February 2014). The trace contains three types of requests: i) *storage requests* issued by desktop clients; ii) *session requests* from client connection events; and iii) *back-end requests* (RPC calls). From the original dataset, we used a subtrace consisting of all the users exhibiting any type of storage activity every 48 hours for 2 weeks. This helps us to discard from our classification many inactive users that are less interesting from a benchmarking perspective.

Our main objective is to distinguish different user groups that exhibit disparate behaviors within U1. To this end, previous works [22], [37] suggested that we can create meaningful user groups based on the *type of storage traffic* and the *intensity of user activity*[2]. Therefore, we classified U1 users into the following stereotypes:

Back-up users: As visible in Fig. 1, 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 (68.12K users).

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 (29.21K users).

Content distribution users: Due to desktop client syncing or sharing activity, there are users that exhibit high amounts of download traffic. We classify a user whose download operations are two orders of magnitude higher than his download operations as content distribution (24.88K users).

As pointed out in Fig. 1, we observed that the activity of users in a Personal Cloud is highly skewed, showing a form of Pareto principle [37] (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 an 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 [32]. 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**).

In the following, we describe our workload model that translates users stereotypes into storage activity.

5 Activity Layer: Modeling Active Users

Here we present a model to capture the activity of users in a real Personal Cloud system. Our aim is to "boil down" the complexity of modeling user activity until only the key aspects remain: the *type* and *intensity* of user operations.

5.1 State Definition

To model the activity of users, it is first required to define the states to emulate from users [28]. In this sense, Fig. 3 shows

2. More complex user classifications are indeed possible [24], [48], but out of the scope of this paper.

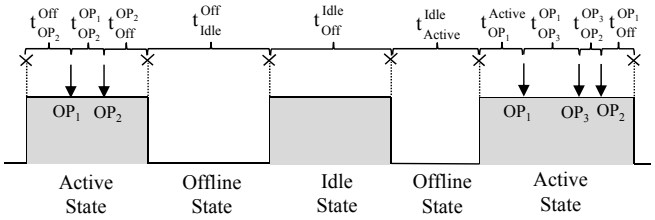


Fig. 3. Activity states of a user in U1.

how a user typically interacts in a Personal Cloud system. It can be distinguished three "general" states:

- **Active:** A user is online and emits storage operations against the system;
- **Idle:** A user is online but with no signal of activity;
- **Offline:** A user is disconnected.

The *Offline* state is irrelevant in this work[3]. The focus is on modeling the activity of users when they are online. The *Idle* state is important to characterize, because it dictates the intensity of the workload. Simply put, if the user is heavy or occasional. The *Active* state, however, allows the emulation of the distinct types of storage activities such as file backup, synchronization, etc. As we show later in this paper, a model considering *Idle* and *Active* states is sufficient to satisfy the benchmarking requirements imposed by stereotypes.

As visible in Fig. 3, a Personal Cloud user may perform several operations on his desktop client while in *Active* state. So, in a similar way to the interactions with a file system, we consider the following operations in our model: `Upload`, `Download`, `Remove`, `Move` and `Sync` — or file update. The execution of such operations on files and directories within a sync folder trigger management tasks at the desktop client. Therefore, we need a workload generator process to model these storage operations as well as for transitioning between *Idle* and *Active* states within the context of a desktop client.

5.2 Semi-Markov Process for Modeling User Activity

Given that user activity shows a continuous-time behavior, a valid approach to characterize it could be a continuous-time Markov process (CTMP). Unfortunately, after the empirical observation in U1 that the sojourn times in all the states are not always exponentially distributed, our choice was a semi-Markov process (SMP) as the engine to govern the execution of our workload generator [31].

An SMP is a stochastic process whose transition behavior can be characterized by two main elements. As the transition between states follow a Markov chain, the first element is a transition matrix $\mathbf{P}$:

$$\mathbf{P} = \begin{bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{bmatrix},$$

where p_{ij} denotes the probability that the process transitions from state i to state j. Given that the system is in state i and will move to state j, the second element is the sojourn time t spent in that state, which is distributed according to cumulative distribution function $Q_{ij}(t)$. When put in matrix form, this is called the kernel of the SMP.

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

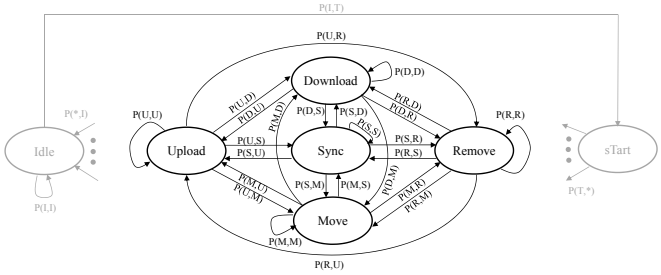


Fig. 4. Markov model for handling user operations. State transitions are denoted tuples formed by the first letters of the source and destination states. To wit, (S,M) denotes a `Sync`→`Move` transition and P(S,M) is the transition's probability. These probabilities build the transition matrix.

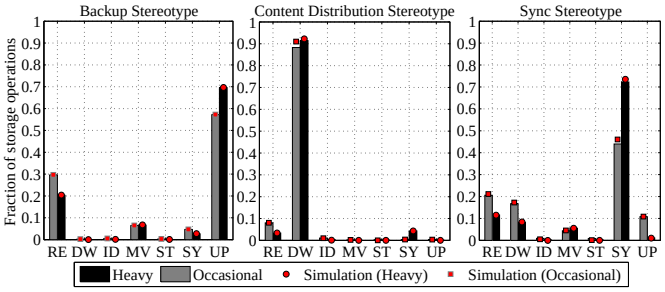


Fig. 5. Simulation results of our workload model against a U1 trace replay of 1K users per stereotype during 2 weeks. Bars correspond to the probability of executing a given operation: Remove (RE), Download (DW), Idle (ID), Move (MV), Start (ST), Sync (SY) and Upload (UP).

Fig. 4 shows a representation of the possible states and transitions described by our SMP. While the client is *Active*, the system transitions from one storage operation to another or may go to the *Idle* state, where the client remains idle for a given period of time.

In practice, modeling user behavior as an SMP involves three tasks: i) Estimation of the initial distribution vector $\mathbf{p}$; ii) Estimation of the transition probabilities $\mathbf{P}$; and iii) Fitting of the sojourn times. All this information is expressed in a compact manner in our stereotype recipes.

5.2.1 Estimating transition probabilities

The first thing to do is to estimate the transition probabilities from the U1 measurement data. To do so, we classify every operation according to the states of our model. Then, we use the following well-known maximum-likelihood estimator to obtain estimates for the transition probability matrix:

$$\widehat{p}_{ij} = \frac{n_{ij}}{n_i},$$

where n_{ij} is the number of transitions $i \to j$ occurring in the U1 trace, and $n_i = \sum_k n_{ik}$ is the number of times that the system resides in state i. We note that a transition matrix $\mathbf{P}$ is output per stereotype. This means that given a stereotype S, only the observed transitions $i \to j$ of those users that match stereotype S are considered. The same also applies for n_i.

Fig. 5 illustrates the fraction of storage operations of 1K users per stereotype selected in U1 for 2 weeks. Fig. 5 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. In general, the stereotype transition

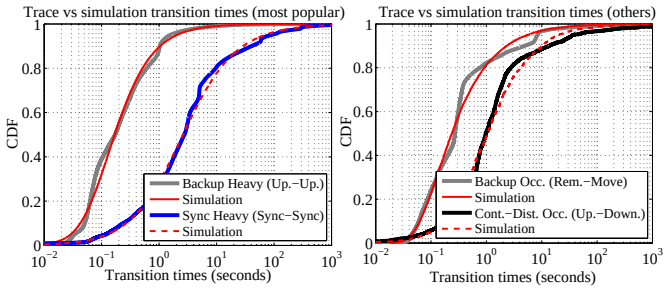


Fig. 6. Comparison of U1 trace transition times with a simulation of our workload model with fitted parameters as input for various stereotypes. This helps us to model the activity intensity of users.

matrix $\widehat{\mathbf{P}}$ agrees well with the storage activity of real users of that stereotype. For frequent storage operations, our results do not show a deviation higher than $\pm 5.8\%$, which is by far acceptable for benchmarking purposes.

Further, we see that for unpopular operations such a deviation becomes higher. For instance, for `Backup-Heavy` and `Backup-Occasional` user stereotypes, SMP results yield an error of 23.08% and 11.75%, respectively. However, as `Downloads` only represent 0.02% and 0.24% of the total amount of operations in these user stereotypes, this error is proportionally negligible in practice.

5.2.2 Fitting Sojourn Times

Now it only remains to determine the distribution $Q_{ij}(t)$ for the sojourn times in each state.

Given a set of U1 users of a given stereotype, we capture of all the sojourn times related to a given transition $i \rightarrow j$. We fit the empirical observations corresponding to each transition against a battery of well-known distributions with maximum likelihood estimated parameters [28], [30]. In our stereotype recipes, each transition is associated with the distribution that reports the best goodness-of-fit. We notice that, although apparently this is a time-consuming process, the SMP model shown in Fig. 4 can be simplified for many stereotypes with marginal error as some state transitions are occasional. Less transitions implies finding less distributions that provide a good fit for $Q_{ij}(t)$.

To validate the accuracy of this method, we compare results from our selected U1 users with simulated users running our workload model. Concretely, Fig. 6 (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 [37]. These observations are consistent with Fig. 6 (right), where our simulations present a similar shape compared to less popular empirical state transitions of `Backup-Occasional` and `Content Distribution-Occasional` users.

6 Data Layer: Mimicking Sync Folders

The second building block of our workload model 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.

6.1 Sync Folder Structure

To evaluate Personal Cloud desktop clients under realistic conditions it is important to model 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 [7]) 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 that 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` 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. [7] 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.

6.2 File Generation

Upon a file operation, desktop clients apply data reduction techniques, such as compression or deduplication, to optimize their operation [44]. Clearly, file sizes and contents (e.g., text, random data) greatly influence the performance of data reduction techniques [33].

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 [37]: `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. 7 we show the fraction of files per category and user stereotype. As a common characteristic, considering `Heavy` stereotypes,

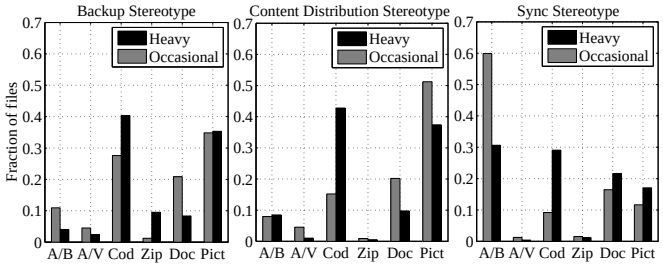


Fig. 7. Fraction of files for each user stereotype per file category: Application/Binary (A/B), Audio/Video (A/V), Pictures (Pict), Code (Cod), Compressed (Zip), Docs (Doc).

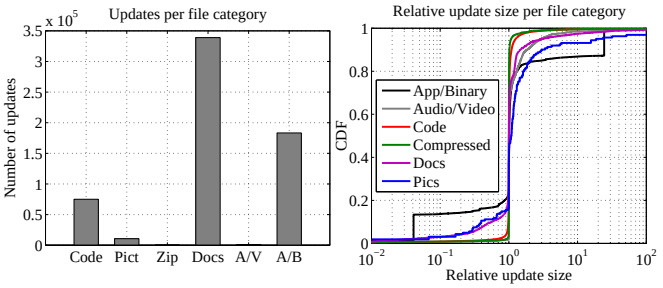


Fig. 8. Number of edits per file category (left) and relative size of updates (right) per file category.

`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`.

In our model, each file category is described by the *distribution of file sizes*[4] —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 [33] 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% [37].

6.3 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. 8, 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. 8 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. 8, 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 subtractions/additions of larger amounts of data. To reproduce the size of updates per file category, we fitted the distributions in Fig. 8 against a battery of distributions[5].

Finally, it is important to note that there are different types of file edits. According to the model Tarasov et al. [58], 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 in [58].

7 Evaluating Personal Clouds

In this section, we demonstrate the potential of `BenchBox` by evaluating commercial Personal Cloud vendors.

7.1 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/Puppet. `BenchBox` sandbox instances send monitoring information to InfluxDB via RabbitMQ for further analysis and for real-time monitoring inspection (Grafana). The source code of `BenchBox` is publicly available[6].

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

4. 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 $\theta = -2.2204\text{e}{-015}$.

5. 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` ($\mu = 0.9999$, $\sigma = 0.0116$, $\nu = 0.3808$).

6. https://github.com/cloudspaces/benchbox

Dropbox	TUE (σ)	iTUE (σ)	oTUE (σ)
Backup Heavy	0.471 (0.047)	0.048 (0.007)	0.423 (0.042)
Backup Occ.	0.554 (0.204)	0.199 (0.163)	0.355 (0.105)
Sync Heavy	0.233 (0.037)	0.023 (0.006)	0.210 (0.032)
Sync Occ.	0.297 (0.151)	0.047 (0.042)	0.250 (0.109)
Download Heavy	0.305 (0.032)	0.225 (0.059)	0.080 (0.048)
Download Occ.	0.378 (0.029)	0.335 (0.052)	0.043 (0.039)
Google Drive	TUE (σ)	iTUE (σ)	oTUE (σ)
Backup Heavy	0.965 (0.078)	0.058 (0.018)	0.907 (0.088)
Backup Occ.	1.027 (0.057)	0.051 (0.003)	0.976 (0.055)
Sync Heavy	0.470 (0.431)	0.021 (0.019)	0.449 (0.412)
Sync Occ.	0.827 (0.280)	0.038 (0.012)	0.789 (0.268)

TABLE 1
TUE metric per stereotype for Dropbox and Google Drive.

All the experiments lasted over 4 hours and were executed on 6 machines. According to Fig. 2, each pair of workload generator and sandbox VMs ran in a single physical machine. This allows us to inspect resource consumption of sandbox VMs per machine. Workload generators were initialized with a unique random seed within the group to introduce some (reproducible) behavior variance in each desktop client. However, seeds were consistent across Personal Clouds. The orchestration and execution of `BenchBox` instances was performed from the manager node.

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 (e.g., `PUT/GET`) in user accounts [35]. 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 time usage limitations of Google Drive REST API, we could not generate download oriented workloads for this provider (`Content Distribution` stereotype). This may be due to prevent service abuses, such as the storage leeching problem [34].

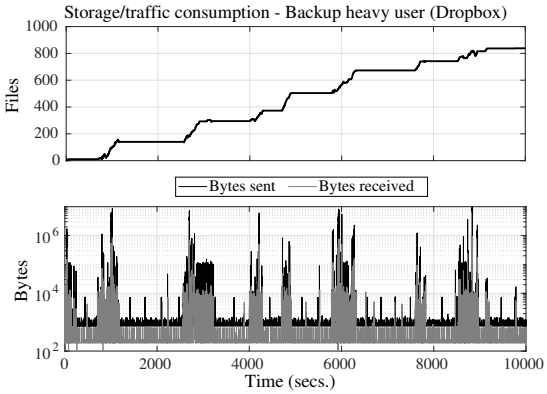
In our evaluation, we aim at understanding the impact of different user behaviors on cloud resources and desktop clients. Therefore, we account for metrics that are heavily influenced by such behavior (e.g., CPU, network traffic, hard disk, RAM). Some of these metrics have not been reported before, so our evaluation complements existing measurements of Personal Cloud desktop clients [21], [44].

Regarding sharing groups, we have reported that disparate sharing topologies exist within the same system, specially in terms of connectivity or clustering [32]. 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).

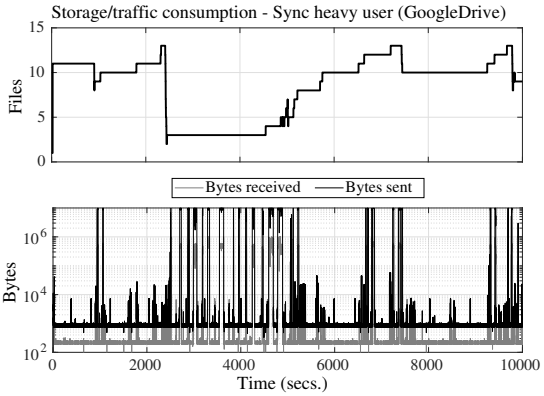
Platform. We executed `BenchBox` and Personal Cloud desktop clients in machines that incorporate an i5 processor, 4GB DDR2 of RAM and 500GB of hard-disk storage running a Debian 3.2 distribution. The execution of desktop clients was performed in VMs with Windows 7 Enterprise (64 bits) as operating system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

7.2 Impact of User Behavior on Fat-Client Efficiency

We analyze the impact of user behavior on the efficiency of desktop clients for both Dropbox and Google Drive services. To measure efficiency, we resort to the Traffic Usage Efficiency (TUE) metric proposed by Li et al. in [44]. This



(a) Backup Heavy user (Dropbox).



(b) Sync Heavy user (Google Drive).

Fig. 9. Relationship between storage and traffic consumption for individual stereotype users.

metric is defined as TUE $= \frac{\text{Total data sync traffic}}{\text{Data update size}}$. That is, a TUE > 1 means that the software client transfers to the server more bytes than the ones actually written to disk, which is an indicator of inefficiency. In this sense, iTUE and oTUE metrics are just the fraction of TUE related to inbound and outbound traffic, respectively. To calculate these metrics, we considered only operations that involve writes to disk, and therefore data transfers (as observed in [44], `Delete` operations have negligible impact on network traffic).

In Table 1, we see that both Dropbox and Google Drive clients are efficient across all stereotypes, as TUE < 1 in all cases. This is in stark contrast with the results presented in other works such as Li et al. in [44], where TUE typically ranges between 1.2 and 13. The reason for this difference lies in the ultimate goal: authors in [44] ran "*what-if*" tests to determine how desktop clients work. To that aim, they set up artificial scenarios, for instance, by assuming that all files are incompressible. However, `BenchBox` pursues other goals like the average-case analysis of these systems by modeling realistic scenarios. Without a 2-layer workload model such as that of `BenchBox`, it would be hard, or almost impossible, to perform a realistic average analysis of these systems, as it demonstrates the TUE results here.

Contrary to conventional wisdom, Table 1 shows that software clients obtain higher efficiency for `Heavy` users than for `Occasional` ones, despite the formers generate more data. To wit, on average, `Backup-Heavy` users in Google Drive and `Content Distribution-Heavy` users

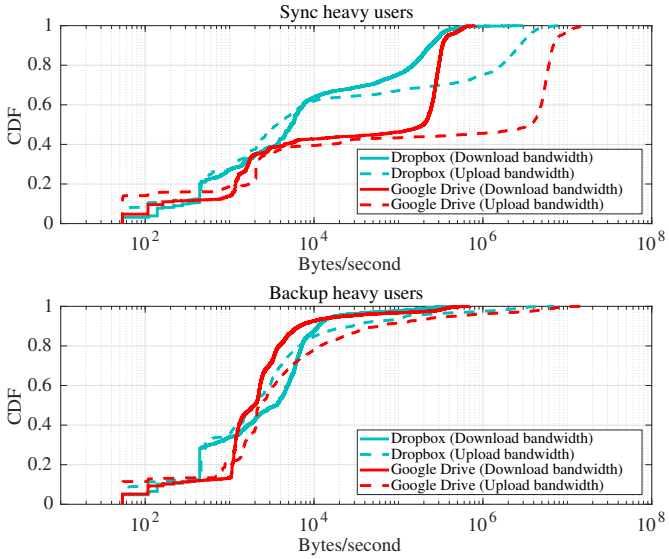


Fig. 10. Aggregated upload/download bandwidth distribution of desktop clients depending on the stereotype/Personal Cloud.

in Dropbox write to disk 1.58x and 1.56x more data bytes compared with their `Occasional` counterparts, respectively. One reason why `Heavy` stereotypes exhibit lower TUE values is that they tend to *manage more compressible files* compared with `Occasional` users (such as documents and code files, as visible in Fig. 7). Given that both Dropbox and Google Drive implement *data compression* [13], [21], [44], managing more compressible files leads to higher efficiency.

Further, `BenchBox` allows practitioners to capture the existing differences across desktop clients under a variety of realistic workloads. For instance, Table 1 confirms that Dropbox exhibits a high efficiency for `Backup` and `Content Distribution` users. The reason is that Dropbox implements sophisticated data reduction techniques [13], [21], [44]. To wit, it implements file bundling to group several small files into a single transfer, and thus minimize network overhead. And it supports data de-duplication to avoid the upload of files with similar content. `BenchBox` allows to study these techniques as well. That is, it is able to mimic the synchronization of a group of files, as visible in Fig. 9(a), and reproduce small file sizes (90% of files are $<$ 1MB) along with their de-duplication ratios.

A very counterintuitive observation can be made w.r.t. `Sync` users. Although `Sync-Heavy` users write 15.8x more bytes to disk than `Backup-Heavy` clients, both Personal Cloud services yielded the highest efficiency for `Sync` users. For Google Drive, this can be partially explained because of the effectiveness of *sync deferment*: a data management technique that delays file synchronization to ameliorate the impact of frequent file modifications [44]. As described in Section 5, our workload model captures the *burstiness of user operations*, including files updates (see Fig. 9(b)). In fact, our experiments show that file updates represent 80% and 48% of the total number of operations for `Sync-Heavy` and `Sync-Occasional` users, respectively. For these stereotypes, we found that 50% of consecutive inter-update time intervals were of $<$ 3 secs and $<$ 4.9 secs. Given that Google Drive has a sync deferment wait interval of 4 seconds, we conclude that low TUE values are due to that most file updates are never transferred to the cloud.

Dropbox	CUE (σ)	MUE (σ)
Backup Heavy	39.149 (13.384)	0.679 (0.178)
Backup Occ.	39.218 (13.329)	0.901 (0.312)
Sync Heavy	8.201 (1.225)	0.218 (0.066)
Sync Occ.	12.197 (5.047)	0.460 (0.241)
Download Heavy	27.730 (10.15)	0.435 (0.140)
Download Occ.	24.615 (2.612)	0.465 (0.049)
Google Drive	CUE (σ)	MUE (σ)
Backup Heavy	23.201 (5.882)	1.095 (0.453)
Backup Occ.	30.221 (22.940)	1.415 (0.359)
Sync Heavy	2.620 (2.920)	0.085 (0.075)
Sync Occ.	18.557 (8.016)	0.651 (0.295)

TABLE 2
CUE and MUE metrics per stereotype for Dropbox and Google Drive.

Furthermore, Table 1 also shows that Dropbox is more efficient than Google Drive for `Sync` users. This is because Dropbox uses *delta encoding* to transmit only the modified parts of a file, instead of the whole file [13]. The benefits of delta updates can be seen in Fig. 10. `Sync-Heavy` users in Dropbox consumes less bandwidth than Google Drive. The reason is that Google Drive does not implement delta encoding, so the entire file is sent to the cloud. In contrast, Dropbox clients only transfer small deltas. However, as shown in Fig. 10, both Google Drive and Dropbox exhibit similar bandwidth performance profiles when file updates are less frequent (e.g., `Backup` users).

Summary. We summarize our observations as follows:

- We found that existing measurements may underestimate the efficiency of clients in real conditions. To solve this problem, `BenchBox` is the first tool providing user-driven evaluations of fat-storage clients.
- Also, `Heavy` users manage more data but they enjoy of a higher efficiency than `Occasional` users. This is because software clients are equipped very well to handle intensive behavior (e.g., bursts of operations on compressible files).
- Our results confirm that Dropbox clients are between 1.85x to 2.78x more efficient than Google Drive. This is due to the data management techniques supported by Dropbox (e.g., delta sync, deduplication).

Contribution. `BenchBox` enables the realistic evaluation of fat-client storage systems under user-driven workloads. Simply put, `BenchBox` permits from average case analysis to fine-grained assessment of specific types of users, thereby reducing the risk of biasing the evaluation of these systems.

7.3 Local Resource Usage on User's Machines

In this battery of experiments, we analyze an unexplored facet of these services: The resources that fat 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 [14].

To start with, we extend the idea of TUE to CPU and memory usage (see Table 2). On the one hand, we propose the CUE (CPU Usage Efficiency) metric, which is defined as $\text{CUE} = \frac{\text{CPU usage per second}}{\text{Data update size}}$. On the other hand, we propose the MUE (Memory Usage Efficiency) metric, which is calculated as $\text{MUE} = \frac{\text{Memory allocation size}}{\text{Data update size}}$. Such metrics will help us to

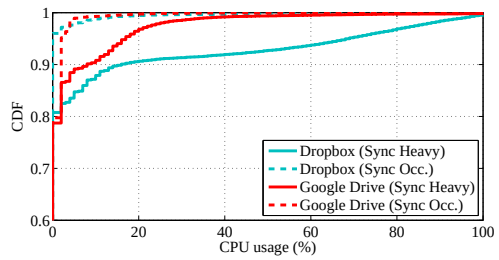


Fig. 11. Aggregated CPU consumption CDF of Dropbox and Google Drive for *Sync-Heavy* users.

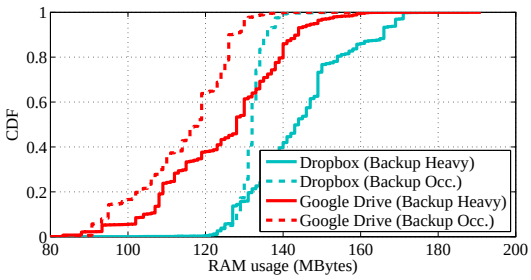


Fig. 12. Aggregated RAM usage CDF of Dropbox and Google Drive desktop clients for *Backup-Heavy* users.

understand the relationship between the amount of data users write to disk and the relative CPU and memory consumption, respectively.

In terms of CPU, Table 2 shows a similar (but weaker) pattern to that of network traffic efficiency. That is, despite `Heavy` users are the ones that consume more compute resources, software clients achieve similar or better efficiency compared to `Occasional` users (see Fig. 11). However, an important difference compared to TUE values is that Google Drive achieves better CUE than Dropbox in many cases. A reason for this is that Google Drive applies *smart data compression*: the client only tries to compress files that are potentially compressible [13]. As data compression is a CPU-intensive task, this technique saves significant compute resources. Given that `BenchBox` models file contents of users, we can infer the benefits of this technique.

The case of `Sync` users is especially interesting (see Fig. 11). For `Sync-Heavy` users, Table 2 shows that Google Drive achieves better CUE than Dropbox. This is mainly due to the combined effect of the aggressive sync deferment mechanism of Google Drive and the high "update burstiness" of these users. Thus, apart from avoiding a traffic overuse problem, Table 2 indicates that Google Drive clients do not consume CPU resources for most updates. However, Table 2 also shows that Dropbox makes a more efficient use of CPU resources for `Sync-Occasional` users. This is because intervals between updates are longer and less "bursty" in this stereotype, which yields Google Drive to propagate more updates to the sever. Given that Google Drive uses full-file sync, this consumes CPU resources on re-compressing files and transferring them to the cloud. We conclude that these experiments support the methodology of `BenchBox`: even for the same stereotype, we need to model both the type of user operations and their intensity.

Conversely to CUE, Table 2 shows that Dropbox clients achieve better MUE values than Google Drive clients. That is, for each byte written to disk, Dropbox allocates between

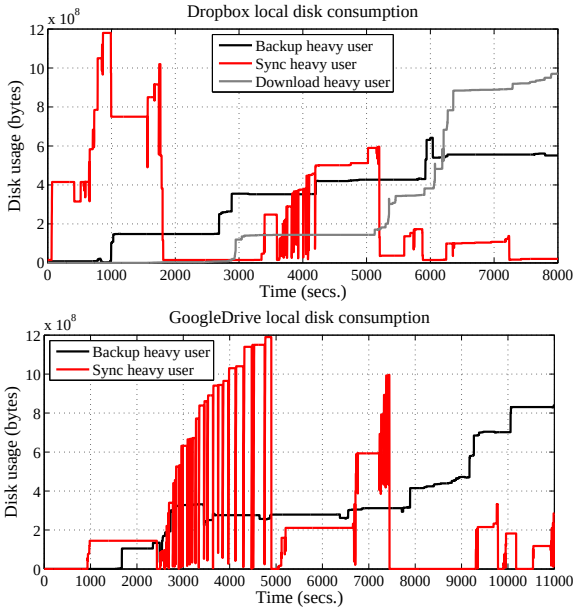


Fig. 13. Time-series evolution of the size of sync folders exhibited by individual stereotype users in Dropbox and Google Drive.

29% and 38% less memory than Google Drive (except for `Sync-Heavy` users). Nevertheless, in absolute values, Fig. 12 shows that Dropbox consumes a higher amount of memory than Google Drive. That is, Dropbox clients allocate by default around $\approx$ 32 MBytes of RAM more than Google Drive clients. This suggests that there could be initialization processes in Dropbox that may be optimized. Thanks to the monitoring tools of `BenchBox`, practitioners can understand potential inefficiencies to optimize fat storage clients.

The last metric that we analyze from a user's machine viewpoint is the usage of disk. That is, Fig. 13 shows the disk usage of individual users of each tested stereotypes in Dropbox and Google Drive. Concretely, the disk usage metric depicts the size on disk of contents written to the Personal Cloud sync folder (user content or desktop client internal data). As expected, `Backup` and `Content-Distribution` users exhibit an increasing usage 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. 9).

However, it is very interesting to observe that `Sync` users tend to exhibit the highest disk utilization in some parts of the experiment, despite these users are not storage intensive. The explanation for the growth of the sync folder for `Sync` users 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 done to exploit temporal locality of file management and avoid extra communication with the server. As shown in Fig. 13, the consequence of this optimization is that under intensive file update workloads the cache can grow dramatically.

Summary. We summarize our observations as follows:

- In practice, simple techniques such as timeout-based sync deferment and smart data compression can save important local machine resources from users.
- We observed that Google Drive is more CPU/memory efficient for `Sync-Heavy`, whereas Dropbox is

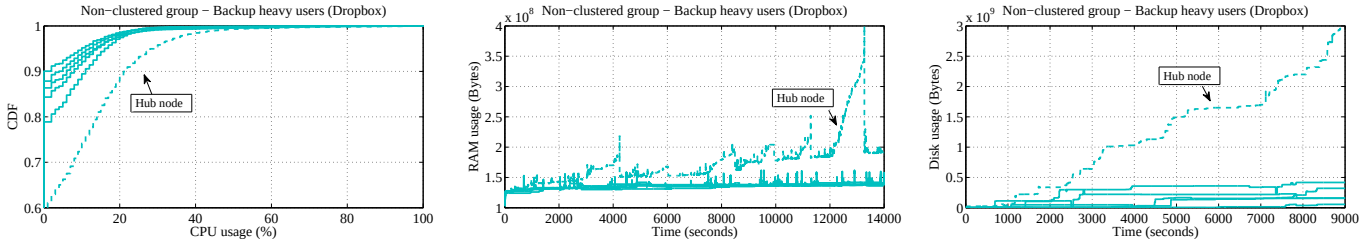


Fig. 15. Local resource consumption of Dropbox desktop clients for `Backup-Heavy` sharing groups (`Non-clustered`).

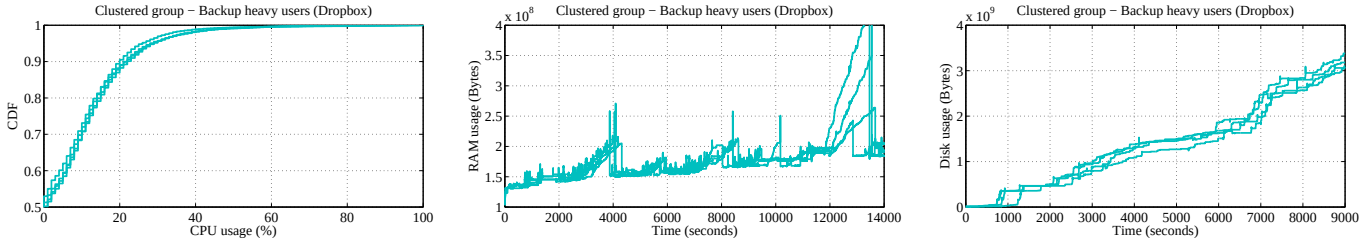


Fig. 16. Local resource consumption of Dropbox desktop clients for `Backup-Heavy` sharing groups (`Clustered`).

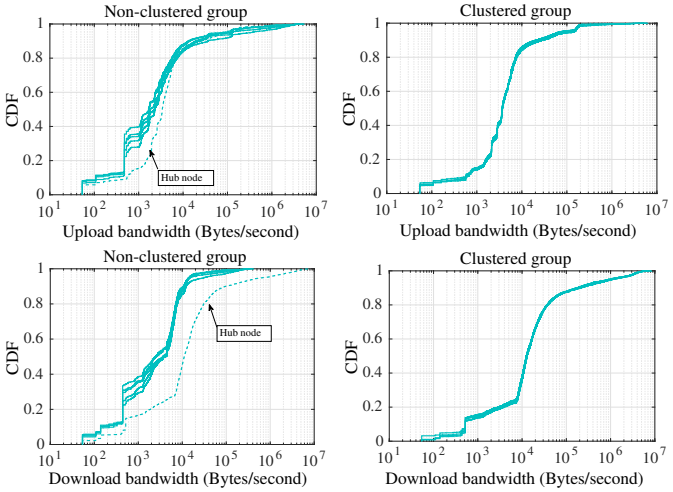


Fig. 14. Upload/download bandwidth of individual `Backup-Heavy` users in Dropbox depending on the data sharing topology.

the winner for `Sync-Heavy` users. This supports the relevance of user modeling to evaluate such systems.

- `Sync` users may experience a hidden and important consumption of disk due to the internal caching mechanisms implemented in software clients.

Contribution. `BenchBox` enables practitioners to analyze how different types of user-driven workloads (including types of operations and data characteristics) impact on the local machine resources of fat storage clients.

7.4 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 [32]. For this set of experiments we used the `Backup-Heavy` stereotype.

Fig. 14 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 of the time the hub node should download data coming from the rest of nodes.

On the other hand, we 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. 15 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. Similarly, in terms of RAM, the hub node exhibits spikes of memory consumption that 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.

Conversely, Fig. 16 shows 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. Fig. 16 also shows a similar problem to the hub node exists across all the nodes forming the `Clustered` topology, with some desktop clients consuming more than 400MB of RAM.

Fig. 15 illustrates 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 (Fig. 16); that is, after 2.5 hours of experiments all the nodes exhibited a sync folder size $>$ 3GB.

Summary. We summarize our observations as follows:

- In `Non-Clustered` topologies the hub node is highly overloaded compared to the rest of members.
- `Clustered` groups present a homogeneous but very high resource utilization.
- Personal Clouds should consider users' sharing links to devise data management techniques for fat clients.

Contribution. `BenchBox` is the first tool that allows practitioners to evaluate the joint implications of user behavior and sharing relationships in a fat-client storage system.

8 Conclusions

To overcome the limitations of today's benchmarks for fat-client storage systems, we presented `BenchBox`: A novel client-centric benchmarking framework that enables modeling groups of users with similar behavior —*user stereotypes*— for generating user-driven workloads. To demonstrate our framework: i) We designed a *workload model* that reproduces both the activity and storage contents of users in a Personal Cloud; ii) We elaborated *stereotype recipes* as input for our model from typical user behaviors found in traces of a large-scale service (UbuntuOne); and iii) We deployed experiments on public Personal Clouds to analyze the impact of user stereotypes on their operation and efficiency.

Our experiments show that the efficiency of desktop clients may greatly vary depending on the type of user supported, as well as their consumption of both cloud and local machine resources. Furthermore, we found that modeling in fine grain the data contents of users is key to understand the actual benefits of the data management techniques embedded in desktop clients. These results support `BenchBox` as a tool that can help researchers and practitioners to optimize fat-client storage systems via realistic user-driven workloads. In our next research actions, we will use `BenchBox` for evaluating other fat-client storage systems in which user behavior plays a key role, including new metrics that take into account their logical operation.

Acknowledgements

This work has been funded by the EU projects *CloudSpaces* (317555) and *IOStack* (644182) and by the Spanish Ministry of Science and Innovation research projects *Cloud Services and Community Clouds* (TIN2013-47245-C2-2-R) and *Software-Defined Edge Clouds* (TIN2016-77836-C2-1-R).

References

[1] Collins dictionary. http://www.collinsdictionary.com/dictionary/english/stereotyping.
[2] Bonnie++. http://www.coker.com.au/bonnie++/, 2001.
[3] Fio. http://freecode.com/projects/fio, 2005.
[4] Filebench. http://sourceforge.net/projects/filebench/, 2008.
[5] Scale-out file sync & share with red hat storage and owncloud. https://owncloud.com/wp-content/uploads/RHS-ownCloud-Performance.pdf, 2014.
[6] Dropbox - inside lan sync. https://blogs.dropbox.com/tech/2015/10/inside-lan-sync/, 2015.
[7] Agrawal, N., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Generating realistic impressions for file-system benchmarking. In *USENIX FAST '09* (2009), pp. 125–138.
[8] Anderson, E., Kallahalla, M., Uysal, M., and Swaminathan, R. Buttress: A toolkit for flexible and high fidelity i/o benchmarking. In *USENIX FAST'04* (2004), pp. 4–4.
[9] Arlitt, M. F., and Williamson, C. L. Web server workload characterization: The search for invariants. In *ACM SIGMETRICS Performance Evaluation Review* (1996), vol. 24, pp. 126–137.
[10] Armstrong, T. G., Ponnekanti, V., Borthakur, D., and Callaghan, M. LinkBench: a database benchmark based on the facebook social graph. In *ACM SIGMOD'13* (2013), pp. 1185–1196.
[11] Barford, P., and Crovella, M. Generating representative web workloads for network and server performance evaluation. *ACM SIGMETRICS Performance Evaluation Review 26*, 1 (1998), 151–160.
[12] Beran, J., Sherman, R., Taqqu, M. S., and Willinger, W. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications 43*, 2/3/4 (1995), 1566–1579.
[13] Bocchi, E., Drago, I., and Mellia, M. Personal cloud storage benchmarks and comparison. *IEEE Transactions on Cloud Computing* (2015).
[14] Casas, P., Fischer, H. R., Suette, S., and Schatz, R. A first look at quality of experience in personal cloud storage services. In *IEEE ICC'13* (2013), pp. 733–737.
[15] Chard, K., Caton, S., Rana, O., and Bubendorfer, K. Social cloud: Cloud computing in social networks. In *IEEE CLOUD'10* (2010), pp. 99–106.
[16] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with YCSB. In *ACM SoCC'10* (2010), pp. 143–154.
[17] CrashPlan. https://www.crashplan.com, 2017.
[18] Crovella, M. E., and Bestavros, A. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking 5*, 6 (1997), 835–846.
[19] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with cfs. In *ACM SIGOPS Operating Systems Review* (2001), vol. 35, pp. 202–215.
[20] Difallah, D. E., Pavlo, A., Curino, C., and Cudre-Mauroux, P. OLTP-Bench: An extensible testbed for benchmarking relational databases. *VLDB Endowment 7*, 4 (2013).
[21] Drago, I., Bocchi, E., Mellia, M., Slatman, H., and Pras, A. Benchmarking personal cloud storage. In *ACM IMC'13* (2013), pp. 205–212.
[22] Drago, I., Mellia, M., M Munafo, M., Sperotto, A., Sadre, R., and Pras, A. Inside dropbox: understanding personal cloud storage services. In *ACM IMC'12* (2012), pp. 481–494.
[23] Feitelson, D. G. *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.
[24] Fisher, D., Smith, M., and Welser, H. T. You are who you talk to: Detecting roles in usenet newsgroups. In *IEEE HICSS'06* (2006), vol. 3, pp. 59b–59b.
[25] Forbes. Roundup of cloud computing forecasts and market estimates. http://www.forbes.com/sites/louiscolumbus/2016/03/13/roundup-of-cloud-computing-forecasts-and-market-estimates-2016, 2016.

[26] GARCIA LOPEZ, P., MONTRESOR, A., EPEMA, D., DATTA, A., HIGASHINO, T., IAMNITCHI, A., BARCELLOS, M., FELBER, P., AND RIVIERE, E. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review 45*, 5 (2015), 37–42.

[27] GOLAB, W., RAHMAN, M. R., AUYOUNG, A., KEETON, K., AND GUPTA, I. Client-centric benchmarking of eventual consistency for cloud storage systems. In *IEEE ICDCS'14* (2014), pp. 493–502.

[28] GONCALVES, G., DRAGO, I., COUTO DA SILVA, A. P., BORGES VIEIRA, A., AND ALMEIDA, J. M. Modeling the dropbox client behavior. In *IEEE ICC'13* (2014), pp. 1332–1337.

[29] GONÇALVES, G., DRAGO, I., DA SILVA, A. P. C., VIEIRA, A. B., AND ALMEIDA, J. M. The impact of content sharing on cloud storage bandwidth consumption. *IEEE Internet Computing 20*, 4 (2016), 26–35.

[30] GONÇALVES, G. D., DRAGO, I., VIEIRA, A. B., COUTO DA SILVA, A. P., ALMEIDA, J. M., AND MELLIA, M. Workload models and performance evaluation of cloud storage services. *Computer Networks* (2016).

[31] GRABSKI, F. *Semi-Markov processes: Applications in system reliability and maintenance*. Elsevier, 2014.

[32] GRACIA-TINEDO, R., GARCÍA-LÓPEZ, P., GÓMEZ, A., AND ILLANA, A. Understanding data sharing in private personal clouds. In *IEEE CLOUD'16*, pp. 392–399.

[33] GRACIA-TINEDO, R., HARNIK, D., NAOR, D., SOTNIKOV, D., TOLEDO, S., AND ZUCK, A. SDGen: Mimicking datasets for content generation in storage benchmarks. In *USENIX FAST'15* (2015), pp. 317–330.

[34] GRACIA-TINEDO, R., SÁNCHEZ-ARTIGAS, M., AND GARCÍA-LÓPEZ, P. Cloud-as-a-gift: Effectively exploiting personal cloud free accounts via REST APIs. In *IEEE CLOUD'13* (2013), pp. 621–628.

[35] GRACIA-TINEDO, R., SÁNCHEZ-ARTIGAS, M., MORENO-MARTÍNEZ, A., COTES, C., AND GARCÍA-LÓPEZ, P. Actively measuring personal cloud storage. In *IEEE CLOUD'13* (2013), pp. 301–308.

[36] GRACIA-TINEDO, R., SÁNCHEZ-ARTIGAS, M., RAMÍREZ, A., MORENO-MARTÍNEZ, A., LEÓN, X., AND GARCÍA-LÓPEZ, P. Giving form to social cloud storage through experimentation: Issues and insights. *Future Generation Computer Systems 40* (2014), 1–16.

[37] GRACIA-TINEDO, R., TIAN, Y., SAMPÉ, J., HARKOUS, H., LENTON, J., GARCÍA-LÓPEZ, P., SÁNCHEZ-ARTIGAS, M., AND VUKOLIC, M. Dissecting UbuntuOne: Autopsy of a global-scale personal cloud back-end. In *ACM IMC'15* (2015), pp. 155–168.

[38] HU, W., YANG, T., AND MATTHEWS, J. The good, the bad and the ugly of consumer cloud storage. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 110–115.

[39] JIN, Y., LIU, M., MA, X., LIU, Q., LOGAN, J., PODHORSZKI, N., CHOI, J. Y., AND KLASKY, S. Combining phase identification and statistic modeling for automated parallel benchmark generation. In *ACM SIGMETRICS'15* (2015), p. In press.

[40] KREITZ, G., AND NIEMELA, F. Spotify–large scale, low latency, p2p music-on-demand streaming. In *IEEE P2P'10* (2010), pp. 1–10.

[41] KRISHNAMURTHY, D., ROLIA, J., MAJUMDAR, S., ET AL. A synthetic workload generation technique for stress testing session-based systems. *IEEE Transactions on Software Engineering 32*, 11 (2006), 868–882.

[42] LI, A., YANG, X., KANDULA, S., AND ZHANG, M. CloudCmp: comparing public cloud providers. In *ACM IMC'10* (2010), pp. 1–14.

[43] LI, J., AND DABEK, F. F2f: Reliable storage in open networks. In *IPTPS'06* (2006).

[44] LI, Z., JIN, C., XU, T., WILSON, C., LIU, Y., CHENG, L., LIU, Y., DAI, Y., AND ZHANG, Z.-L. Towards network-level efficiency for cloud storage services. In *ACM IMC'14* (2014), pp. 115–128.

[45] LI, Z., WILSON, C., JIANG, Z., LIU, Y., ZHAO, B. Y., JIN, C., ZHANG, Z.-L., AND DAI, Y. Efficient batched synchronization in dropbox-like cloud storage services. In *ACM/IFIP/USENIX Middleware'13*. 2013, pp. 307–327.

[46] LOPEZ, P.-G., SANCHEZ-ARTIGAS, M., TODA, S., COTES, C., AND LENTON, J. Stacksync: Bringing elasticity to dropbox-like file synchronization. In *ACM Middleware'14* (2014), pp. 49–60.

[47] MAGER, T., BIERSACK, E., AND MICHIARDI, P. A measurement study of the wuala on-line storage service. In *IEEE P2P'12* (2012), pp. 237–248.

[48] MAIA, M., ALMEIDA, J., AND ALMEIDA, V. Identifying user behavior in online social networks. In *ACM SocialNets'08* (2008), pp. 1–6.

[49] NORCOTT, W. D., AND CAPPS, D. IOzone filesystem benchmark. http://www.iozone.org.

[50] POUWELSE, J., GARBACKI, P., EPEMA, D., AND SIPS, H. The bittorrent p2p file-sharing system: Measurements and analysis. In *IPTPS'05* (2005), pp. 205–216.

[51] ROWSTRON, A., AND DRUSCHEL, P. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review* (2001), vol. 35, pp. 188–201.

[52] RUFFO, G., SCHIFANELLA, R., SERENO, M., AND POLITI, R. 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*, IEEE, pp. 77–86.

[53] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. Measurement study of peer-to-peer file sharing systems. In *Electronic Imaging* (2001), pp. 156–170.

[54] SCHMIDT, M., HORNUNG, T., LAUSEN, G., AND PINKEL, C. SP^2Bench: a SPARQL performance benchmark. In *IEEE ICDE'09* (2009), pp. 222–233.

[55] SELTZER, M., KRINSKY, D., SMITH, K., AND ZHANG, X. The case for application-specific benchmarking. In *USENIX HotOS'99* (1999), pp. 102–107.

[56] TARASOV, V., BHANAGE, S., ZADOK, E., AND SELTZER, M. Benchmarking file system benchmarking: It *IS* rocket science. *USENIX HotOS'11* (2011).

[57] TARASOV, V., KUMAR, S., MA, J., HILDEBRAND, D., POVZNER, A., KUENNING, G., AND ZADOK, E. Extracting flexible, replayable models from large block traces. In *USENIX FAST'12* (2012), p. 22.

[58] TARASOV, V., MUDRANKIT, A., BUIK, W., SHILANE, P., KUENNING, G., AND ZADOK, E. Generating realistic datasets for deduplication analysis. In *USENIX ATC'12* (2012), pp. 1–12.

[59] TRAEGER, A., ZADOK, E., JOUKOV, N., AND WRIGHT, C. P. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage 4*, 2 (2008), 5.

[60] WANG, L., ZHAN, J., LUO, C., ZHU, Y., YANG, Q., HE, Y., GAO, W., JIA, Z., SHI, Y., ZHANG, S., ZHENG, C., LU, G., ZHAN, K., LI, X., AND QIU, B. BigDataBench: A big data benchmark suite from internet services. In *IEEE HPCA'14* (2014), pp. 488–499.

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

[62] ZHENG, Q., CHEN, H., WANG, Y., DUAN, J., AND HUANG, Z. COSBench: A benchmark tool for cloud object storage services. In *IEEE CLOUD'12* (2012), pp. 998–999.

Raúl Gracia-Tinedo is a postdoc in the Architectures and Telematic Services Research Group at Universitat Rovira i Virgili (URV), Spain. His research interests include distributed storage management, cloud computing, and performance evaluation of systems. Gracia-Tinedo received his PhD in computer engineering from URV in 2015. He received the Best Dataset Award at the 2015 ACM Sigcomm Internet Measurement Conference. Contact him at raul.gracia@urv.cat.

Chenglong Zou is a MSc student of computer engineering and security at University Rovira i Virgili (URV), Tarragona, Spain, in the Architectures and Telematic Services (AST) research group. He has participated in the FP7-CloudSpaces project. His research interests are related to distributed storage and cloud computing. Contact him at chenglong.zou@urv.cat.

Marc Sánchez-Artigas received the PhD degree in 2009 from the Universitat Pompeu Fabra, Barcelona, Spain. During his PhD studies, he worked at Ecole Polytechnique Fédérale de Lausanne (EPFL). In the same year he joined the Universitat Rovira i Virgili, where he currently works as Associate Professor. He received the Best Paper Award from IEEE LCN'07 and the Best Dataset Award from ACM IMC'15. He has published over 60 articles. Now he is actively involved in the coordination of several Spanish and European projects in Cloud storage. Contact him at marc.sanchez@urv.cat.

Pedro García-López is a professor in the Computer Engineering and Mathematics Department at URV, where he also leads the Architectures and Telematic Ser vices Research Group. His research interests include distributed systems, peer-to-peer systems, cloud storage, software architectures, and middleware and collaborative environments. García-López has a PhD in computer science from University of Murcia. Contact him at pedro.garcia@urv.cat.