MUBox: Multi-User Aware Personal Cloud Storage

Michael Nebeling, Matthias Geel, Oleksiy Syrotkin and Moira C. Norrie
Department of Computer Science, ETH Zurich
{nebeling,geel,norrie}@inf.ethz.ch

ABSTRACT

Personal cloud services such as Dropbox are used increasingly to support collaborative work, even though they typically have poor support for tracking files and users' activities and collaborators often rely on other communication channels to be notified of changes. We present a meta-cloud storage service, *MUBox*, that, independent of a particular cloud storage service, provides improved support for collaboration. First, users can switch to activity views that list user activities rather than files, which is an example of an increasingly available feature in popular cloud storage clients. Second, MUBox introduces multi-user aware folder views that embed information on the last changes performed by collaborators. These folder views are enhanced based on a new concept of shadow files which act as placeholders for files that have been moved or renamed. A user study (N=16) with realistic folder exploration tasks shows that activity views have a significant effect on the accuracy and confidence of users in workspace awareness tasks, while shadow files significantly improve the speed, accuracy and confidence of users in traceability tasks. We describe how existing services could implement these features as well as a new concept for *voting* on changes to shared folders that could improve asynchronous collaboration.

Author Keywords

personal cloud storage; collaborative work; workspace awareness; file traceability.

ACM Classification Keywords

H.5.2. User Interfaces: Interaction styles

INTRODUCTION

Personal cloud storage services were developed to enable users to synchronise their personal data across devices. However, they are nowadays often used for sharing files with friends and colleagues and even for collaborative work in professional settings. Many cloud services let a user share an entire folder with one or more other users. The folder becomes visible in the other users' workspaces, where they can add or modify files and subfolders. While there are some differences in how various cloud services treat the original owner of the file and the rest of the group, there is no explicit role management and any user can perform delete, rename or move actions. As the number of collaborators grows, it becomes more difficult to be aware of changes and track activities.

Based on our own experiences of using a shared Dropbox folder as the basis for collaboration in a European project, we identified a number of common problems. The project involves more than 25 people from 8 different organisations spread across Europe. The team decided to use a shared Dropbox folder as a repository for documentation, deliverables and various forms of media. Given the fairly large number of contributors, the level of activity in the shared folder is high, and it is difficult to track the actions of the other project members. Not everyone is able to closely follow the evolution of the shared folder and the more time that elapses since the last check, the more challenging it becomes to understand the changes. Specifically, we encountered five main problems:

1. **Lack of overview of specific collaborators' activities**: It is difficult to keep track of the activities of other users, especially when multiple users are responsible for changes.
2. **Tracking the origin of files**: When a new file or folder appears, it is not clear whether it has been uploaded (created), copied from another location, or just renamed. While some cloud storage providers list the user responsible for the last modification, the actual activity is often not clear.
3. **Tracking files that have disappeared**: Likewise, if a particular file or folder is not at its original location, it may have been deleted, moved or renamed.
4. **Hesitation to delete or otherwise modify a shared folder**: Even though everyone is entitled to make modifications, collaborators are often reluctant to do so and prefer to limit their activities to their own files. This problem can lead to an unwieldy folder structure over time [1, 14].
5. **Need for external communication channels**: Email is often used to alert users to activities.

In this paper, we present *MUBox*—a meta-cloud storage service that addresses these problems based on a small set of new features for asynchronous collaborative use. A key goal in designing MUBox was to alleviate Problem 1 by raising collaborators' workspace awareness which is integral and critical to successful collaborative work [5, 13]. We introduce multi-user aware folder views that embed information on the last changes of users. We specifically decided to focus on changes to file locations and filenames, rather than changes to file contents. We believe these constitute the most important activities that users need to become aware of in shared

CHI 2015, April 18–23 2015, Seoul, Republic of Korea

http://dx.doi.org/10.1145/2702123.2702233

folders. As another way of supporting this, we introduce activity views that are activity-centric folder views showing a history of users' activities rather than the typical list of files.

We tackled Problems 2 and 3 by supporting the user's ability to establish a provenance relationship between files using backward and forward traceability mechanisms. We are primarily concerned with coarse-grained, file-level traceability where traces point to old or new locations of the files. Summarising the history of a file or identifying all provenance relationships based on different versions as done in [10] was deliberately left out of scope. In a shared folder, delete, rename and move activities are more disruptive to collaborative work than if a file was updated or copied. We develop the concept of shadow files to show relationships between files and users' activities that would otherwise not be clearly visible. Shadow files are placeholders for files that have been moved or renamed, so that no files appear missing to other users. Backward traces will point to previous locations of the files. Thus, tracking the origin of files should become easier.

We approached Problems 4 and 5 by introducing voting mechanisms with automatic notifications as a way of encouraging all users to modify shared folders. This can be done using two different strategies, inspired by common practices in version control systems. In the first strategy, a user can perform changes that may be reverted if other users reject them. In the second strategy, users may suggest changes, but no changes will be carried out unless approved by other users. In either case, no external communication is necessary as users will automatically be alerted via in-built notifications. In particular, users can delete files without fear that someone may still need the file or may not be aware of the deletion.

We begin by presenting the background to our work, followed by a detailed description of MUBox. Then we present the implementation and define a cloud service abstraction to make sure that our proposal is independent of current providers and make it easier for future developers to implement the concepts. Finally, we present an initial user study with 16 participants that evaluates activity views and shadow folders. The results show that activity-centric folder views significantly increase users' confidence and accuracy, but not efficiency, in understanding the actions of collaborators. On the other hand, shadow files significantly improve users' confidence, accuracy and speed in understanding file-to-file relationships

BACKGROUND

Insufficient visibility of the activities of collaborators is one of the major problems in information sharing [18]. A previous study [17] reports that the lack of notifications and lack of visibility are some of the biggest problems in file sharing systems. More recent work [16] investigates user experience with services such as Dropbox and Google Docs for multiple collaboration scenarios. The interviewed users complained about insufficient visibility of collaborators' activities and difficulties in being aware of the parts of the shared folder for which one is responsible. In [12], it was reported that users experience problems understanding the functionality of cloud providers because they have inaccurate conceptual models of the cloud. The authors propose design guidelines that cloud services should adhere to in order to become more accessible to users: notifying the user when a given file was last accessed; adding visual status indicators of syncing; phrasing user messages and labels accurately and unambiguously. The authors also argue that vendors need to work on a new conceptual model of cloud storage instead of trying to exploit a familiar paradigm, such as a file system or web-based collaboration application, in a new setting. Another study [14] investigating users' behaviour in group information repositories revealed that users almost exclusively touch files they have created themselves and are particularly reluctant to delete files that could be useful to someone else in the future. Such behaviour can result in clutter and frustrate users. One approach to overcome the particular problem of deletion was explored in GrayArea [1], which extends the folder view with a grey area at the bottom where users may drag unimportant files. GrayArea was shown to effectively reduce folder clutter, but requires major changes to common file system user interfaces and workflows.

Workspace awareness is known to be an important requirement for supporting productivity in collaborative work [5]. In the area of distributed software engineering many different ways of promoting awareness have been studied with the common aim of enhancing a person's understanding of the activities of collaborators, as well as increasing knowledge about tasks and artefacts [13]. Many of these are text-based: email, mailing lists, chats (instant messaging or group chats), RSS feeds and wikis. Collaborators frequently use version check-in logs in configuration management systems and commit logs in revision control systems to familiarise themselves with the actions of other users. Many proposed solutions use some form of tagging. For example, Jazz.net, CASS [9] and Augur [7] all use tags to help developers navigate the code and subscribe to notifications on topics of interest.

Others mainly focused on exploring the relations between software project entities and visualising them, e.g. Rational Team Concert [3]. Here, visualisations provide a view of the project that users can quickly grasp, thereby improving traceability of software components. The authors argue that this, in turn, helps to increase developers' awareness. Another approach is to track developers' interactions. For example, FASTDash [2] provides information about which class and method a given developer has currently checked out and is working on, signalling potential conflicts. Information about developer interactions and artefact relations can be combined. For example, Team Tracks [4] marks more frequently accessed parts of the code as more important. Further, in Mylyn [11], a notion of time was added so that more recently accessed artefacts are given an increased weighting.

In the domain of personal file sharing, providing information on file provenance, i.e. on the history of ownership and operations performed on the file [8, 15], is considered a key factor for improving awareness. Various systems that capture and manage provenance information have been developed, some of them application-specific. For example, Chimera [6] was developed for use with large scientific data repositories, storing data derivation procedures and derived

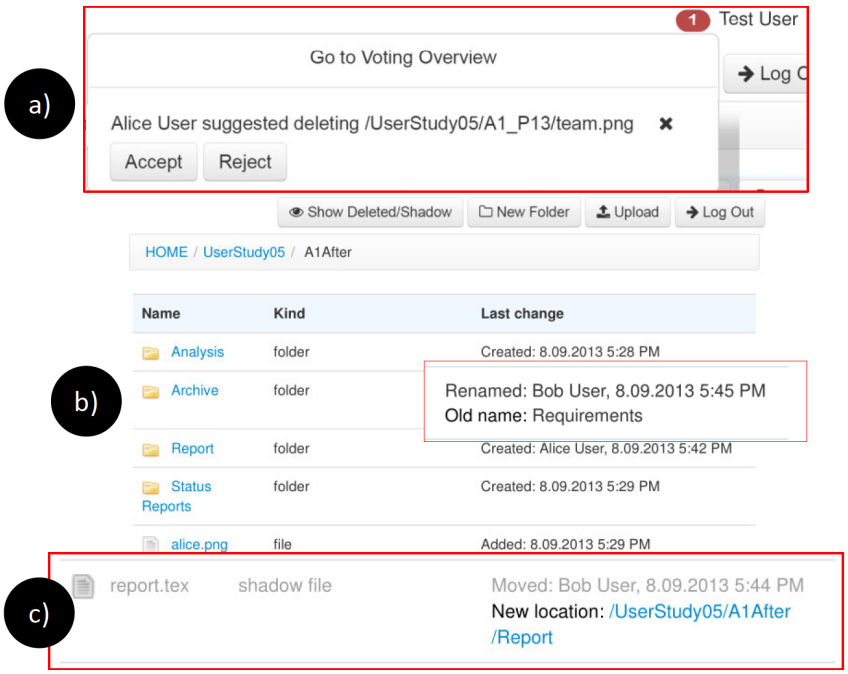


Figure 1. Main features of MUBox: a) notifications on changes *suggested* by other users, b) multi-user aware folder views highlighting last changes by others, e.g. Bob renamed Requirements to Archive, and c) shadow file with trace, e.g. Bob moved report.tex to the Report folder.

data and letting users query the data and provenance metadata using a domain-specific language. In contrast, FiPS [15] is an application-independent provenance system that could be added on top of any conventional file system and implemented as an operating system kernel module. It manages files as well as file provenance information by intercepting file system calls used to collect metadata that let it recreate the history of a file. Such provenance data was also used by Karlson et al. [10] who developed a copy-aware file system by hooking into the operating system and office applications to capture what they refer to as "copy events", e.g. Save As, Attach and Copy/Paste. Using this information, they constructed a new folder visualisation that included provenance relationships on different versions of files. Users found the new folder visualisation especially useful to make sense of unfamiliar data, e.g. their co-worker's folder.

Our first step was to review some of the most popular personal cloud services: Dropbox, Google Drive and OneDrive to identify typical shortcomings. As an example, in some services, when a user renames a file, it is regarded as newly created, and the file with the old name is displayed as deleted. Possible problems encountered by collaborators are, firstly, that the old file is marked as deleted, thus possibly confusing the rest of the team, and, secondly, that there is no apparent connection between the old and new filenames. Users would need to restore the old file first and compare it with the new file to see if they actually have the same content. Moreover, for files moved by another user to a different folder, other folders would have to be checked manually or a global search had to be performed.

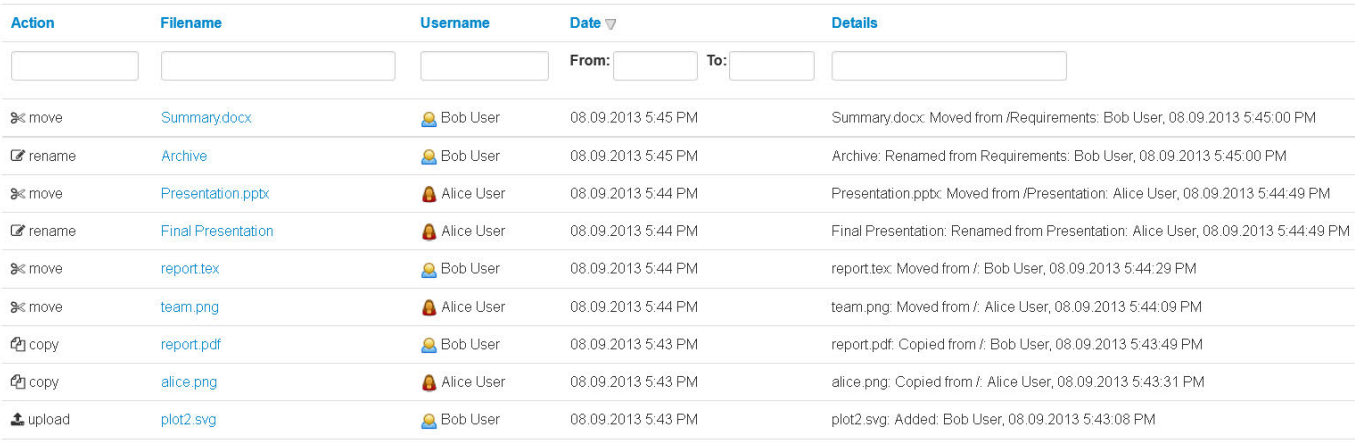
In summary, cloud storage providers have only begun to add support for collaboration and different providers have opted for different techniques. Google Drive has presence indicators, chatting capabilities and an activity stream. Dropbox allows users to inform collaborators via email without leaving the web interface and also provides a list of events in all of the user's workspace or in a specific shared folder. Given the aforementioned shortcomings, we decided to experiment with new features to improve the folder sharing experience and make awareness of the other users' actions and file traceability easier to achieve.

MUBOX

MUBox (Figure 1) is a meta-cloud storage service with a web-based front-end through which users can access Dropbox and Google Drive accounts. While it resembles common interfaces, it is independent of a particular cloud storage solution and other providers could be easily integrated. MUBox is a lightweight, yet highly configurable, system allowing individual features to be turned on and off as required.

After a user logs in, they are presented with the folder view shown in Figure 1b). A grid with three columns displays the folder structure. The first column, *Name*, contains a list of folders and files ordered alphabetically. For each entry in the list, the second column specifies the *Kind* in terms of four categories: regular file, deleted file, shadow file and shared folder. The *Last change* column can display different information depending on the experimental settings. The minimum functionality is to display the username of the user who made the last change to the entry. If shadow folders and traces are enabled, the column also displays the last operation performed on the file and may also show a backward or forward trace. In the case that a user other than the current user was responsible for the change, that user will also be listed.

If the user clicks on a file row in the grid, the grid header displays a row of buttons signifying file operations. Alterna-



Action	Filename	Username	Date	Details
			From: To:	
move	Summary.docx	Bob User	08.09.2013 5:45 PM	Summary.docx: Moved from /Requirements: Bob User, 08.09.2013 5:45:00 PM
rename	Archive	Bob User	08.09.2013 5:45 PM	Archive: Renamed from Requirements: Bob User, 08.09.2013 5:45:00 PM
move	Presentation.pptx	Alice User	08.09.2013 5:44 PM	Presentation.pptx: Moved from /Presentation: Alice User, 08.09.2013 5:44:49 PM
rename	Final Presentation	Alice User	08.09.2013 5:44 PM	Final Presentation: Renamed from Presentation: Alice User, 08.09.2013 5:44:49 PM
move	report.tex	Bob User	08.09.2013 5:44 PM	report.tex: Moved from /: Bob User, 08.09.2013 5:44:29 PM
move	team.png	Alice User	08.09.2013 5:44 PM	team.png: Moved from /: Alice User, 08.09.2013 5:44:09 PM
copy	report.pdf	Bob User	08.09.2013 5:43 PM	report.pdf: Copied from /: Bob User, 08.09.2013 5:43:49 PM
copy	alice.png	Alice User	08.09.2013 5:43 PM	alice.png: Copied from /: Alice User, 08.09.2013 5:43:31 PM
upload	plot2.svg	Bob User	08.09.2013 5:43 PM	plot2.svg: Added: Bob User, 08.09.2013 5:43:08 PM

Figure 2. Activity views are shown as a grid that can be sorted and filtered by *Action*, *Filename*, *Username*, *Date*, *Details*. Possible actions are: `newfolder`, `upload`, `move`, `copy`, `rename`, `delete`. Details describe activities using verbs, "Created" (for new folder), "Added" (for file upload), "Moved", "Copied", "Renamed", "Deleted", along with the responsible users.

tively, a user may right-click the file row and display a context menu listing the same operations. Usual operations for a file are: download, show versions, delete, rename, cut and copy. Cutting or copying a file followed by pasting let the user move or copy the file to a different folder. For a folder, the operations are: share folder, delete, rename, cut and copy. Clicking a hyperlink displaying the filename starts a download of the file. Clicking a folder name opens that folder.

Below we describe the three main ideas implemented in MUBox to improve awareness, traceability and asynchronous collaboration: activity views, shadow files and voting. One key design goal was to maintain the simplicity. All the proposed collaboration features are either toggable (activity views/shadow files) or can be configured on a per-folder basis (voting). The design rationale is that the new awareness/traceability features should only be shown if the user has a concrete information need, but otherwise they should not increase the complexity of the system in place.

Activity-centric Folder Views

As a first step towards improving awareness, we introduced activity views (Figure 2) that list users' activities rather than files and folders. The user can switch to the corresponding activity view from each folder in order to access the following activity information: the name of the activity, the affected file or folder, the user who made the change, the time of the change, and any relevant details. In the case of a rename, move or copy, the details would include either the source or the destination file or folder. When the user hovers the mouse cursor over the filename, a tooltip displays the relative path to the file at the time the operation was completed. Our design of activity views differs from Dropbox's events or Google Drive's activity stream in two important ways.

First, it is essentially a different view of the folder rather than a log file based on journal-style text entries similar to `Bob renamed Paris.jpg to Eiffel Tower.jpg on 16.04.2014`. Activity-centric folder views are based on grids that show one operation per row and details of each operation in dedicated columns. The grid is both searchable and sortable by column. The search fields for each column allow the user to filter the rows, while sorting controls the order of rows. Since MUbox can distinguish between moved/renamed and newly uploaded files, a user may first filter by action, e.g. `upload`, and then by date to quickly get an overview of files that have been uploaded at a given time.

Second, we decided to present the activity information starting from a given folder and including all subfolders. We think this is a good trade-off as opposed to showing the activities for the entire workspace or limiting the view to only a single folder. The deeper in the folder hierarchy the user descends, the more specific the activity view becomes.

Shadow Files with Backward/Forward Traces

To keep track of deleted files (or folders), some providers such as Dropbox show a placeholder together with the information, "deleted file", at the previous location. We propose to extend this concept to track renamed or moved files and provide a link to the new location. Specifically, MUBox introduces the concepts of shadow files and forward and backward traces, allowing users to explore file provenance.

From the user's point of view, a shadow file is a placeholder used to indicate that a file has been renamed or moved to a different folder. In existing cloud storage services, such files are usually shown as deleted, which misrepresents the activities that resulted in the change and may therefore lead to confusion. Therefore, we explicitly distinguish shadow files from deleted files with the goal of making it clear to the user that a given file is in fact not deleted and exists under a different name or path in the shared folder. The *Show Deleted/Shadow* button shown in Figure 1 lets the user show or hide deleted and shadow files. MUBox lists all files in the same folder view, but visually separates regular files from deleted and shadow files by greying out the latter similar to how many operating systems show hidden files on demand.

Our notion of traces is closely related to shadow files. Figure 3 illustrates the two concepts. Forward traces are attributes of a shadow file that point to the new name or new location of

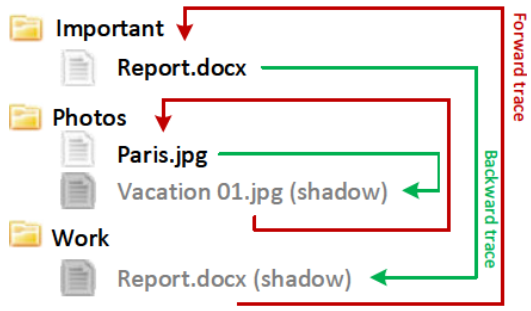
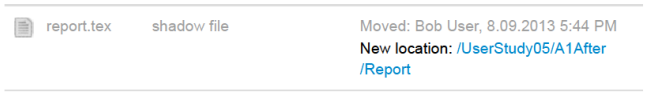
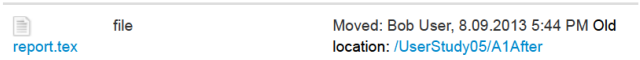


Figure 3. Shadow files are placeholder files shown in gray as part of the folder view. The concept above illustrates forward and backward traces. The file `Vacation 01.jpg` was renamed to `Paris.jpg`, and the forward trace of the corresponding shadow file `Vacation 01.jpg` points to `Paris.jpg`. The file `Report.docx` was moved to the folder `Important`, and a backward trace points back to `Work`.



(a) Forward trace



(b) Backward trace

Figure 4. After moving `report.tex` to a subfolder `Report`, a shadow file is shown at the original location with forward trace and the file at the new location shows a backward trace.

that file and occur after a rename or move operation. Backward traces are links from a current file to its source file. The source file may have been moved, renamed or copied. Thus, using our terminology, a backward trace is either a connection between a regular and a shadow file (in the case of moving or renaming) or between two regular files (in the case of copying). We decided for both a forward trace and a backward trace to provide the user with as complete information as possible. Entering a folder and glancing at the trace information should make it clear whether *new* files have originated from another location or are indeed newly uploaded files. Therefore, preserving both kinds of traces as opposed to only forward traces, makes navigation within the workspace more flexible. A trace is visualised simply as text and a hyperlink in the *Last change* column that tells the user the old or the new location of a file. Forward traces are displayed next to the shadow file to which they correspond (Figure 4(a)). Backward traces are displayed next to the file that was created as a result of a rename, move or copy operation (Figure 4(b)).

Shared Folders with Voting Strategies and Schemes

MUBox also aims to encourage more active asynchronous collaboration. To this end, we propose voting capabilities with an integrated notification system so that users can perform file operations without the fear of interfering with their collaborators' work. We differentiate between voting strategy (determines when a change is applied) and a voting scheme (determines how changes can be rejected/approved). Depending on the voting strategy, the change may be performed immediately, with the possibility of reverting it later, or performed later after approval. Additionally, whenever a user makes a disruptive change (i.e. delete, rename or move) to a file in a shared folder with voting enabled, the other users will receive a notification and may vote immediately or postpone their decision.

We believe that introducing such a voting system will not only increase the visibility of collaborators' activities, but also function as a safeguard and encourage team members to be more active in carrying out folder maintenance tasks, without waiting for the original owners to clean up their files and folders. We refer to the process when votes are collected and evaluated as the "voting process".

Since voting is a rather heavy process in contrast to lightweight aspects of cloud file sharing, it should only be implemented where necessary, for example, for folders that contain deliverables or business-critical files. As a consequence, it can be configured on a per-folder basis and is disabled by default. We decided against taking file and folder ownership into account in the voting process. First, a straightforward approach quickly breaks down when it comes to shared folders since an owner of a folder, i.e. the user that created it, may not necessarily be the one responsible for its content or the most active user of that folder. Second, popular services such as Dropbox have a very basic ownership model typically limited to top-level shared folders rather than subfolders and files, and explicit user actions are required to transfer ownership. For these reasons and based on our experiences with the EU project, we felt that an additional access control layer was not required and might actually work against facilitating easy collaboration, especially among equal project partners and with rapidly evolving folder structures.

Voting Strategies

A voting process is regulated by a voting strategy. We have developed the following two strategies: **1) perform a file operation subject to confirmation**, and **2) suggest a file operation subject to approval**. Strategy 1 performs an operation immediately, notifies other users and starts a voting process. Only if they vote against the operation, is it reverted. We have designed this strategy primarily for small teams with the intention of having minimal impact on existing workflows. Users can still work with their files as usual and the changes become instantly visible. However, notifications and voting in that case can help prevent major changes from going unnoticed and to correct mistakes. Strategy 2 allows any user to suggest an operation. The other users are then notified and the operation is only performed if approved. This strategy is well-suited to model more rigid workflows, for example weekly reviews of changes.

The difference between the two strategies is also reflected in the user interface. If a file is to be renamed with Strategy 1, the context menu option for files says "Rename (voting)". With Strategy 2, "Suggest rename" is displayed instead. Outside of voting-enabled folders, the menu item is simply "Rename". This ensures that users are aware of a) whether voting is enabled and b) if it is, which strategy is in effect. In the following, we will refer to the first strategy as *confirmation* strategy and to the second as *approval* strategy.

Scheme	Strategy	Default Action
Veto (*with* time constraint)	Either	Accept
Percentage (*with* time constraint)	Confirmation	Accept
	Approval	Reject
Percentage (*without* time constraint)	Either	None

Table 1. Voting scheme/strategy combinations ordered from least to most intrusive to optimistic workflows (accepting most voting processes).

Voting Schemes

Complementary to the voting strategies, we offer two distinct voting schemes. A voting scheme is a rule governing the decision on users' votes and when a voting process can be closed and the change either accepted or rejected. We derived the following schemes, which are inspired by well-known decision-making principles: **i) percentage with or without time constraint** and **ii) veto with time constraint**.

The percentage scheme is a variant of the majority vote that *accepts* a vote if the majority of a certain *percentage* of users vote in favour of the change. With that scheme, one can specify the percentage of users that need to participate in a majority vote before it is closed and evaluated. The core idea is to be able to lower the overhead which voting processes often carry with them. For example, a team-wide majority vote (100 % participation rate) might be unproductive in large teams because it can take a long time to collect all the votes. By contrast, having a lower percentage allows a few active users, e.g. core team members, to vote on changes quickly.

In contrast, the veto scheme *rejects* an action as soon as a single person with the power of veto votes against it. In our design of this scheme, any user has the veto power and can thus revert a change immediately. Veto power is particularly useful for teams with a designated leader who assumes the role of guardian or maintainer of a shared folder. If this is the case, the other team members may choose to simply mute the notifications by MUBox for that particular folder.

For both voting schemes, a configurable time constraint specifies a timeout after which a voting process is aborted with a default action. That default action depends on the particular combination of voting strategy/scheme chosen and is summarised in Table 1.

The percentage scheme combined with the confirmation strategy has the default action to accept, whereas the approval strategy defaults to reject if no consensus was reached within the given timeout. The implicit acceptance of changes in the former combination assumes an optimistic workflow and is most effective if most voting processes are likely to get accepted. In this case, a rejection may cause a major interruption to the workflow, which can be used intentionally to send a strong signal to collaborators that there is disagreement. With the veto scheme, the default action is always to accept a change, regardless of the current strategy. Due to its nature, the veto scheme works only in conjunction with a time constraint, whereas the percentage scheme can also be used without any timeout. In the latter case, a decision can only be reached after enough users have voted, making it the most rigid voting process available.

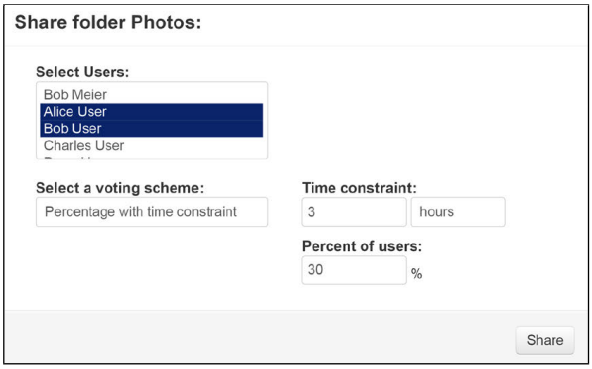


Figure 5. Voting scheme selection while sharing a folder

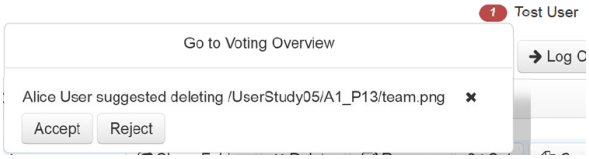


Figure 6. Notification badge and a voting alert dropdown

Sharing Folders and Ownership

MUBox allows any folder to be shared via a context menu. Figure 5 shows the modal dialogue presented to the user at the time of sharing the folder "Photos". The figure shows the selection of Alice and Bob as team members for this particular folder. The selected voting scheme is "Percentage with time constraint" and at least 30 % of the users have to vote on changes within 3 hours. The person who initially shared the folder (the owner) can later change the voting scheme and add or remove collaborators. The other collaborators do not have such privileges.

The choice of the "right" strategy/scheme is ultimately a social rather than a technical issue. Consequently, we do not impose or mandate one particular combination, but rather strive to provide the necessary configuration options to allow team leaders to model a wide range of organisational processes.

Notifications and Voting

After a user initiates a voting process by performing a disruptive operation, the user's vote is saved as a vote in favour of the change and notifications are sent to the collaborators. If they log into MUBox during the voting process, a red notification badge in the upper right corner of the screen informs them about ongoing votes. They can open the notification dropdown and choose to accept or reject the operation, as seen in Figure 6. After the change is accepted or rejected, a notification is sent to the vote initiator. MUBox can also show an overview of all pending voting processes in which the current user can cast a vote or exercise their right of veto.

IMPLEMENTATION

MUBox is implemented as a web application to facilitate deployment in distributed multi-user settings and provide a consistent user experience across different platforms. It com-

Method Name	Description
`getDeltaData(<delta cursor>)`	Retrieves a list of changes.
`createFolder(<path>)`	Creates a folder.
`shareFolder(<path>, <owner>, <user>)`	Shares a folder with another user.
`upload(<parent path>, <file name>)`	Uploads a file to folder.
`copyOrMove(<action>, <from path>, <to path>)`	Copies or moves a file/folder.
`rename(<old path>, <new path>)`	Renames a file/folder.
`delete(<path>)`	Deletes a file/folder.
`undelete(<file>)`	Restores a file/folder.

Table 2. Cloud abstraction methods

prises two components: 1) a service front-end on the client-side, and 2) a server-side web service that hosts the cloud abstraction layer and provides a persistence layer for the application. The service front-end is a client-side JavaScript application developed using AngularJS[1] and UI Bootstrap[2]. The client communicates with the server via HTTP calls to a REST-style interface using JSON for data exchange. The implementation of the web service is based on Spark[3]. Spark's primary abstraction is a route which maps HTTP methods (i.e. GET, POST, DELETE etc.) and URL patterns to callbacks that implement most of the business logic and calls to the cloud abstraction and persistence layer. To ease development, all layers have been implemented in Java and run within the same JVM as the web service. We will now explain each of the layers in more detail.

Cloud Abstraction Layer

To achieve a provider-independent solution, the *CloudStorage* interface provides a cloud storage abstraction for uniform access to vendor-specific protocols and API calls. This allows us to seamlessly switch between different cloud storage providers without having to adapt the web interface. The CloudStorage interface (without authorization) is summarised in Table 2. Each method additionally expects a user parameter which has been omitted for brevity. As proof of concept, we implemented the interface for Dropbox and Google Drive since both offer mature Java SDKs for their services and have gained widespread adoption.

Most interface methods are straightforward and can be easily mapped to existing cloud storage services. The *getDeltaData* method initialises the local copy of the cloud storage folder structure and keeps it synchronised with the underlying cloud storage provider. The first time this method is called, the implementation queries the cloud storage service for the entire folder structure of a user's account. The call also returns a token, i.e. delta cursor in Dropbox and change ID in Google Drive, that can be used in subsequent calls to get only the changes that have taken place since the last call.

A notable challenge arose when *shareFolder* had to be implemented. At the time of writing, Dropbox did not have a public API to share folders programmatically with other users. Even worse, folder metadata retrieved via the official Dropbox API neither indicated with whom this folder has been shared nor if the folder was shared at all. As a workaround, users have to share folders manually via the official Dropbox web interface and then specifically mark those folders as shared in MUBox. On the other hand, folder sharing is supported in Google Drive and the implementation therefore straightforward. Another limitation of the Dropbox API is the inability to restore an entire deleted folder. To restore a folder, one has to recursively restore the files originally stored in this folder and all its subfolders.

Another issues is the unification of file identifiers. In Dropbox, file paths are unique by design and used as identifiers. In Google Drive, however, paths are not unique and a global file id is used instead. We decided to adopt the Dropbox approach and access files and their metadata through our cloud abstraction layer using a combination of a unique path and user ID. To achieve interoperability with Google Drive, we internally maintain a mapping between paths and Drive's file ids, which are passed to Drive API calls when necessary.

As a consequence of adopting unique file paths, there cannot be more than one file with the same name in any given folder. Since this is the case in all major desktop file systems, such behaviour corresponds to the mental model that most end-users have of file structures and is an acceptable tradeoff.

Persistence Layer and Data Model

To realise the back-end of MUBox, we use MongoDB, a document-oriented NoSQL database. MongoDB stores data objects as JSON-like documents in binary format. Documents are grouped into collections which are mainly organisational, i.e. documents contained within them do not need to follow a specific schema. Having a flexible schema while experimenting with the data model allowed us to test our assumptions quickly. Moreover, data is exchanged as JSON, which facilitates integration with the MUBox web application because there is no need for serialisation.

For each user, we store a complete copy of the file hierarchy, i.e. metadata only, in the database. Each file node is stored as a single document in MongoDB. Whenever the folder structure is modified via the MUBox web interface, we update the metadata in the database and call the *CloudStorage* API to propagate the updates to the corresponding storage provider. We do not access the content of files in the cloud storage unless there is a user request to download a file.

To model hierarchical folder structures, the approach of materialised paths is used. For each filedata document, we store the complete path with / as separators including the filename. A single MongoDB query using a regular expression is suf-

[1] http://angularjs.org/
[2] http://getbootstrap.com/
[3] http://www.sparkjava.com/

ficient to select only the files in a given folder or all the descendants of a given folder. During normal interaction with a cloud storage client, users spend considerable time navigating the folder hierarchy. Therefore, looking up file metadata is one of the most frequent database queries. Using materialised paths not only makes querying easy and efficient, but also lets us use the same path in database queries as well as calls into the cloud storage APIs. If a folder is shared, the shared folder hierarchy is replicated for each user with whom the folder has been shared with. This approach made it easy to experiment with different views of the folder structure for each user if necessary.

EVALUATION

We conducted controlled experiments to evaluate users' efficiency, accuracy and confidence using activity views and shadow files with traces. For the voting system, we are planning a deployment of MUBox and a longitudinal study in the context of the EU research project mentioned earlier.

Method

We used a within-subjects design involving two experiments to evaluate 1) activity views and 2) shadow files and traces. Each experiment involved one task with the respective MUBox features turned on and one task with them turned off. We have refrained from direct comparison to existing personal cloud storage services and used MUBox in all tasks for better control. But the off conditions are similar to using Dropbox and hence provide a baseline. Also, we were only interested in comparing task performance within, not between, experiments. The off conditions used the same MUBox config but different tasks, and so results cannot be assumed to be equal.

Experiments and task order were rotated and counterbalanced to reduce carryover effects. Each task required participants to familiarise themselves with a shared folder structure different for each task. Before beginning the task, participants received a printout with a diagram of the folder structure. They were also shown an incomplete folder structure in the MUBox interface and asked to complete it so that it matched the structure on paper. In each task, completing the structure meant they had to create one empty folder and upload two files. The files to be uploaded were provided. The folder hierarchy completion was not part of the task per se, but it was our attempt to familiarise the users with the system and each of the shared folders. Then the actual study started. Assuming that the printout of the folder structure contained the initial hierarchy, the user was demonstrated what this folder structure looked like after two collaborators made changes to some files and folders. The collaborators were always two users with the names Alice and Bob. To save time, we pre-created the folder structure with the changes made. After participants saw the modified folder structure, they were given a questionnaire, and the timing started. Participants needed to use the available features in MUBox to fill in the questionnaire that tested their understanding of exactly what changes took place in the shared folder. When the users reported completion of the task, the timing stopped.

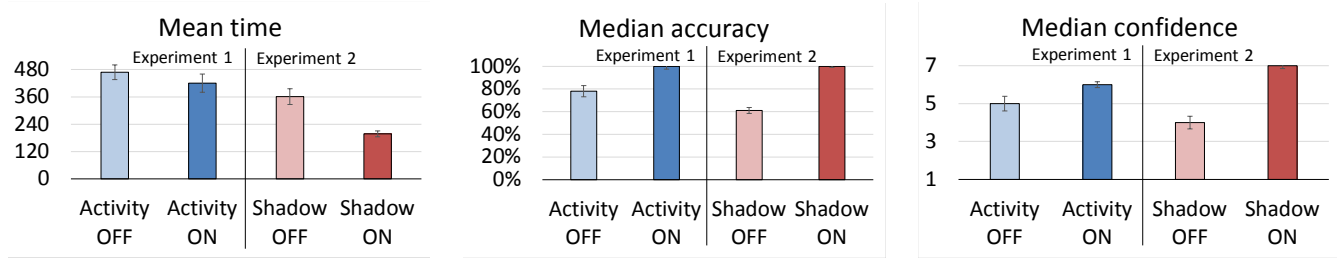
The goals of our experiments were to verify whether the features improved workspace awareness and file traceability. Our approach was to test the time it took to perform the task, the accuracy of the responses and the users' own estimates of their confidence, which they provided in post-task questionnaires. Because of the way the tasks were designed, we could argue that the time it takes to perform the task, as well as the user's confidence and accuracy while performing it, can be signs of whether the user is aware of the activities of the teammates and whether file history is traceable. Although we could obtain such indirect evidence, we cannot claim that the features actually improve awareness or traceability.

Evaluating Activity Views

We have argued that activity views may help users be more aware of others' activities and improve workspace awareness. We expected that the use of activity views would let participants do the task in less time, answer more questions correctly and be more confident in their answers. We developed an awareness questionnaire that contained the names of the collaborators, Alice and Bob, and asked participants to list the file operations each collaborator had performed. In both scenarios used for the tasks, each collaborator performed the following actions: creating a folder, uploading a file, renaming a folder with one file in it, copying a file and moving another file. Participants' answers were considered correct if they specified 5 correct operations per user, including the name of the operation (e.g., "copy") and the destination if applicable (for copying, moving, renaming). If the user listed other changes, they were ignored for the score. Participants received partial points if parts of the answers were correct. Shadow files were disabled for the duration of the two tasks. In the off condition without activity views, the user had to make educated guesses using the normal folder view. For example, if a file with a new name appeared in the shared folder and no other changes were apparent, one could assume that the file was uploaded. If a file with the same name appeared in a different folder, the best guess would be that the file was copied. In the on condition, participants could navigate to the folder's activity view and see the list of changes. All our participants chose to consult activity views for the task.

Evaluating Shadow Files and Traces

Our first goal was to test whether the use of shadow files with traces improves file traceability and lets the user understand a file's immediate history. If this was the case, enabling shadow files should reduce the time it takes to do the task, increase the number of correct answers the users give, and boost the users' confidence in their own correctness. The two test collaborators, Alice and Bob, each made the following changes: creating a folder, uploading a file, copying a file, moving a file, renaming a file and deleting a folder with two files in it. We developed a traceability questionnaire with 5 questions that concerned particular files and asked participants what happened to those files. The questions were about the two files that were renamed, two files that were moved, and the deleted folder. Participants had to select the operation performed on the file and write the name of the user who performed it. In the case of a move or a rename operation, they were also asked for the final name or location of the file. Participants



(a) **Mean task completion times** (in seconds)

(b) **Median accuracy** (percentage of correct answers)

(c) **Median confidence** (user's self-reported confidence, min=1, max=7)

Figure 7. Results of within-subjects experiments (N=16) with activity views and shadow files disabled/enabled (off/on). Error bars show standard error.

could get partial credit if parts of answers were correct. Activity views were disabled for the duration of the two tasks. In the off condition without traces, it was not possible to say conclusively who deleted a given file. Therefore, we accepted "it was deleted" without the username specified.

Results

We had 16 participants (5 female), aged between 23 and 31 (median age: 25.5), all with Dropbox experience, and 12 of them were also Google Drive users. First, Figure 7(a) shows mean completion times for both tasks per experiment. As can be seen, there is a larger difference in the average times for the tasks involving shadow files than the tasks evaluating activity views. Second, Figure 7(b) shows the median accuracy of the users. Using accuracy as a discrete measure, we refer to it as the number of points participants got for their answers. The maximum was 16 for the activity views tasks and 9 for the shadow files tasks. Finally, Figure 7(c) shows the median confidence of the users for all the tasks. In our experiments, confidence was a discrete value, from 1 (least confident) to 7 (most confident). In general, the differences between conditions look more pronounced for the shadow files than for the activity views tasks. In both experiments, the accuracy and confidence ratings were higher and showed much smaller variance in the on conditions.

For significance testing, we used built-in R functions to evaluate the results. The null hypothesis was that the measurements with the feature enabled and without it are not significantly different. The alternative hypothesis was that the measurements are significantly different. The independent variable was the presence of a particular feature. The dependent variables were the task completion time, the participant's accuracy, and the participant's own subjective estimate of their confidence. For ordinal quantities, such as accuracy and confidence, we ran the Friedman rank sum tests. If the p-value obtained in the test was less than 0.05 and therefore significant, we ran the Wilcoxon signed rank test to verify the significance of the result. Since the time it takes to do the task is a continuous measure, we ran the repeated measures ANOVA test as pairwise t-tests with the Holm correction. A p-value < 0.05 meant the result was significant.

For the activity view-related tasks, there was no significant effect on the time it took to do the tasks ($p > 0.22$). However, there were significant effects on confidence ($p < 0.003$) and accuracy ($p < 0.009$). Presence of shadow files and traces had a significant effect on the time ($p < 0.001$). In addition, the effects on confidence ($p < 0.007$) and accuracy ($p < 0.002$) were also significant. While not all results were significant, they are particularly encouraging since users showed more confidence in both tasks with MUBox's features on.

Based on the results, we can conclude that activity views appear to significantly increase people's accuracy and confidence. However, the use of activity views may not be significantly faster. On the other hand, shadow files with traces appear to increase participants' accuracy and confidence. Furthermore, the use of shadow files can be significantly faster.

Users generally agreed that the tasks were realistic (median scores of 6 on a 1-to-7 Likert scale for both activity views and shadow files tasks). This leads us to believe that the results of the study could also have an impact on real-life shared folder maintenance needs. We also asked the users for other subjective estimates in the post-task questionnaires. Here, the users tended to consider both activity views and shadow files effective features to help them do the tasks (median score of 6 for activity views and 6.5 for shadow files). These effectiveness estimates contrast with the ones in the off conditions (median of 4 for activity views and 2.5 for shadow files).

In post-study questionnaires, the users gave generally positive feedback. Specifically, they found that both activity views and shadow files were useful features (median scores of 7 for both). They also generally agreed that existing cloud storage providers should implement features similar to activity views and shadow files (median of 6.5 for both).

Limitations

We decided to do controlled experiments to demonstrate the potential of the proposed features. While users found the tasks to be realistic, there were two limitations.

First, the folder structures in the tasks were similar, but not identical. We especially strived to make them as similar as possible for the two tasks that pertained to the same feature. Each folder hierarchy had 14 files, but could have different numbers of folders (5 in the activity views and 6 in the shadow files tasks). The users were told to assume that filenames were unique within each task.

Second, opening a file that may have been moved or renamed to examine its contents can be one strategy to infer the operation performed on the file. When preparing the study, we

decided to make all the files empty to keep the experiment simple. Without the files' contents, however, participants had to rely on the uniqueness of filenames in the shared folder hierarchy, which was a given in the experiment, but is often not the case in real shared folders. Detailed evaluations of the proposed features including voting in a long-term deployment of MUBox in different collaborative settings are planned.

CONCLUSION

There are powerful document management systems (e.g. Alfresco, Huddle, SharePoint) designed to support enterprise document management workflows. MUBox aims to be a lightweight extension of existing cloud storage solutions such as Dropbox–i.e. no version control and no heavyweight configuration in terms of file ownership and access control lists. We considered that the main reason for the vast adoption of Dropbox and the like is that they are simple and easy to use. MUBox adds important features for collaboration without adding unnecessary complexity. It was specifically designed to explore minimally invasive features to support collaborative work on top of existing solutions. We conducted controlled experiments to investigate the effects of two of the features implemented in MUBox and how they aid workspace awareness and file traceability. Our results are encouraging and could have an impact in that developers might be more confident in adopting such features in the future.

First, our study shows that the less invasive features, activity-centric folder views and shadow files with traces, already significantly improve awareness and traceability. Second, an important aspect of MUBox is that it can be tailored to different usage scenarios. That is, more advanced features such as voting and notifications can be turned on/off depending on requirements. Third, our initial user feedback and internal use of MUBox show that change voting schemes are easily understandable and useful. In particular, they align very well with how people work with shared files, but without the need for external communication such as email.

We think of MUBox as a platform for further research on extending personal cloud storage services to collaborative work rather than the final solution. We also note that it is not our intention to replace existing personal cloud storage services. Rather, the idea was to show how a system like MUBox could be designed to implement the proposed features for collaborative use on top of existing services. Therefore, the proposed cloud abstraction layer is an important contribution of this research. It is important in two different ways. First, the abstraction makes sure that MUBox is a vendor-independent solution so that collaborators can benefit from the same workspace awareness, file traceability and voting mechanisms across different providers—for now, this includes Dropbox and Google Drive. Second, the cloud abstraction is also meant to provide guidance to future developers of personal cloud storage services who may want to implement the proposed features.

REFERENCES

1. Bergman, O., Tucker, S., Beyth-Marom, R., Cutrell, E., and Whittaker, S. It's not that important: demoting personal information of low subjective importance using GrayArea. In *Proc. CHI* (2009).
2. Biehl, J., Czerwinski, M., Smith, G., and Robertson, G. FASTDash: A visual dashboard for fostering awareness in software teams. In *Proc. CHI* (2007).
3. Calefato, F., Gendarmi, D., and Lanubile, F. Adding social awareness to Jazz for reducing socio-cultural distance between distributed teams, 2009. Eclipse-IT.
4. DeLine, R., Czerwinski, M., and Robertson, G. Easing Program Comprehension by Sharing Navigation Data. In *Proc. VL/HCC* (2005).
5. Dourish, P., and Bellotti, V. Awareness and coordination in shared workspaces. In *Proc. CSCW* (1992).
6. Foster, I., Vöckler, J., Wilde, M., and Zhao, Y. Chimera: AVirtual Data System for Representing, Querying, and Automating Data Derivation. In *Proc. SSDBM* (2002).
7. Froehlich, J., and Dourish, P. Unifying artefacts and activities in a visual tool for distributed software engineering teams. In *Proc. ICSE* (2004).
8. Jensen, C., Lonsdale, H., Wynn, E., Cao, J., Slater, M., and Dietterich, T. G. The life and times of files and information: a study of desktop provenance. In *Proc. CHI* (2010).
9. Kantor, M., and Redmiles, D. Creating infrastructure for ubiquitous awareness. In *Proc. INTERACT* (2001).
10. Karlson, A. K., Smith, G., and Lee, B. Which version is this?: improving the desktop experience within a copy-aware computing ecosystem. In *Proc. CHI* (2011).
11. Kersten, M., and Murphy, G. C. Using task context to improve programmer productivity. In *Proc. SIGSOFT* (2006).
12. Marshall, C., and Tang, J. That syncing feeling: early user experiences with the cloud. In *Proc. DIS* (2012).
13. Omoronyia, I., Ferguson, J., Roper, M., and Wood, M. A review of awareness in distributed collaborative software engineering. *SPE* (2010).
14. Rader, E. Yours, mine and (not) ours: social influences on group information repositories. In *Proc. CHI* (2009).
15. Sultana, S., and Bertino, E. A file provenance system. In *Proc. CODASPY* (2013).
16. Voida, A., Olson, J. S., and Olson, G. M. Turbulence in the clouds: challenges of cloud-based information work. In *Proc. CHI* (2013).
17. Voida, S., Edwards, W. K., Newman, M. W., Griner, R. E., and Ducheneaut, N. Share and Share Alike: Exploring the User Interface Affordances of File Sharing. In *Proc. CHI* (2006).
18. Xu, J., Zhang, J., Harvey, T., and Young, J. A Survey of Asynchronous Collaboration Tools. *Information Technology Journal* (2008).