Int J Digit Libr (2016) 17:175–187
DOI 10.1007/s00799-016-0181-7

Using a file history graph to keep track of personal resources across devices and services

Matthias Geel[1] · **Moira C. Norrie**[1]

Received: 4 January 2016 / Revised: 5 June 2016 / Accepted: 17 June 2016 / Published online: 7 July 2016

Abstract Personal digital resources now tend to be stored, managed and shared using a variety of devices and online services. As a result, different versions of resources are often stored in different places, and it has become increasingly difficult for users to keep track of them. We introduce the concept of a file history graph that can be used to provide users with a global view of resource provenance and enable them to track specific versions across devices and services. We describe how this has been used to realise a version-aware environment, called Memsy, and report on a lab study used to evaluate the proposed workflow. We also describe how reconciliation services can be used to fill in missing links in the file history graph and present a detailed study for the case of images as a proof of concept.

Keywords File provenance · File reconciliation · Cross-device indexing · Resource management

1 Introduction

Users now commonly work with multiple computing devices in their daily lives, often using external storage devices or cloud services as a way of moving digital resources between devices and sharing them with others. At the same time, web-based media services and social networking sites are used not only to share resources such as images, but also to manage them and make them accessible from different locations. However, while it has become much easier to globally access and share resources, it has become more difficult to keep track of different versions stored in different places [21].

The versioning of resources may be deliberate or accidental. Deliberate versioning can occur because a user chooses to keep track of the history of a resource, or to create variants for different purposes. For example, they might choose to create different versions of an image to optimise the resolution for different viewing contexts. If one of these image versions is uploaded to a social networking site, it will be transformed and stored on the service, thereby creating yet another version of the resource.

Accidental versioning occurs due to our ways of working with multiple devices and services, where we often create copies and then modify them without synchronising the copies. We now expect to be able to work on the same document in different locations, often blurring the distinction between private and working environments [10]. Users can move the files that they are working on between devices in several ways including using a cloud storage service such as Dropbox, using an external storage device such as a USB flash drive or emailing an attachment to themselves. Even though working on a document stored in a cloud storage service ensures synchronisation, many users prefer to first create a local copy before editing. The result is that the user ends up with multiple versions in different locations and often struggles to remember later where a particular version is stored. The problem becomes worse if they have renamed files and moved them between folders.

Our goal is to provide users with a global view of their digital resources so that they can get answers to questions such as: "Where is the latest version of this document and on which device/service is it stored? Does the previous version still exist somewhere? Where is the original? How many copies exist and to which, if any, file hosting services have

✉ Moira C. Norrie
norrie@inf.ethz.ch

Matthias Geel
geel@inf.ethz.ch

[1] Department of Computer Science, ETH Zurich, Zurich, Switzerland

they been uploaded?". Importantly, we want to do this without requiring them to change their ways of working. The solution, therefore, has to be embeddable in standard desktop environments and work alongside existing applications and file management tools.

To achieve this, we propose the concept of a *file history graph* for managing metadata about a user's entire information space, possibly spanning several computing and storage devices as well as online file hosting services. If a user has access to a particular version of a file, the file history graph enables them to get an overview of the provenance of that file and track down a particular version such as the latest or the original. As proof of concept, we have developed Memsy, a version-aware environment that helps users keep track of resources based on the file history graph.

For legacy reasons or due to restricted access to external services, there may, however, be missing links in the file history graph. For example, an image uploaded to Facebook not only gets transformed but also loses its original id, and this results in a gap in the file history graph unless the user explicitly creates a link. We have addressed this by integrating a reconciliation engine that uses file system properties and content-based fingerprints to find similar files and help users fill in the gaps. Given the popularity of uploading images to various media hosting services and social networking sites, and the difficulty of tracing these images back to the originals stored locally, we chose to investigate a reconciliation service for images.

This article extends a previous publication [15] by not only providing more details of the approach and the Memsy implementation, but also adding a section that describes how we adapted similarity-based image retrieval techniques to support reconciliation and reports on our evaluations.

An overview of related research and studies is provided in Sect. 2 before presenting our approach and the general architecture of the Memsy version-aware environment in Sect. 3. Details of the global resource catalogue, file history graph and the reconciliation engine are given in Sects. 4, 5 and 6, respectively. We then describe the user experience in terms of how they interact with the environment and access its services in Sect. 7, before discussing the technical challenges of implementing such a system in a highly distributed setting along with our Memsy solution in Sect. 8. We report on a lab study carried out to evaluate the Memsy workflow in Sect. 9 and present concluding remarks in Sect. 10.

2 Related work

The requirements for information work to be carried out successfully have changed as the number of personal computing devices has increased [10,28,30]. The study by Dearman et al. [10] revealed that participants found managing information across devices the worst aspect of using multiple devices, and they argue that it is important for system designers to consider a user's collection of devices and no longer assume a single personal computer.

As highlighted in interviews with 22 professionals from different industries carried out by Santosa and Wigdor [30], consumer-oriented cloud storage solutions have now become an integral part of many personal workflows. While these services allow resources to be accessed on multiple devices, none of their participants regarded the cloud as a complete data solution, and data fragmentation remains an important topic even in the context of the cloud. Participants explicitly stated that they had difficulties in remembering where in the cloud they put their data, and consequently, the authors recommended that methods need to be found for improving the awareness and visibility of what is happening in the cloud. In a recent project within our group, we specifically looked at ways of increasing awareness within teams when cloud services such as Dropbox are used to support collaborative work [27]. In contrast, here we address the issue of how to make users more aware of how their own personal data are distributed across a range of devices and services, including Dropbox, to help them keep track of and reconcile different versions.

Research into how human memory works has shown that users quickly forget important computing tasks [9]. Elsweiler et al. [13] point out similarities between everyday memory lapses and memory lapses in the context of personal information management and suggest that tools should be provided to help users recover from such lapses. Therefore, the ability to track and re-find information in a desktop environment is a highly desirable property for today's information management tools [18]. It has been shown that one effective way of providing memory cues is to use the idea of provenance information [4].

Previous research has explored two main strategies for helping users to keep track of their resources. The first category of tools aims to provide better search and organisational facilities based on metadata/time [12] or tagging [8]. The second category focuses on the associations between resources rather than the properties of the resources themselves and includes navigation by associations [6] as well as the exploitation of semantic relations between resources [20]. However, most of these solutions have been designed with the underlying assumption of a single desktop computing environment, and it is unclear how they could be adapted to multi-device environments. In fact, the authors of the *Stuff I've Seen* system [12] report from their user study: "The most requested new feature [in the user study] is unified access across multiple machines".

The fact that users have different work practices is reflected in how they set up their working environments and organise their files. For example, while some users store

every document created or downloaded in a well-defined folder hierarchy, others regard the desktop as a temporary workspace for all kinds of documents currently being worked on and recently downloaded, similar to how they might use their physical desks [32]. When working with multiple devices, it has been shown that cross-device work patterns can often be complex [30].

Since users often put a lot of effort into customising their working environment and file organisation, they tend to be reluctant to change their work habits. For example, Jaballah et al. [17] proposed a digital library system that enables documents to be browsed via various metadata axes, but a diary study showed that most users preferred the more limited folder view, and this was, in large part, attributed to the time required to become familiar with a new application interface. Other studies on personal information management have shown that users are actually very fond of their folder structures [19] as a means of not only categorising information but also decomposing problems and projects into smaller units. Despite their shortcomings, folder hierarchies, therefore, remain the basis for file organisation. However, using visualisations such as greying out older versions has been proposed to help users to find the latest versions of files within folder structures [2].

The apparent reluctance of users to move away from established ways of working led us to conclude that user behaviour changes very slowly, and new solutions for managing personal resources should try to build upon current work practices and the conceptual models underpinning them rather than revolutionising them. Also, we argue that it is crucial to embrace rather than oppress the diversity in strategies and methods people employ to organise information, and take these into account when designing new tools for personal resource management.

We now review previous approaches for capturing file provenance data without changing users' work practices. A general primer on provenance and different methods that have been used to capture provenance data is given by Carata et al. [5]. They make the case for tracking provenance information at different levels, including that of the file system.

Collecting provenance data at the system level promises to gather lineage data in an application-independent manner. The Provenance-Aware Storage Systems (PASS) [25] project and the file provenance system FiPS [31] are two examples of provenance systems that operate at the system level and collect provenance data automatically without application developers having to adopt application-specific provenance solutions. Whereas PASS focuses on command-line invocation of data transformations and derives provenance information from system calls as well as the current execution environment, FiPS proposes a stackable file system on top of an existing file system that intercepts any file system calls passed through the Virtual File System (VFS) layer. We note that PASS has been extended with support for cloud storage solutions [26], but they focused on commercial solutions such as Amazon Web Services (AWS) rather than consumer-oriented cloud storage services such as Dropbox, Google Drive and OneDrive and did not consider online media hosting services.

To capture provenance events at the application and content level, a number of technological solutions and tools have been developed. The TaskTracer system [11] hooks into the Microsoft Office suite by installing .NET COM addin objects that capture save-as operations. A different approach was taken by Karlsen et al. [21] where their copy-aware environment monitors copy-events by injecting a C++ hook library into a number of selected applications and the operating system itself. A custom Outlook integration allows email attachments to be tracked as well. Both systems intercept file manipulation operations in Windows such as copy, move and rename. A Windows clipboard hook was employed to capture copy/paste operations within and between applications. While application-level instrumentation enables a very fine-grained collection of provenance events, it is questionable how such an approach scales to a larger number of end-user applications. Even worse, when it comes to third-party file hosting services, it is virtually impossible to instrument those services with provenance capabilities. Using similarity metrics at the metadata and content level, our goal is to offer a best-effort approach to reconcile non-tracked files with existing provenance data.

A UI-less version of the TaskTracer tool was also used for a longitudinal study of knowledge workers at Intel Corporation, conducted by Jensen et al. [18]. This field study analysed several sources of provenance events and documented their type and frequency. In particular, their quantitative data suggest that provenance events occur frequently in typical computer use by knowledge workers and that those provenance events were, in fact, memorable to the participants of the field study.

Although the IRCUS system [29] was developed for an entirely different use case, namely that of helping users to identify related content when deleting sensitive data from files or projects, there are many similarities to our proposed approach. First of all, they had the same goal of not changing users' work practices and being able to integrate the service transparently without modifications to applications or the operating system. Second, they also observe file system events to derive relations between files as well as using content overlap and access patterns to determine whether files are related.

In summary, while previous research has demonstrated feasible solutions for automatically capturing provenance events at the file system level [25,31] as well as the application level [11,18,21], they do not propose solutions for identifying and tracking files across several collaborating

computing devices and services, even though these systems could support cross-device provenance by design. Further, they do not address the problem that there will always be gaps in the provenance information due to files that were created/modified outside the reach of the system or before the system was in use. We believe that a reconciliation service is an essential part of any system designed to keep track of personal resources. Information retrieval techniques, such as similarity search, near-duplicate detection and fingerprinting, are valuable tools to fill in these knowledge gaps when provenance information is not available.

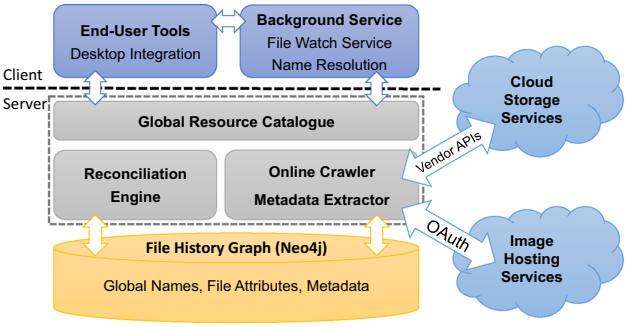

Fig. 1 Architecture of the Memsy version-aware environment

3 Version-aware environment

Our vision of a version-aware environment is similar to that of the copy-aware computing ecosystem proposed by Karlson et al. [21], but with a stronger focus on the highly distributed nature of current digital ecosystems. We try to embrace rather than avoid the fragmented nature of personal information spaces, abandoning the single desktop assumption and accepting that complete control over one's computing environment is often difficult to achieve. Our aim is, therefore, to help users to keep track of resources rather than trying to impose strict controls and synchronisation policies that would change their ways of working.

The approach that we propose is to build an infrastructure on top of the familiar desktop environment that provides users with a unified view of their personal information space across their multiple computing and storage services as well as online file hosting services. We use the term *personal information environment* to refer to the sum of all devices, services and applications that a user employs to manage and share their personal information. A distributed approach is used to collect provenance information across the personal information environment to build a global provenance index.

The architecture of the Memsy environment that we developed is shown in Fig. 1. While the concepts, principles and core components are general, we developed a full set of components for Windows-based systems to demonstrate how the approach could be integrated into an existing desktop system and to enable us to carry out user studies.

Since system access should be independent of any individual computing device, we opted for a classic client–server approach. The system comprises three main components—the *global resource catalogue*, the *file history graph* and the *reconciliation engine*—which together realise a "version-aware" computing environment for end users. Below, we outline the main features of these components and how they work together with the other system components, before going on to describe them in detail in the following sections.

Global resource catalogue This is the main service that orchestrates all components and communicates with various client services and tools. It provides a global view of a single user's personal information space by ensuring that each resource is not only uniquely identifiable but can also be located in an unambiguous way. A file that is part of the global resource catalogue is called a *managed file*. All other files, including those that are not part of a user's personal information space, are *unmanaged*. The global resource catalogue is responsible for collecting updates from client devices as well as cloud storage and image hosting services and applying them to the file history graph. Online services are crawled at regular intervals or on demand.

File history graph The file history graph is the data structure that stores the metadata required to identify and locate files across devices and services. Essentially, it is an index that provides a lightweight, implicit versioning mechanism for files.

Reconciliation engine This component reconciles *unmanaged* files, which have not been tracked previously or cannot be tracked reliably (for example, files on a cloud storage service beyond our control), with existing file histories. In both cases, the goal is to integrate those files into the global resource catalogue and create any missing links in the file history graph. Reconciliation is either done automatically based on file properties or user-driven. In the latter case, we use content-based similarity metrics to aid users in their decisions.

4 Global resource catalogue

To build a global catalogue of a user's personal information space, each file needs to have a unique and non-ambiguous mapping between a global namespace and its actual location. Figure 2 shows the structure of the global address scheme and an example mapping. Our scheme follows the recommendations for Uniform Resource Identifiers.[1]

A vital part of that scheme is the numerical identifier of the storage device on which the file is stored

[1] http://tools.ietf.org/html/rfc3986.

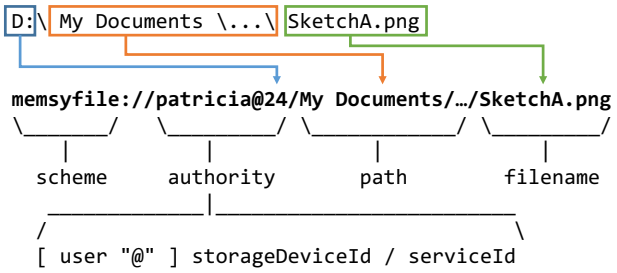

Fig. 2 Global namespace scheme. The example given uses a `storageDeviceId`

(`storageDeviceId`). In our environment, everything that has a mount point and is writeable by the user can be assigned a globally unique identifier that is then stored in the root directory of that particular storage device. This allows us to reliably recognise external storage devices such as USB flash drives or external hard disks, even when they are moved between computers.

To differentiate file hosting services, we use pre-defined names, e.g. *dropbox* or *googledrive*, as identifiers (`serviceId`). The path component corresponds to the original path. Because number-based storage device identifiers are not self-descriptive, they can also be mapped to more user-friendly names. By default, we map them to labels provided by the operating system, but users can supply more meaningful labels, e.g. "Red Kingston USB Stick". Since these mappings are stored in the global resource catalogue and not on the devices themselves, users are able to restore device identifiers if they are deleted by accident. A full file scan might be necessary to provide an up-to-date file history graph. Similarly, devices and all their associated location nodes can be purged from the file history graph should the device be lost.

In this address translation scheme, we assume that an absolute file path, including the file name, cannot be occupied by more than one file at the same time. This is true for all mainstream file systems (NTFS, ext3, HFS+), although they may internally use unique file identifiers instead of paths. Also, many user applications prohibit filenames that only differ in case, for example, Windows Explorer, and the same is true for many file hosting services.

On each computing device belonging to the user's personal information environment, a background service is deployed that monitors connected storage devices and tracks the user's file activities. These computing devices also synchronise a list of connected storage devices upon startup and whenever the list changes. This information is used later to tell users about the last known location of "movable" storage devices such as USB flash drives. Global name resolution at runtime is straightforward, because the local service knows the identifiers and mount points of all locally connected storage devices. This is especially useful on the Windows platform, where the mount points, i.e. drive letters, of external storage devices usually depend on the order in which they were connected.

5 File history graph

The file history graph is a data structure that serves two purposes: It records the metadata of each new version of a file and also stores the last known location(s) for each file version. From a data model point of view, each file history has a root node that denotes a newly created file and a tree of successor version nodes. Each version node may point to one or more location nodes. The resulting structure is, therefore, a forest of directed trees, with each tree representing a separate version history.

Similar to other provenance systems [11,18,21], a single file history is a directed acyclic graph. However, in contrast to these systems, we neither instrument any applications nor are we interested in content-based provenance events. Instead, our aim is to collect the cross-device and cross-service provenance chain of files as a whole, with a focus on providing information about their last known locations. For example, if a user moves a file from location A to location B and, then, sometime later moves it from location B to location C, we do not need to store the history of where it has been located, but rather only the last known location C. Therefore, when a file is moved, the old location is deleted in the file history graph and the new location added. Similarly when a file is renamed, we do not need to keep a history of the names, and, so, simply update the name of the corresponding entry in the file history graph. The only time when a new node is added to the file history graph is when a new version of a file is created either explicitly or by creating a copy on another device or service. Consequently, since we do not keep records of the full provenance chain with regards to renames and moves, we have a much more compact file history graph.

Figure 3 illustrates how the file history graph structure captures file operations. A *create* event triggers the creation of a new file history with root V_1 and location L_1. If V_1 already exists, it represents a *copy* operation, and we simply add a new location node to the existing node. When a *modify* event occurs, we first identify the (old) file version node that corresponds to the location of the modified file. If that version node is a leaf node, we derive a new version node and reattach the location node to it. If it is not a leaf, a newer version must already exist somewhere else. In that case, we perform a fork. All other file operations (*rename*, *move* and *delete*) only affect the last known locations of file versions and update them accordingly. As a consequence of the delete operation, file version nodes may end up with zero location nodes attached to them, but they are kept for future reference.

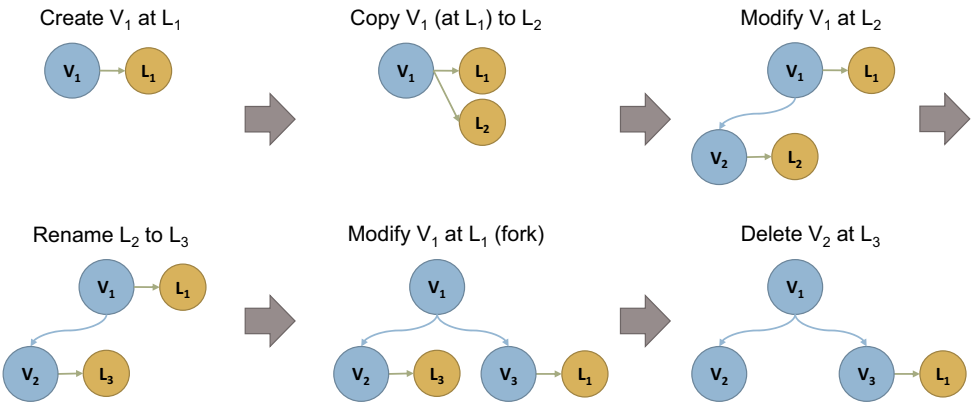

Fig. 3 Evolution of a single file history

Each file version node is uniquely identified by the (MD5) hash of the file content. A database index over these hashes provides very fast retrieval of entry points into the file history graph.

It is an important characteristic of our proposed index structure that we retain metadata about previous file versions, even if they no longer exist in the personal information space. With the information in these file histories, the system is able to provide answers to many user questions about the location of copies, latest versions and file origins across devices.

File events are sent immediately to the server if the device is online; otherwise, they are cached locally and stored on the storage device where the operations occurred. Thus, the cache travels with the storage device, and if, for example, a USB drive is shared between computers that are both offline, the order of executed operations on that USB flash drive is always guaranteed, without the need for complicated time-synchronisation. We have implemented the cache as an event-queue backed by a transactional embedded database (HSQLDB). That queue is consumed by a background thread, and entries are only removed if the event was successfully submitted to the server; otherwise, processing is halted until Internet connectivity is restored. Since all events (update, move, rename, delete) are idempotent if executed in order, we not only achieve at-least-once semantics, but can also guarantee that all locally observed transactions are eventually reflected correctly in the global file history graph.

Finally, we note that while our current file history graph has the advantage of compactness since it does not record the full provenance chain including moves and renaming, it could be expanded to record this information. This would allow additional functionality, such as the ability to reproduce changes in a folder over time. This feature could be useful if a user remembers where a file was located previously and now wants to discover its current location. In a related project, MUBox [27], we explored this idea in a multi-user context, introducing the concept of shadow files to represent files that had been moved or renamed by other users. It would be interesting to investigate the usefulness of this concept in a single user context.

6 Reconciliation engine

Regardless of how well a system is able to track files within a controlled environment, there are bound to be resources encountered that either have not been tracked previously or originate from a system that cannot be monitored directly. We call the process of turning such unmanaged resources into managed ones *reconciliation*.

There are three main cases where file reconciliation is required. The first deals with legacy issues arising from the fact that users will already have files scattered across different devices and services when they start using a system such as Memsy, and the different versions of these files may have to be reconciled at some point. The second concerns the use of file hosting services where it is often difficult to reliably track resources from the outside. Even worse, some image hosting services resize and recompress uploaded images, thus changing the actual file content quite significantly. Third, there will always be cases where a file enters a personal environment from *the outside*, for example, as an e-mail attachment, downloaded file or copy from somebody else's USB stick, and may need to be reconciled with existing versions within the user's environment.

To align such files with our global resource catalogue and the file history graph, we primarily rely on hashing and file path matching. To recognise copies, we need to be able to tell whether two files have the same content. Popular file hashing functions, such as MD5 or SHA1, have proven to be very effective [3], and we, therefore, use them to realise this functionality. We note that although there have been some reservations about using cryptographic hashes to implement compare-by-hash [16], for the purpose of re-establishing provenance relationships, we are of the opinion that a fast best-effort approach is more valuable than perfect correctness. Since we store these hashes for previously encountered

versions as well as the current ones, we can reliably reintegrate files that correspond to older versions. For example, a user might have received a report from a co-worker by e-mail, saved it in a local, observed location and modified its content. Several weeks later, they might want to retrieve the revised version but cannot remember where they stored it. Luckily, they remember the e-mail that contained the original file. With the attachment from that e-mail serving as an entry point into the file history graph, they can learn about the location of the revised version. Having such provenance information allows users to discover newer versions of documents based on the copy of an older one.

Since one of the main use cases for reconciliation concerns tracking versions of images, we decided to investigate this case in detail. A scenario might be that a user uploads an image onto a social networking website such as Facebook and, when a friend requests a copy of the image, wants to trace the original to send their friend one with higher resolution. If a provenance relationship between the original image stored on a local hard disk and the uploaded image could be established, then the user could navigate directly to the original rather than having to search manually, which can be particularly tiresome if they have many similar images.

A lot of research has addressed the problem of finding similar images using techniques such as interest point extraction [24], local image descriptors such as PCA-SIFT [22] and min-Hash based similarity metrics [7,23]. Although some of these techniques can be used effectively to detect what we consider as duplicates, most of them are concerned with answering queries to find similar images in the sense of content-based image retrieval. Since we are only interested in finding different versions of the same image and not in finding similar, but distinct, images, we decided instead to focus on a specific class of computer vision algorithms that calculate a fixed-size image signature which can be considered as a "perceptual hash".

The main idea of such an image signature algorithm is to map an image to a *sparse image representation* which is significantly reduced in size but retains important perceptual information. An advantage of such low-dimension representations is that they can be compared efficiently. An appropriate distance metric is then used to compare these sparse image representations to find near-duplicates.

We investigated three candidate algorithms. (1) A naive grey scale (GS) algorithm that computes very small thumbnails and compares them on a per-pixel basis. This is basically an aggressive JPEG compression scheme applied to the luminosity channel. The size of the thumbnail is set as a parameter, but, by default, is as small as 5×5 pixels. (2) An approach based on discrete cosine transform (DCT) [1] which transforms the image space to frequency space and only retains the frequencies representing the coarse-grained structure of the image. Roughly speaking, DCT transforms a vector into a set of basis vectors, each of which describes one particular cosine wave in the spatial dimension. (3) discrete wavelet transform (DWT) based on Haar wavelet, which inspects the image at different scales and, for each of them, extracts a high-frequency signal for both dimensions individually as well as for the combined dimension.

When a user initiates a reconciliation request, a ranked list of matching images is returned in increasing order of distance from the input image. To evaluate the different algorithms, we carried out an experiment using a set of 10,495 images randomly selected from Flickr as the user's local image collection. From this collection of images, we randomly selected 1000 and modified them programmatically to simulate the set of images uploaded to an image hosting service. We then ran the reconciliation algorithms for each image of the collection of modified images against the full collection of images downloaded from Flickr.

All experiments were run on an Intel Core i7-2700 machine with four physical cores @ 3.50GHz and 24GB of RAM. The images were stored on an Intel SSD 520. All the calculations were done in-memory.

Table 1 shows the results of the experiment for the three different algorithms and five different forms of image modifications. Performance is compared in terms of two quality metrics: MAP@5 is the mean average precision at position 5, while AFP is the average false positives. For a single query, the false positive count measures how many negative results users have to skip until they have found the first true match. Within our test suite, this metric is equivalent to the inverse of the average precision—1, if the query yields exactly one relevant image. We argue that this metric is a better estimate

Table 1 Performance in user-driven mode

Modification	Algo.	MAP@5	AFP
Resizing	GS	0.999	0.005
	DCT	**1.000**	**0.000**
	DWT	0.999	0.002
Compression	GS	0.997	0.015
	DCT	**1.000**	**0.000**
	DWT	0.999	4.386
Oil filter	GS	0.992	0.115
	DCT	**1.000**	**0.000**
	DWT	**1.000**	**0.000**
Gaussian noise	GS	0.980	0.425
	DCT	**1.000**	**0.000**
	DWT	0.999	0.002
High contrast	GS	0.739	63.043
	DCT	**0.999**	**0.003**
	DWT	0.997	4.127

Bold values indicate the best results per modification

for the expected workload of a user determined to find all provenance relationships of an entire set of images.

As can be seen, DCT and DWT performed very well, with DCT achieving almost perfect results. Even GS achieved very high precision in all scenarios except the High Contrast. Our results indicate that these algorithms are robust with respect to minor image manipulations such as resizing and small changes in noise and contrast.

We accept that it is much more challenging to reconcile an image with its original if major image post-processing has been carried out, for example, significant cropping as well as major modifications in colours and tone. However, many users often keep a local copy of the processed version as well as uploading it which could provide the necessary entry point in the file history graph to either track down the original or a high-resolution version of the one uploaded to a social networking site.

It is beyond the scope of this paper to present our experiments on image reconciliation in detail. Our aim was to provide sufficient information to give some idea of the methods used and to convince the reader that it can offer valuable support to the user faced with the task of filling missing gaps in the file history. Further details of the experiments and results can be found in [14] where we also describe automatic reconciliation processes and other experiments using string similarity measures over filenames as a basis for reconciliation.

7 User experience

To use the system, a Memsy client has, first, to be installed on each of the user's computing devices. Through these clients, the user can select local folders to be added to the observed environment. This means that the files within these folders will then become managed in the sense that they will be included in the global resource catalogue. This allows very fine-grained control over which parts of the file system are monitored.

Users can also connect and add external storage devices such as personal USB flash drives. As described previously, such storage devices can be labelled to make it easier to recognise them when they show up in search results. Note that each storage device, even external ones, only have to be configured once as their settings are stored directly on the devices.

After setting up the environment, users can continue to work as usual, freely creating, moving, renaming, copying and deleting files as well as creating new folder structures within the monitored folders. Behind the scenes, each file operation is propagated to the global resource catalogue, and the file history graph will be updated, so the user can later track files and get information about their locations.

If a user wants to find the latest version of a file, they follow a simple procedure shown in Fig. 4. (1) Small overlay icons in the file explorer view offer a quick glance of the current file status. From a user experience point of view, this works similarly to overlay icons in desktop integrations of popular cloud storage services such as Dropbox or version control systems such as TortoiseSVN. A tick symbol indicates that the current file is up-to-date, while an exclamation mark warns the user that a more recent version is stored somewhere else. Such icon overlays give a simple and unobtrusive way of providing status information to users. They integrate well with the desktop experience, work in all standard file dialogues and are well suited to handling directories with a large number of files.

When the user right-clicks on a file, a few Memsy-specific commands appear in the contextual menu. From there, the user can launch the Memsy Companion application, which is a small desktop tool that displays information retrieved from the global resource catalogue for that particular file (2). In the example shown in the top right of Fig. 4, the user sees that there are two newer versions located on other devices and services. Since the Memsy Companion application runs locally and can communicate with the local background service to retrieve the list of currently attached storage devices, it can show for each version whether that particular file can be accessed directly. Hovering over entries that are currently not available provides information about its last known location, in the case of this example, a USB flash drive.

A different context-menu entry invokes a search for similar files based on the currently selected file. The similarity search works as explained in Sect. 6 where we considered the example of tracing the original of an image uploaded to a social networking site. We now consider an alternative scenario where the users come across a document on their desktop which they use as a temporary work space and recall that this version was derived from a template. They are unable to remember where this template is stored, so they invoke a similarity search. The results are shown in Fig. 5.

From the results, the user learns from the first entry in the list that there is an exact copy on an external storage device, and from the second and third entries that there are similar, but not identical, files with the same name. One of these is stored locally and can be inspected directly. The other is located on another device that cannot be accessed directly. However, by clicking on the *Explain* button, they can view some more statistical information about the similarity that could help them judge whether or not it is worth accessing the other device to investigate the files stored there.

8 Implementation

Two different strategies are used to get the information needed about resources in various locations. *Crawlers* implement a polling-based model of observation that can either be

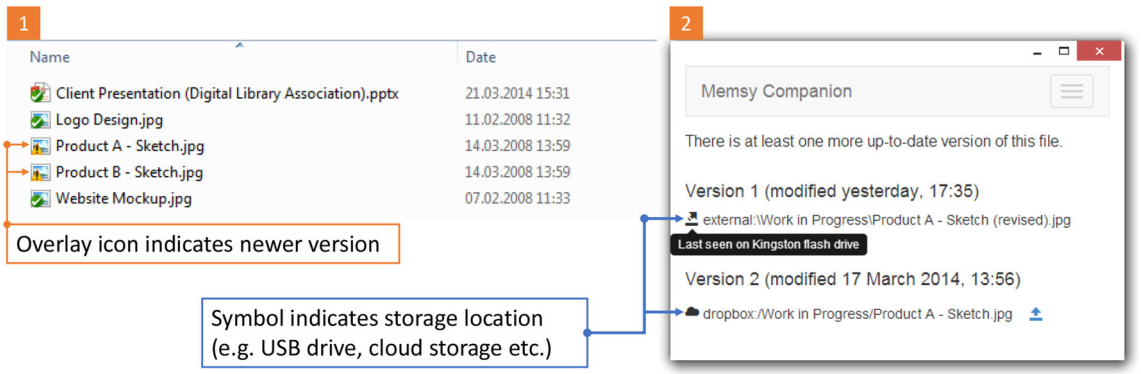

Fig. 4 A version-aware environment: (1) Overlay icons inform user about file status (cropped screenshot from Windows file explorer). Context menu entry invokes custom application (2) with further information about related versions

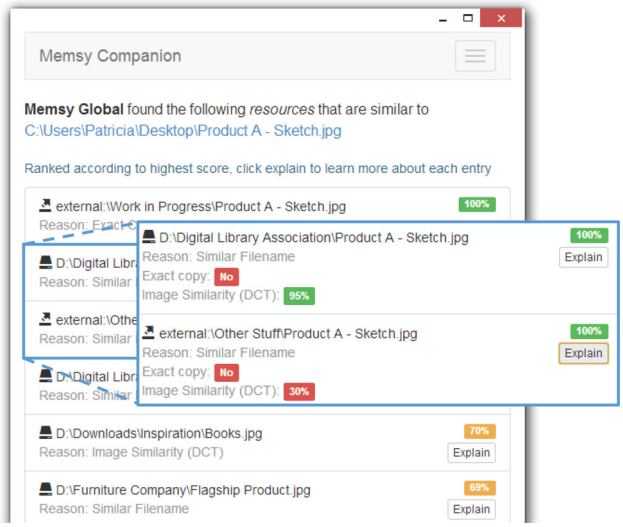

Fig. 5 Results of a similarity search

triggered automatically at fixed intervals or on demand by a user. *Watchers* implement an event-based observation model which means that they are notified by the operating system or online service when something changes.

File system watchers are supported in many operating systems, including Windows, and so, watchers are attached to the folders that the users add to the Memsy environment. A crawler is only necessary to build the initial index of files or resync a folder in the event that the watcher missed some events which can happen if the user kills the background process either deliberately or by accident.

Since few online services support push notifications, we mainly rely on crawlers to detect changes in the resources stored on these services. We have implemented crawlers for Dropbox, Google Drive, Facebook and Flickr. The former two cloud storage crawlers process all supported files, whereas the latter two only retrieve and process images. To avoid polling the entire data store which could be very expensive, many cloud storage APIs offer a delta function, which returns not only a list of events but also a token that can be sent with the next request so that only events that occurred since the token was generated will be sent.

Both crawlers and watchers forward their observed provenance events to the global resource catalogue which applies the necessary changes to the file history graph. Note that, in contrast to revision control systems such as Subversion and Git, users are not required to do any explicit check-outs or commits. Instead, whenever the background watchers deployed on the user's various computing devices detect changes in any of the observed folders, the file history graph gets updated automatically. Even though this approach is quite lightweight, it enables users to locate copies, latest versions and originals of resources.

To demonstrate and evaluate our approach, we developed a reference implementation of the Memsy environment in Scala, a JVM-based, multi-paradigm programming language that offers functional and object-oriented constructs. The global resource catalogue called *Memsy Global* runs on a dedicated server. It is built with a micro web framework called Scalatra.[2] As a backend for the file history graph, we use Neo4j,[3] a graph database implemented in Java, which is a perfect fit for the tree structure of our file history graph.

On the client side, we deploy a background service called *Memsy Local*, which is responsible for monitoring a user's file activities. It communicates with the server using its REST-style API over HTTP, with JSON as the data exchange format. To capture file system events, we make use of the Java 7 File API (NIO.2). When a new storage device is added to the environment, a new unique identifier is requested from the server and stored in a property file in the device's root folder. This enables external storage devices such as USB flash drives or portable hard disks to be reliably recognised when moved between computers. Unfortunately, Java does

[2] http://www.scalatra.org/.

[3] http://www.neo4j.org/.

not natively support device notifications for USB drives, but we work around this by periodically polling the list of mounted file systems for changes.

The user interface has been implemented for the Windows platform using a number of high- and low-level technologies. Both icon overlays and the context menu have been realised as Windows shell extensions. To indicate a file's status, an icon overlay handler was created for the Windows file explorer that queries the local Memsy client, which in turn may have to ask Memsy Global for the current status. Because icon overlays are precious resources in Windows, since only up to 16 handlers can be registered, our integration builds upon TortoiseOverlays, the icon handler of many popular desktop integrations of version control systems, such as Subversion, Git and Mercurial. This makes the icons easily recognisable by users of any of these solutions.

Our dedicated end-user tool, the Memsy Companion application is a Chrome app. These apps resemble native desktop applications but run on top of Google Chrome and can be built using web technologies. The tool communicates with the local background service and the global resource catalogue to help the users to keep track of their resources. We have chosen this web technology-centric approach, because it allows us to seamlessly mix various technologies from high-level web interfaces down to low-level shell extensions. The consistent use of the HTTP protocol also facilitates similar integrations for other operating systems.

9 Evaluation

To evaluate our approach, we conducted a small user study with two primary goals. First, we wanted to gather valuable insight into work practices and strategies for keeping track of resources across different devices. Second, we wanted to find out whether the tool support would be sufficient when a user cannot remember what happened to a particular document.

The main study design challenges were: *How can we reliably create a condition that reflects the situation where a user has actually forgotten the location of the latest version? How can we eliminate non-controllable factors such as an individual's particular cognitive abilities?* For these reasons, we decided to perform a controlled lab experiment instead of an external field study.

9.1 Study setup

We set up three workstations in a room to represent different work places. Each workstation had a labelled USB stick. All six devices were configured to belong to the same user, and all tools were pre-installed on the workstations. We prepared a set of 20 files for our hypothetical Memsy user: five presentations (PowerPoint), five reports (Word) and ten images (JPEG). These files had been copied to a managed folder on each workstation.

9.1.1 Participants

Our participants were recruited from an interdisciplinary European research project in which our group was a partner. This allowed us to recruit not only computer science researchers (three professors, three research assistants) but also people from media education/pedagogy (4), plus a designer and an architect. In total, we had 12 participants (50 % female, 50 % male) from industry and academic institutions. Although we had a rather small group of participants, they were from five different organisations with quite different work environments, and this matched our belief that a diverse set of study subjects would yield more interesting results than a large but homogeneous subject pool.

9.1.2 Procedure and tasks

We divided our 12 participants into groups of 3 persons. All participants of a group were invited at the same time, and each participant was randomly assigned to a designated work station. The study started with a pre-task questionnaire that collected background information about a participant's experiences with file managers and their current techniques for version management of documents and copying files between devices.

The main study was a sequence of information management tasks, where each participant performed exactly the same steps but worked on a different subset of the files. In the first part of the study, participants were asked to perform three common file management and document editing tasks: update and rename a presentation, copy an image to a newly created subfolder and modify it, copy a report to the USB drive and modify it. While we asked participants to follow our instructions closely, we did not tell them how to accomplish the tasks. It was up to the individual user to decide whether they worked with keyboard shortcuts, drag and drop or context menus. If they made a mistake (e.g. copied the wrong file), we asked them to revert the change however they saw fit. At that stage, the background service monitored the file management operations behind the scenes and propagated the changes to the global resource catalogue.

In the second part, participants were asked to gather information for an upcoming presentation which included files possibly modified by the other participants (one presentation, one report, two images) and spread across the other two workstations and USB sticks. To complete the task, they had to indicate if there was a newer version of any of those files, and, if there was, where the newer versions were stored (location, drive and path). At any time, they could invoke the

Memsy companion app, shown in Fig. 4, for further information.

9.1.3 Results

In total, each participant had to perform six individual file operations (modify, rename, move) in the first part of the study and four file enquiry tasks in the second part. To complete the enquiry tasks, they had to indicate if there was a newer version, and if there was, where it was stored in terms of the location, drive and path. Since the enquiry tasks depended on the execution of the correct file management operations by another participant in the first part of the study, mistakes made in the first part by one participant propagated to the second part for another participant. This was the case for two tasks, resulting in a total task completion rate of 95.83 %.

In our pre-task questionnaire, we asked participants about their current work practices for managing versions and copying files between devices. In both questions, multiple answers were possible. By far, the most popular method for managing multiple versions of office documents was a personal file naming scheme (ten participants). Although less popular, folder structures were still used for this purpose by half of the participants. Although revision control systems are widely used to manage source code, our results indicate that they are considerably less popular for managing versions of documents (4), let alone to share files between devices (2). None of our participants used a dedicated document management system such as SharePoint.

Interestingly, even though all 12 participants answered that they used cloud storage services to move files between devices, only 2 used the implicit versioning provided by some cloud providers. This might be due to the fact that popular services, such as Dropbox and Google Drive, by default, only store previous versions for up to 30 days. We conclude that cloud storage services have become a ubiquitous tool for transferring files between machines, but other means such as USB sticks (11 participants) and sending e-mail to yourself (11) still have their uses. These results affirm some of the assumptions underlying our initial use case and further inform our proposed approach.

All participants were able to use the Memsy Companion app to answer most of the questions from the second part correctly (overall completion rate was 46 out of 48 tasks). Figure 6 shows the results from the post-task questionnaire. For each of the statements, we asked participants to rate their level of agreement on a 7-point Likert scale, where 1 is *strongly disagree* and 7 is *strongly agree*.

The Memsy companion app as well as the entire experience offered by the Memsy environment received fairly positive ratings. Most participants felt that it was well suited to the task at hand (median 6), that it was easy and intuitive to complete the task (median 6), and that they were efficient in doing so (median 6). Participants also agreed with our underlying assumption that existing tools [offered by the operating system] provide poor support for the task (median 6). However, while many participants could relate to the task (median 5), most users (mode 4) were undecided about whether such a tool would actually be useful for them. These general concerns were also reflected in some of the feedback we received, for example P3 noted: "The general issue is what do I actually do when I realise that I don't have the most recent version in front of me".

We conclude that, while our prototype is successful in answering the kind of questions posed in the introduction of this paper, users may require more information to help them decide what to do when confronted with multiple versions on different devices. As P9 pointed out, "[The tool should provide] some overview of what kind of change has been done" and P10 agreed by saying "[The tool should] provide the differences of versions (visual comparison of versions)".

With regard to desktop integration, reactions from users were quite positive (median 6), but some expressed a preference for an even deeper integration with the regular desktop computing environment. To that end, we plan to experiment with other forms of shell extensions and consider providing more detailed information as part of the regular file explorer interface. However, further studies are required to steer development in the right direction.

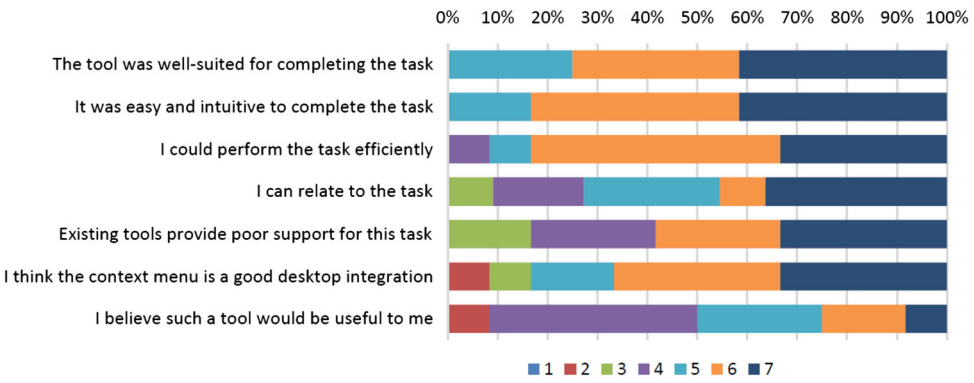

Fig. 6 Results from post-task questionnaire

10 Conclusion

The Memsy environment was designed to tackle the challenges of keeping track of personal resources across multiple devices and online services. Our aim was to provide a solution that could integrate with existing work practices and applications rather than replace them, and our Memsy prototype has demonstrated how this can be achieved. However, our initial studies have shown that providing more information about the nature of changes made to files, together with suggestions for how to resolve what are seen as problematic situations, would be desirable, and this is something to explore in future research.

We believe that the idea of a consolidated, global resource catalogue could benefit a number of existing PIM tools. We, therefore, see our work as only part of a foundation for new infrastructures and tools required to manage personal information in future digital ecosystems. For example, many file systems and services which use tagging store the tags in such a way that they are effectively lost when a file is moved to another device or service. We would propose that such file metadata be incorporated into the file history graph so that it is available globally and is independent of device and service. This idea could be extended to likes or comments associated with images posted on social networking or media hosting sites. By associating the data with the file history rather than a particular version, it would effectively be propagated to other versions. Another advantage of this approach is that it provides a means for attaching custom attributes to files stored in external repositories, such as cloud storage services, even though these services do not provide native support for custom attributes. This is something that we have investigated by building an information management layer on top of Memsy to offer a distributed personal resource management system that reflects features of many modern PIM tools, such as a collection model for managing resources and various forms of flagging, tagging, rating and labelling along with faceted search over these attributes. Details of this work can be found in [14].

So far, we have only considered single-user environments. However, in any realistic setting, personal files are often shared with other users who may simply create a copy or edit them. In fact, nowadays, cloud storage services are commonly used as the basis for collaboration. One interesting direction for future work would be, therefore, to investigate the implications of multi-user environments where personal resources could be shared with other Memsy users. In the MUBox project [27], we developed a multi-user cloud storage solution which showed how provenance information could be used in multi-user environments to improve awareness as well as supporting forward and backward traces. Our experiences in MUBox could usefully inform the design of an extended Memsy system capable of supporting multiple users with personal information spaces that may overlap. We note that this would require some fundamental changes to the Memsy model, including the integration of an ownership concept into the resource model and new methods for generating and handling global unique identifiers so that they would be unique across all personal information spaces.

References

1. Ahmed, N., Natarajan, T., Rao, K.: Discrete cosine transform. IEEE Trans. Comput. **C-23**(1), 90–93 (1974)
2. Bergman, O., Elyada Oded andn Dvir, N., Vaitzman, Y., Ben Ami, A.: Spotting the Latest version of a file with Old'nGray. Interact. Comput. **27**(6), 630–639 (2015)
3. Black, J.: Compare-by-hash: a reasoned analysis. In: Proc. USENIX Annual Technical Conference (ATC'06), pp. 85–90 (2006)
4. Blanc-Brude, T., Scapin, D.L.: What Do People Recall About Their Documents?: Implications for Desktop Search Tools. In: Proc. 12th Intl. Conf. on Intelligent User Interfaces (IUI'07), pp. 102–111 (2007)
5. Carata, L., Akoush, S., Balakrishnan, N., Bytheway, T., Sohan, R., Seltzer, M., Hopper, A.: A primer on provenance. Commun. ACM **57**(5), 52–60 (2014)
6. Chau, D.H., Myers, B., Faulring, A.: What to Do When Search Fails: Finding Information by Association. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'08), pp. 999–1008 (2008)
7. Chum, O., Philbin, J., Zisserman, A.: Near duplicate image detection: min-hash and tf-idf Weighting. In: Proc. 19th British Machine Vision Conf. (BMVC 2008), pp. 812–815 (2008)
8. Cutrell, E., Robbins, D., Dumais, S., Sarin, R.: Fast, Flexible Filtering with Phlat. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'06), pp. 261–270 (2006)
9. Czerwinski, M., Horvitz, E.: An Investigation of Memory for Daily Computing Events. In: Proc. 16th British HCI Group Annual Conference (HCI 2002), pp. 230–245 (2002)
10. Dearman, D., Pierce, J.S.: It's on My Other Computer!: Computing with Multiple Devices. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'08), pp. 767–776 (2008)
11. Dragunov, A.N., Dietterich, T.G., Johnsrude, K., McLaughlin, M., Li, L., Herlocker, J.L.: TaskTracer: A Desktop Environment to support Multi-Tasking Knowledge Workers. In: Proc. 10th Intl. Conf. on Intelligent User Interfaces (IUI'05), pp. 75–82 (2005)
12. Dumais, S., Cutrell, E., Cadiz, J., Jancke, G., Sarin, R., Robbins, D.C.: Stuff I've Seen: A System for Personal Information Retrieval and Re-Use. In: Proc. 26th Annual Intl. ACM SIGIR Conf. on Research and Development in Informaion Retrieval (SIGIR'03), pp. 72–79 (2003)
13. Elsweiler, D., Ruthven, I., Jones, C.: Towards memory supporting personal information management tools. J. Am. Soc. Inf. Sci. Technol **58**(7), 924–946 (2007)
14. Geel, M.: Memsy: A Personal Resource Management Infrastructure. Ph.D. thesis, Diss. 23028, ETH Zurich (2015)
15. Geel, M., Norrie, M.C.: Memsy: Keeping Track of Personal Digital Resources Across Devices and Services. In: Proc. 19th Intl. Conf. on Theory and Practice of Digital Libraries (TPDL 2015), *LNCS*, vol 9316, pp. 71–83 (2015)
16. Henson, V.: An Analysis of Compare-by-Hash. In: Proc. 9th Conf. on Hot Topics in Operating Systems (HOTOS'03), pp. 13–18 (2003)

17. Jaballah, I., Cunningham, S.J., Witten, I.H.: Managing Personal Documents with a Digital Library. In: Proc. 9th European Conf. on Digital Libraries (ECDL 2005), pp. 195–206 (2005)
18. Jensen, C., Lonsdale, H., Wynn, E., Cao, J., Slater, M., Dietterich, T.G.: The Life and Times of Files and Information: A Study of Desktop Provenance. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'10), pp. 767–776 (2010)
19. Jones, W., Phuwanartnurak, A.J., Gill, R., Bruce, H.: Don't Take My Folders Away!: Organizing Personal Information to Get Things Done. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'05), Extended Abstracts, pp. 1505–1508 (2005)
20. Karger, D.R., Bakshi, K., Huynh, D., Quan, D., Sinha, V.: Haystack: A Customizable General-Purpose Information Management Tool for End Users of Semistructured Data. In: Proc. 2nd Biennial Conf. on Innovatie Data Systems Research (CIDR 2005), pp. 13–27 (2005)
21. Karlson, A.K., Smith, G., Lee, B.: Which Version is This?: Improving the Desktop Experience within a Copy-Aware Computing Ecosystem. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'11), pp. 2669–2678 (2011)
22. Ke, Y., Sukthankar, R.: PCA-SIFT: A More Distinctive Representation for Local Image Descriptors. In: Proc. 2004 IEEE Computer Society Conf. on Computer Vision (CVPR'04), pp. 506–513 (2004)
23. Lee, D.C., Ke, Q., Isard, M.: Partition Min-Hash for Partial Duplicate Image Discovery. In: Proc. 11th European Conf. on Computer Vision (ECCV 2010), *LNCS*, vol 6311, pp. 648–662 (2010)
24. Mikolajczyk, K., Schmid, C.: Scale and affine invariant interest point detectors. Int. J. Comput. Vis. **60**(1), 63–86 (2004)
25. Muniswamy-Reddy, K.K., Holland, D.A., Braun, U., Seltzer, M.I.: Provenance-Aware Storage Systems. In: Proc. USENIX Annual Technical Conference (ATC'06), pp. 43–56 (2006)
26. Muniswamy-Reddy, K.K., Macko, P., Seltzer, M.I.: Provenance for the Cloud. In: Proc. 8th USENIX Conf. on File and Storage Technologies (FAST'10), pp. 197–210 (2010)
27. Nebeling, M., Geel, M., Syrotkin, O., Norrie, M.C.: MUBox: Multi-User Aware Personal Cloud Storage. In: Proc. 33rd ACM Conf. on Human Factors in Computing Systems (CHI'15), pp. 1855–1864 (2015)
28. Oulasvirta, A., Sumari, L.: Mobile Kits and Laptop Trays: Managing Multiple Devices in Mobile Information Work. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'07), pp. 1127–1136 (2007)
29. Ritzdorf, H., Karapanos, N., Čapkun, S.: Assisted Deletion of Related Content. In: Proc. 30th Annual Computer Security Applications Conference (ACSAC'14), pp. 206–215 (2014)
30. Santosa, S., Wigdor, D.: A Field Study of Multi-device Workflows in Distributed Workspaces. In: Proc. ACM Intl. Joint Conf. on Pervasive and Ubiquitous Computing (UbiComp'13), pp. 63–72 (2013)
31. Sultana, S., Bertino, E.: A File Provenance System. In: Proc. 3rd ACM Conf. on Data and Application Security and Privacy (CODASPY'13), pp. 153–156 (2013)
32. Zacchi, A., Shipman, F.: Personal Environment Management. In: Proc. 11th European Conf. on Digital Libraries (ECDL 2007), pp. 345–356 (2007)